Improved Algebraic Cryptanalysis of QUAD, Bivium and Trivium via Graph Partitioning on Equation Systems

Kenneth Koon-Ho Wong[1], Gregory V. Bard[2]

[1] Information Security Institute, Queensland University of Technology, Brisbane QLD 4000, Australia
kk.wong@qut.edu.au
[2] Mathematics Department, Fordham University, The Bronx NY 10458, USA
bard@fordham.edu

Abstract. We present a novel approach for preprocessing systems of polynomial equations via graph partitioning. The variable-sharing graph of a system of polynomial equations is defined. If such graph is disconnected, then the corresponding system of equations can be split into smaller ones that can be solved individually. This can provide a tremendous speed-up in computing the solution to the system, but is unlikely to occur either randomly or in applications. However, by deleting certain vertices on the graph, the variable-sharing graph could be disconnected in a balanced fashion, and in turn the system of polynomial equations would be separated into smaller systems of near-equal sizes. In graph theory terms, this process is equivalent to finding balanced vertex partitions with minimum-weight vertex separators. The techniques of finding these vertex partitions are discussed, and experiments are performed to evaluate its practicality for general graphs and systems of polynomial equations. Applications of this approach in algebraic cryptanalysis on symmetric ciphers are presented: For the QUAD family of stream ciphers, we show how a malicious party can manufacture conforming systems that can be easily broken. For the stream ciphers Bivium and Trivium, we achieve significant speedups in algebraic attacks against them, mainly in a partial key guess scenario. In each of these cases, the systems of polynomial equations involved are well-suited to our graph partitioning method. These results may open a new avenue for evaluating the security of symmetric ciphers against algebraic attacks.

1 Introduction

There has been a long history of the use of graph theory in solving systems of equations. Graph partitioning techniques are applied to processes such as reordering variables in matrices to reduce fill-in for sparse systems [19, Ch. 7] and partitioning a finite element mesh across nodes in parallel computations [42]. These techniques primarily focus on linear systems over the real or complex numbers. In this paper, we apply similar graph theory techniques to systems of

multivariate polynomial equations, and develop methods of partitioning these systems into ones of smaller sizes via their "variable-sharing" graphs. These techniques are intended to work over any field, finite or infinite, but are particularly suited to GF(2) for use in algebraic cryptanalysis of symmetric ciphers. In most algebraic cryptanalysis, the symmetric ciphers are described by systems of polynomial equations over GF(2) or its algebraic extensions. The graph theory methods introduced in this paper can be used to improve the efficiency of solving these systems of equations, which would translate to a reduction of the security of these ciphers. This will be exemplified with the QUAD [9], Bivium [48] and Trivium [20] stream ciphers.

Computing the solution to a system of multivariate polynomial equations is an NP-hard problem [5, Ch. 3.9]. A variety of solution techniques have been developed for solving these polynomial systems over finite fields, such as linearization and XL [18], Gröbner bases, and resultants [6, Ch. 12], as well as recent ones such as SAT-solvers [7], Vielhaber's AIDA [50], Raddum-Semaev method [47], and the triangulation algorithm [35]. Over the real and complex numbers, numerical techniques are also known, but require the field to be ordered and complete—GF(2) is neither. The graph partitioning method introduced in this paper could be a novel addition to the variety of methods available, principally as a preprocessor.

From a multivariate polynomial system of equations, a variable-sharing graph is constructed with a vertex for each variable in the system, and an edge between two vertices if and only if those variables appear together in any equation in the system. Clearly, if the graph is disconnected, the system can be split into two separate systems of smaller sizes, and they can be solved for individually. However, even if the graph is connected, we show that it may be possible to disconnect the graph by eliminating a few variables by, for example, guessing their values when computing over a small finite field, and thereby splitting the remaining system. This suggests a divide-and-conquer approach to solving systems of equations. When the polynomial terms in the system of equations are very sparse, we show that the system can usually be reduced to a set of smaller systems, whose solutions can be computed individually in much less time. It should be noted that for large finite fields, and infinite fields as well, the technique of resultants can be used to achieve similar objectives [53].

In order for a partition of a system to be productive, the minimum number of variables should be eliminated, and the two subsystems must be approximately equal in size. This ensures that the benefit of partitioning the system is maximised. These conditions lead to the problem of finding a balanced vertex partition with a minimum-weight vertex separator on its variable-sharing graph, which is an NP-hard problem [31, 43]. Nevertheless, heuristic algorithms can often find near-optimal partitions efficiently [32].

In this paper, we offer two cryptographic applications of vertex partitioning arising from the algebraic cryptanalysis of stream ciphers, where both achieve positive results. First, we describe a method whereby a manufacturer of a sparse implementation of QUAD [9], a provably-secure infinite family of stream ciphers,

could "poison" the polynomial system in the cipher, and thereby enable messages transmitted with it to be read by the manufacturer. Second, we present an algebraic cryptanalysis of Trivium [20], a profiled stream cipher in the eSTREAM project, as well as its reduced versions Bivium-A and Bivium-B, and discuss the implications of graph partitioning methods on solving the corresponding systems of equations. Improvements to partial key guess attacks against Trivium and Bivium are observed.

Section 2 introduces the necessary background in graph theory and graph partitioning. Section 3 shows how a system of polynomial equations can be split into ones of smaller sizes using graph partitioning methods. Section 4 provides results for some partitioning experiments and analyses the feasibility of equation solving via graph partitioning methods. Section 5 presents the applications of graph partitioning methods on the algebraic cryptanalysis of QUAD, Bivium and Trivium. Conclusions will be drawn in Section 6. In Appendix A, we discuss the possibility for vertex connectivities of variable-sharing graphs becoming a security criterion for symmetric ciphers.

2 Preliminaries

Let $G = (V, E)$ be a graph with vertex set V and edge set E. Two vertices $v_i, v_j \in V$ are connected if there is a path from v_i to v_j through edges in E. A disconnected graph is a graph where there exists at least one pair of vertices that is not connected, or if the graph has only one vertex. A graph $G_1 = (V_1, E_1)$ with vertex set $V_1 \subseteq V$ and edge set $E_1 \subseteq E$ is called a subgraph of G. Given a graph G, subgraphs of G can be obtained by removing vertices and edges from G. Let $G = (V, E)$ be a graph with k vertices and l edges, such that $V = \{v_1, v_2, \ldots, v_{k-1}, v_k\}$, $E = \{(v_{i_1}, v_{j_1}), (v_{i_2}, v_{j_2}), \ldots, (v_{i_l}, v_{j_l})\}$. Removing a vertex v_k from V forms a subgraph $G_1 = (V_1, E_1)$ with $V_1 = \{v_1, v_2, \ldots, v_{k-1}\}$ and $E_1 = \{(v_i, v_j) \in E \,|\, v_k \notin \{v_i, v_j\}\}$. We call G_1 the subgraph of G induced by the vertex set $(V - \{v_k\})$.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two subgraphs of G. G_1, G_2 are considered disjoint if no vertices in G_1 are connected to vertices in G_2. Clearly, the condition $V_1 \cap V_2 = \emptyset$ is necessary but insufficient.

2.1 Graph Connectivity

The goal of partitioning a graph is to make the graph disconnected by removing some of its vertices or edges. The number of vertices or edges that needs to be removed to disconnect a graph are its vertex- or edge-connectivities respectively.

Definition 2.1. *The* vertex connectivity $\kappa(G)$ *of a graph G is the minimum number of vertices that must be removed to disconnect G.*

Definition 2.2. *The* edge connectivity $\lambda(G)$ *of G is the minimum number of edges that must be removed to disconnect G.*

Clearly, a disjoint graph has vertex connectivity zero. On the other extreme, a complete graph K_n, where all n vertices are connected to each other, has vertex connectivity $(n-1)$. The removal of all but one vertex from K_n results in a graph consisting of a single vertex, which is considered to be disconnected.

2.2 Graph Partitioning

The process of removing vertices or edges to disconnect a graph is called vertex partitioning or edge partitioning respectively. All non-empty graphs admit trivial vertex and edge partitions, where all connections to a single vertex are removed. This is obviously not useful for most applications. In this paper, we only consider balanced partitions with minimum-weight separators, in which a graph is separated into subgraphs of roughly equal sizes by removing as few vertices or edges as possible. More specifically, our primary focus is on balanced vertex partitions.

Definition 2.3. *Let $G = (V, E)$ be a graph. A vertex partition (V_1, C, V_2) of G is a partition of V into mutually exclusive and collectively exhaustive sets of vertices V_1, C, V_2, where V_1, V_2 are non-empty, and where no edges exist between vertices in V_1 and vertices in V_2. The removal of C causes the subgraphs induced by V_1 and V_2 to be disjoint, hence C is called the vertex separator.*

For a balanced vertex partition, we require V_1 and V_2 to be of similar size. For a minimum-weight separator, we also require that C be small. This is to ensure that the vertex partition obtained is useful for applications.

Definition 2.4. *Let $G = (V, E)$ be a graph, and (V_1, C, V_2) be a vertex partition of G with vertex separator C. If $\max(|V_1|, |V_2|) \leq \alpha|V|$, then G is said to have an α-vertex separator.*

The problem of finding α-vertex separators is known to be NP-hard [31, 43].

Definition 2.5. *Let $G = (V, E)$ be a graph. If (V_1, C, V_2) is a vertex partition of G, then define*

$$\beta = \frac{\max(|V_1|, |V_2|)}{|V_1| + |V_2|} = \frac{\max(|V_1|, |V_2|)}{|V| - |C|} = \frac{\alpha|V|}{|V| - |C|}$$

to be the balance of the vertex partition. Note further if $|C| \ll |V|$ then $\alpha \approx \beta$.

Suppose the balance of a vertex partition of G into (V_1, C, V_2) is β, then the partition also satisfies $\max(|V_1|, |V_2|) = \beta(|V_1| + |V_2|) \leq \beta|V|$, and hence the G has a β-vertex separator. Therefore, theorems that apply to α-vertex separators would also apply to vertex partitions with balance β. See [43] for more details of α-vertex separators. Several theorems governing the existence of α-vertex separators have been shown in [3, 27, 38, 41].

Figure 1 presents examples of balanced and unbalanced partitions, and their respective β values. The vertex separators C are circled, with the partitioned vertices V_1, V_2 outside. The removal of the vertices in the separators disconnects the graphs. For a balanced partition, β should be close to $1/2$.

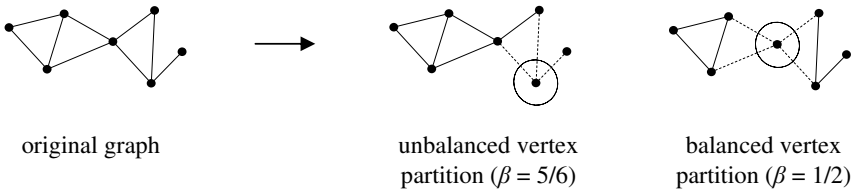

Fig. 1. Balanced and Unbalanced Vertex Partitions

2.3 Partitioning Algorithms and Software

While balanced partitioning is an NP-hard problem, a variety of heuristic algorithms have been found to be very efficient in finding near-optimal partitions.

One efficient scheme for balanced graph partitioning is called multilevel partitioning. Suppose a graph G_0 is to be partitioned. Firstly, G_0 "coarsened" progressively into simpler graphs $G_1, G_2, \ldots, G_r$ by contracting adjacent vertices. The process of choosing vertices for contraction is called matching. After reaching a graph G_r with the desired level of simplicity, a partitioning is performed. The result is then progressively refined back through the chain of graphs $G_{r-1}, G_{r-2}, \ldots, G_0$. At each refining step, a contracted vertices are expanded and partitioned. The output is then a partition of G_0. Details of multilevel partitioning can be found in [30, 32]. Examples of partitioning and refinement algorithms include the ones by Kerighan-Lin [34] and Fiduccia-Mattheyses [25].

Balanced edge partitioning is widely used in scientific and engineering applications, such as electric circuit design [49], parallel matrix computations [37], and finite element analysis [42]. Software packages are readily available for computing balanced edge partitions using a variety of algorithms [8, 11, 26, 29, 44, 45, 51]. On the other hand, balanced vertex partitioning has fewer applications, one of which being variable reordering in linear systems [19]. We are not aware of publicly available software that could be used for directly computing balanced vertex partitions with minimum-weight vertex separators.

This is also true for multilevel vertex-partitioning algorithms. Therefore, we have chosen to compute vertex partitionings through the use of the multilevel edge-partitioning software Metis [33] for our study. The Matlab interface Meshpart [28] to Metis is used to access the algorithms. It also contains a routine to convert an edge partition found by Metis to a vertex partition. We have also implemented an alternative greedy algorithm for this task. Both are used for the experiments in Section 4 and for the algebraic cryptanalysis of Trivium in Section 5.2.

Unless otherwise stated, from here on we will only consider the problem of balanced vertex partitioning with minimum-weight vertex separators (sometimes simply referred to as vertex partitioning or partitioning) and its applications to solving systems of multivariate polynomial equations.

3 Partitioning Polynomial Systems

In this section, our method for partitioning systems of multivariate polynomal equations by finding balanced vertex partitions of their variable-sharing graphs is described.

Definition 3.1. *Let F be the polynomial system*

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

of m polynomial equations in the variables $x_1, x_2, \ldots, x_n$. The variable-sharing *graph $G = (V, E)$ of F is obtained by creating a vertex $v_i \in V$ for each variable x_i, and creating an edge $(v_i, v_j) \in E$ if two variables x_i, x_j appear together (with non-zero coefficient) in any polynomial f_k.*

Example 3.1. Suppose we have the following quadratic system of equations over GF(2), where the variables $x_1, x_2, \ldots, x_5$ are known to take values in GF(2).

$$\begin{aligned} x_1x_3 + x_1 + x_5 &= 1 \\ x_2x_4 + x_4x_5 &= 0 \\ x_1x_5 + x_3x_5 &= 1 \\ x_2x_5 + x_2 + x_4 &= 0 \\ x_2 + x_4x_5 &= 1 \end{aligned} \tag{1}$$

The corresponding variable-sharing graph G and a balanced vertex partition is shown in Figure 2.

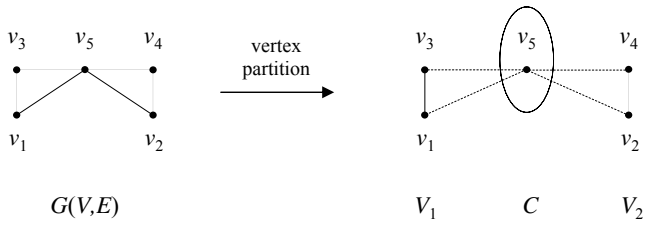

Fig. 2. Variable-sharing graph of the quadratic system (1) and a vertex partition.

The quadratic system can then be partitioned into two systems of equations with the common variable x_5 as follows.

$$\begin{aligned} x_1x_3 + x_1 + x_5 &= 1 \\ x_1x_5 + x_3x_5 &= 1 \end{aligned} \qquad \begin{aligned} x_2x_4 + x_4x_5 &= 0 \\ x_2x_5 + x_2 + x_4 &= 0 \\ x_2 + x_4x_5 &= 1 \end{aligned} \tag{2}$$

Since $x_5 \in \mathrm{GF}(2)$, we can substitute all possible values of x_5 into (1) and compute solutions to the reduced systems to give

$$\begin{aligned} x_5 = 0 &\Rightarrow \text{no solution} \\ x_5 = 1 &\Rightarrow (x_1, x_3) = (0, 1), (x_2, x_4) = (1, 0) \\ &\Rightarrow \boldsymbol{x} = (0, 1, 1, 0, 1) \end{aligned}$$

The solution obtained is the same as if we had directly computed the solution to the full system of equations. However, the systems have been reduced to having less than half the number of variables compared to the the original, at the cost of applying guesses to one variable.

This method of guessing and solving will be used for the algebraic cryptanalysis of the Trivium stream cipher in Section 5. For simplicity, from here on we might use the terms for variables and vertices interchangeably to denote the variables in the polynomial systems of equations and their corresponding vertices in the variable-sharing graphs, provided there are no ambiguities.

4 Graph Partitioning Experiments

To evaluate the practicality of partitioning large systems of equations, experiments have been performed on random graphs of different sizes resembling typical variable-sharing graphs. These experiments were run on a Pentium M 1.4 GHz CPU with 1 GB of RAM using the Meshpart [28] Matlab interface to the Metis [33] partitioning software.

Definition 4.1. *Let $G = (V, E)$ be a graph. The* degree $\deg(v)$ *of a vertex $v \in V$ is the number of edges $e \in E$ incident upon (connecting to) v.*

Definition 4.2. *The* density $\rho(G)$ *of a graph $G = (V, E)$ is the ratio of the number of edges $|E|$ in G to the maximum possible number $\frac{1}{2}|V|(|V| - 1)$ of edges in G.*

In each experiment, random graphs $G = (V, E)$ are generated, each with prescribed number of vertices $|V|$, number of edges $|E|$, and average degree d of its vertices. Their densities ρ are also computed. For each graph, a vertex partition is performed to give (V_1, C, V_2), where C is the vertex separator. The balance measure β is then computed, and the time required is also noted. Some experimental results are shown in Table 1.

It can be observed from Table 1 that the graph of β is likely to be correlated with the average degree d of the graphs. Small vertex separators can be obtained when the number of edges is a small factor of the number of vertices. At $d = 16$, the value of β is near its upper bound of 1, which means that those partitions are unlikely to be useful. Since the maximum number of edges for a graph of size n is $O(n^2)$, the edge density must be smaller with a larger graph for practical partitions. This is a reasonable assumption for polynomial systems, since certain sparse systems have only a small number of variables in each equation, regardless

Table 1. Vertex Partitioning Experiments

$\|V\|$	$\|E\|$	ρ	d	$\|C\|$	$\|V_1\|$	$\|V_2\|$	β	Time
64	64	0.0308	2	5	31	28	0.5254	61.26 ms
64	128	0.0615	4	15	30	19	0.6122	63.06 ms
64	256	0.1231	8	26	28	10	0.7368	80.95 ms
64	512	0.2462	16	32	3	29	0.9063	67.36 ms
128	128	0.0155	2	7	64	57	0.5289	64.80 ms
128	256	0.0310	4	28	60	40	0.6000	66.73 ms
128	512	0.0620	8	55	45	28	0.6164	63.27 ms
128	1024	0.1240	16	62	63	3	0.9545	83.51 ms
1024	1024	0.0020	2	51	508	465	0.5221	74.58 ms
1024	2048	0.0039	4	222	482	320	0.6010	90.05 ms
1024	4096	0.0078	8	418	355	251	0.5858	113.66 ms
1024	8192	0.0156	16	509	511	4	0.9922	168.55 ms
4096	4096	0.0005	2	183	2039	1874	0.5211	122.48 ms
4096	8192	0.0010	4	877	1903	1316	0.5912	175.20 ms
4096	16384	0.0020	8	1697	1539	860	0.6415	289.24 ms
4096	32768	0.0039	16	2037	2047	12	0.9942	548.75 ms

of the total number of variables in the system. This fact is true for the case of Trivium in Section 5. One could say that the weight or length (i.e. the number of terms) of the polynomials should be short for useful cuts to be guaranteed.

It is also noted that the time required to compute vertex partitions are quite short for the practical graph sizes considered. Therefore, we can safely assume that the time complexity of the partitioning algorithm is negligible compared to that required to solve the partitioned equation systems, which in the worst case is exponential in the number of variables and the maximum degree.

5 Applications to Algebraic Cryptanalysis

A bit-based stream cipher can be thought of as an internal state $s \in \mathrm{GF}(2)^n$ and two maps, $g : \mathrm{GF}(2)^n \to \mathrm{GF}(2)^n$ and $f : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$. At each clock tick, $s_{t+1} = g(s_t)$, and so g can be called the state-update function. Also at each clock tick $f(s_t) = z_t$ is outputted, and this is called the keystream. Given plaintext bits $p_1, p_2, \ldots$, the stream cipher encrypts them using the keystream into ciphertext bits $c_1, c_2, \ldots$ via $c_t = p_t + z_t$, with the addition being over GF(2).

At the start of an encryption, a key-initialisation phase would take place, whereby a secret key k and a known initialisation vector IV are used to set s to its secret initial state s_0. The cipher then begins its keystream generation phase, and outputs a series of keystream bits $z_1, z_2, \ldots$, as explained above.

By the Universal Mapping Theorem [6, Th. 72], since f and g are maps from finite sets to finite sets, we know that they can be written as polynomial systems of equations over any field, but GF(2) is most useful to us. Since stream ciphers are traditionally designed for implementation in digital circuits, where

there are economic motivations to keep the gate-count low, f and g can often be represented by very simple polynomial functions. Then, if both p_t and c_t are known for enough timesteps, one can write a system of equations based on $z_t = p_t + c_t$ using f and g. This forms the foundation of algebraic cryptanalysis.

To perform an algebraic cryptanalysis of the stream cipher, the cipher is first described as a system of equations. Its variables usually correspond to the bits in the key k or the initial state s_0. If the variables are from k, solving the system is called "key recovery", and the cipher is immediately broken. If the variables are from s_0, solving the system is called "state recovery", and the key could be derived from the solution, whose difficulty depends on the specific cipher design.

Every attack on every cipher has its nuances, and so above description is necessarily vague. For an overview of algebraic cryptanalysis, see [6]. For techniques of algebraic cryptanalysis on specific types of ciphers, see [17, 16, 15, 2, 54]. Some uses of graph theory for algebraic attacks can also be found in [48, 52]. In this section, two applications of our equation partitioning to algebraic cryptanalysis are presented. Firstly, we show a malicious use of the stream cipher QUAD [9]. Then, we describe and perform an algebraic cryptanalysis to the stream cipher Trivium [20] and its variants Bivium-A and Bivium-B [48]. We discuss only the equations arising from the cipher, and refer the reader to the respective references of these ciphers for their design and implementation details.

5.1 QUAD

The stream-cipher family QUAD is given in [9]. The security of QUAD is based on the Multivariate Quadratic (MQ) problem. The heart of the cipher is a random system of kn quadratic equations in n variables over a finite field $\mathrm{GF}(q)$. Usually, we have $q = 2$, but implementations with $q = 2^s$ have also been discussed [55]. This system of equations is not secret, but publicly known, and there are criteria for these equations, such as those relating to rank, which we omit here. In a different context, QUAD has been analyzed in [55, 4], and [6, Ch. 5.2].

Equations of QUAD The authors of QUAD recommend $k = 2$ and $n \geq 160$, so it is assumed that we have a randomly generated system of $2n = 320$ equations in $n = 160$ unknowns. The system is to be drawn uniformly from all those possible, which is to say that the coefficients can be thought of as generated by fair coins.

Each quadratic equation is a map $\mathrm{GF}(2)^n \to \mathrm{GF}(2)$, so the first set of n equations form a map $\mathrm{GF}(2)^n \to \mathrm{GF}(2)^n$ called f_1, and the second set of n equations also form a map of the same dimensions called f_2. The internal state is a vector s of 160 bits. The first 160 equations are evaluated at s, and the resulting vector $f_1(s_t) = s_{t+1}$ becomes the new state. The second 160 equations are evaluated to become the output of that timestep $z_t = f_2(s_t)$. The vector z_t is added to the next n bits of the plaintext p_t over $\mathrm{GF}(2)$, and is transmitted as the ciphertext $c_t = p_t + z_t$. (Each bit is added independently, without carries.) There is also an elaborate setup stage which maps the secret key and an initialization vector to the initial state s_0.

Finding a pre-image under the maps f_1, f_2 i.e. finding s_i given s_{i+1} and z_i, is equivalent to solving a quadratic system of $2n$ equations in n unknowns, and is NP-hard [5, Ch. 3.9]. This is further complicated by the fact that the adversary would not have s_{i+1}, but rather only $z_i + p_i$.

Given a known-plaintext scenario, where the attacker knows both the plaintext $p_1, p_2 \ldots, p_n$ and ciphertext $c_1, c_2, \ldots, c_n$, one can write the following system of equations.

$$\begin{aligned} c_1 + p_1 &= z_1 = f_2(s_1) \\ c_2 + p_2 &= z_2 = f_2(s_2) = f_2(f_1(s_1)) \\ &\vdots = \vdots \\ c_t + p_t &= z_t = f_2(s_t) = f_2(\underbrace{f_1(f_1(f_1(\cdots f_1(}_{t-1 \text{ times}} s_1) \cdots))))\end{aligned}$$

The interesting fact here is that $f_2(f_1(f_1(\cdots f_1(s_1)\cdots))))$ and higher iterates might be quite dense even if f_1 is sparse. The authors of QUAD have excellent security arguments when the polynomial system is generated by fair coins. In this case, the variable-sharing graph of the cipher will have density close to that of a complete graph, and our graph partition method will have little use. However, it will have on average 6440.5 monomials per equation or roughly 2 million in the system, which would require a large gate count or would be slow in software. Thus, in their conference presentation, the authors of QUAD mention that a slightly sparse f might still be secure against algebraic attacks, because of the repeated iterations and the general difficulty of the MQ problem. Nevertheless, in this case, if a sparse system can be chosen such that it contains a small balanced vertex separator, then the cipher can be made insecure by a malicious attacker as follows.

Poisoned Equations and QUAD One could imagine the following scenario, which is inspired by Jacques Patarin's system "Oil and Vinegar" [36]. A malicious manufacturer does not generate the system at random, but rather creates a system that is sparse and has vertex connectivity of 20, for some vertex partition with $\beta \approx 0.6$. Our experiments in Section 4 show that this is a feasible partition. The malicious manufacturer would claim that the system is sparse for efficiency reasons and it might have a considerably faster encryption throughput than a QUAD system with quadratic equations generated by fair coins.

Some separator of 20 vertices divides the variable sharing graph into roughly 56 and 84 vertices. This means that an attacker would need only to know the plaintext and ciphertext of one 160-bit sequence, and solve the equation

$$f_2(\underbrace{f_1(f_1(\cdots f_1(f_1(}_{i-1 \text{ times}} s_1)) \cdots))) = p_t + c_t$$

.

For any guess of the key, this would be solving 56 equations in 56 unknowns and 84 equations in 84 unknowns. Such a problem is certainly trivial for a SAT-solver, as shown in [7], [5, Ch. 3] and [6, Ch. 7]. Only 2^{20} such systems would need to be solved, and with a massive parallel network, such as BOINC [1], this would be feasible [12], although experiments would be required for confirmation.

Remedy to Poisoned Systems for QUAD While finding a balanced vertex partition of a graph G is NP-hard, calculating the vertex connectivity $\kappa(G)$ is easier. If $\kappa(G) > 80$, for example, then there is no vertex partition, balanced or otherwise, with fewer than 80 vertices in the vertex separator. Then, by calculating $\kappa(G)$, a manufacturer of QUAD could prove that they are not poisoning the quadratic system. There are also techniques to generate functions with verifiable randomness [14], which could be used to construct polynomial systems of equations for QUAD, such that they are provably not poisoned.

5.2 Trivium

Trivium [20] is a bit-based stream cipher in the eSTREAM project portfolio for hardware implementation with an 80-bit key, 80-bit initialization vector, and a 288-bit internal state. As at the end of the eSTREAM project, after three phases of expert and community reviews, no feasible attacks faster than an exhaustive key search on the full implementation of Trivium were found. However, Trivium without key initialisation, as well as its reduced versions Bivium-A and Bivium-B with a 177-bit internal state, admit attacks faster than exhaustive key search. Cryptanalytic results on Trivium and Bivium have been presented in [10, 21, 22, 23, 39, 40, 46, 50].

Equation Construction The equations governing keystream generation from the initial state s_0 can be found in [20] for Trivium and [48] for Bivium. In the algebraic cryptanalysis presented in this paper, we do not consider the initialisation phase from the key k and initialisation vector IV, and hence we are performing state recovery of the cipher.

Trivium can be described as a system of 288 multivariate polynomial equations in 288 variables, but we found that this is too dense for partitioning to be useful. Instead, we use the system of quadratic equations presented in [48], which contains more variables, but is very sparse. The quadratic system of Trivium consists of 954 sparse quadratic equations in 954 variables, and observed keystream from 288 clocks. Similarly, the polynomial system of Bivium-A and Bivium-B consists of 399 sparse quadratic equations in 399 variables, and observed keystream from 177 clocks. We attempt to solve these equations via partitioning.

Equation Partitioning The sparse quadratic equations for Trivium and Bivium are constructed as per [48], and their variable-sharing graphs are then computed.

Figure 3 shows the adjacency matrix for the variable-sharing graph of Trivium. The sparsity of this matrix appears promising for a reasonable partition. Graphs for Bivium are of similar sparsity.

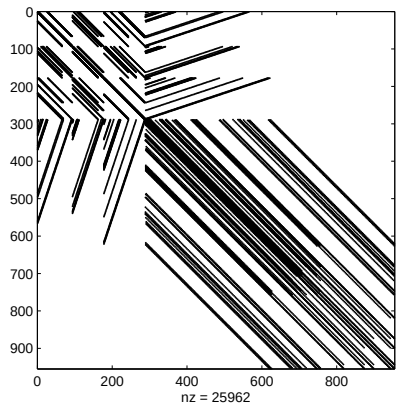

Fig. 3. Graph Adjacency Matrix of Trivium Equations

Partitioning these variable-sharing graphs $G = (V, E)$ into vertex sets V_1, V_2 and vertex separator C with [33] as in Section 4 gives the results shown in Table 2. From these results, it seems that both of the Bivium ciphers admit very

Table 2. Partitioning Equations of Bivium-A, Bivium-B and Trivium

Cipher	State Size	Number of Variables	$\lvert C\rvert$	$\lvert V_1\rvert$	$\lvert V_2\rvert$	β
Bivium-A	177	399	96	156	147	0.5149
Bivium-B	177	399	14	128	127	0.5020
Trivium	288	954	288	476	190	0.7147

balanced partitions, whereas Trivium did not. Nevertheless, using our alternative greedy algorithm for converting edge partitions from the Metis software to vertex partitions, we were able to find a balanced partition for Trivium with $|C| = 295$ and $\beta \approx 0.5$, at the cost of having a larger vertex separator.

The sizes of the vertex separators C are the number of variables that must be eliminated to separate the systems into two. In algebraic attacks, this corresponds to the number of variables whose values are to be discovered or guessed at a complexity of $2^{|C|}$. The process of guessing certain bits in order to find a solution is called partial key guessing. If the guessed bits are correct, then solving the remaining system would lead to the solution.

Table 3. Partial Key Guessing on Trivium and Bivium

Cipher	All Guesses in $\|C\|$	n	m	q	Time	Memory
Bivium-A	No	24	422	193	26 s	42 MB
Bivium-A	No	20	421	200	195 s	234 MB
Bivium-A	No	18	417	203	2558 s	843 MB
Bivium-A	Yes	18	417	195	80 s	127 MB
Bivium-A	Yes	16	415	201	1101 s	751 MB
Bivium-A	Yes	14	413	202	2023 s	1200 MB
Bivium-B	No	80	479	143	392 s	1044 MB
Bivium-B	No	78	477	146	740 s	1044 MB
Bivium-B	No	76	475	141	1213 s	1044 MB
Bivium-B	Yes	70	469	132	12 s	62 MB
Bivium-B	Yes	66	465	136	623 s	546 MB
Bivium-B	Yes	62	461	141	3066 s	1569 MB
Trivium	No	280	1333	329	13 s	80 MB
Trivium	No	272	1224	343	155 s	554 MB
Trivium	No	264	1217	344	594 s	1569 MB
Trivium	Yes	178	1130	499	18 s	596 MB
Trivium	Yes	176	1127	499	4511 s	1875 MB
Trivium	Yes	174	1126	501	10543 s	3150 MB

For Trivium, the separator size is at least the internal state size, so a partition on the equation system is not useful, as guessing the variables in the separator is as costly as an exhaustive search on the initial state. For Bivium and Bivium-A, the separator sizes are less than the internal state size, but larger than the key size of 80-bits. This means that the time complexity of partial key guessing on all bits of the separators would be higher than that of a brute-force search on the key, but lower than a brute-force search on the initial state.

Partial Key Guessing and Perforated Systems However, we can attempt to guess fewer bits than the size of the separator C. The remaining system would not be separated, but it can still be solved. Since it is close to being partitioned, we will call such systems "perforated". We have discovered by experiments that partial key guesses on subsets of bits in C provide significant advantages over those on random bits, in that the reduced polynomials systems are much easier to solve. The experiments were performed using Magma 2.12 [13] with its implementation of the Gröbner basis algorithm F_4 [24] for solving the reduced polynomial systems. The results are shown in Table 3, where n is the number of bits guessed, m is the number of equations resulting from the guess, with q of them being quadratic. Correct guesses are always used to reduce the polynomial systems, which means that the time and memory use presented are for solving the entire system arriving at a unique solution. All values are averaged over 10 individual runs.

The experimental results show that the time required for partial key guessing on n bits is reduced significantly if those bits are taken from the separator. This means that, by finding partitions to the system of equations, we have reduced the resistance of these ciphers to algebraic cryptanalysis, since a feasible partial-key-guess attack can potentially be launched on fewer bits with this extra information. For example, with Bivium-B, the time to compute a solution by guessing 78 bits randomly is roughly equivalent to that by guessing 66 bits in the separator. Hence, the time complexity for an attack on Bivium-B is reduced from $2^{78}T_B$ to $2^{66}T_B$ with the use of the separator, where T_B denotes the time complexity required to compute a solution to a reduced system of Bivium-B. For Trivium, the improvement is even more pronunced. The time complexity could be reduced from $2^{280}T_C$ to $2^{178}T_C$, which T_C denotes the time complexity required to compute a solution to a reduced system of Trivium.

In an actual algebraic attack, many of the guesses will result in inconsistent equations with no solutions, which can be checked and discarded easily. This means that the time required to process a guess is at most T_B or T_C. A full attack attempt was launched on Bivium-A with a partial key guess on 20 bits in its separator. About $200,000$ guesses of out the possible 2^{20} were made, with each guess taking on average about 0.15 seconds to process. This is much faster than the 45 seconds required from the experimental results to process a correct guess to completion.

Although the time complexity for the algebraic attack on Trivium is significantly reduced through our partitioning method, it is still much higher than that of exhaustive key search, which is 2^{80}. On the other hand, this method applied on Bivium-A and Bivium-B may be faster than exhaustive key search, depending on the time complexity of solving the reduced equation systems.

A Bit-Leakage Attack There is another scenario whereby the graph partitioning would provide an advantage to algebraic cryptanalysis. Suppose by some means, accidental or deliberate, some bits of the internal state of a cipher could be leaked to an attacker. This would occur in a side-channel attack setting. If the attacker could control which bits are leaked, then the best choices would be those variables in the separator. If all bits in the separator are leaked, then the equation system is immediately split into two, and the time complexity of solving for the remaining bits is significantly reduced. If only some bits in the separator are leaked, we have shown in the earlier experiments that this leads to faster attacks than if an equal amount of random bits are leaked. This also means that if bits can be leaked from the separator, fewer of them would be needed before the system of equations can be solved in a reasonable time, compared to the case where bits are leaked randomly.

6 Conclusions

In this paper, the concept of a variable-sharing graph of a system of polynomial equations was defined. It has been shown that this concept can be used to break

systems of polynomial equations into pieces, which can be solved separately, provided that the graph has a vertex partition satisfying various requirements: namely that the vertex separator should be small, and the partition should be balanced. We also presented methods for finding the partition, and methods for using the partition to solve polynomial systems of equations over GF(2).It has been shown that balanced vertex partitions are feasible to obtain for some sparse systems of polynomial equations. Experiments on random graphs of reasonable size and sparsity, resembling variable-sharing graphs of equation systems, have been performed.

The practicality of this partitioning technique has been demonstrated in the algebraic cryptanalysis of the stream cipher Trivium and its reduced versions, where we have found balanced partitions of useful sizes. These partitions provide information for launching more effective algebraic attacks with partial key guessing, and improves the attack time by at least a few orders of magnitude. Furthermore, we show how the partitioning technique can be used to poison the provably secure stream cipher QUAD, so that a malicious manufacturer can recover the keystream much more efficiently.

As discussed earlier, this paper has provided a novel technique for preprocessing large sparse systems of equations, which could be used together with popular techniques such as Gröbner basis methods to significantly reduce the time for computing solutions to these systems. It has also been shown that this technique provides improvements to algebraic cryptanalysis, and further research into this area is warranted, since there may be security implications for other ciphers that are susceptible to this technique.

References

[1] BOINC: Berkeley Open Infrastructure for Network Computing, `http://boinc.berkeley.edu/`

[2] Al-Hinai, S., Batten, L., Colbert, B., Wong, K.K.H.: Algebraic attacks on clock-controlled stream ciphers. In: Batten, L.M., Safavi-Naini, R. (eds.) 11th Australasian Conference on Information Security and Privacy — ACISP 2006. Lecture Notes in Computer Science, vol. 4058, pp. 1–16. Springer, Melbourne, Australia (2006)

[3] Alon, N., Semour, P., Thomas, R.: A separator theorem for graphs with an excluded minor and its applications. Journal of the American Mathematical Society 3(4), 801–808 (Oct 1990)

[4] Arditti, D., Berbain, C., Billet, O., Gilbert, H., Patarin, J.: QUAD: Overview and recent developments. In: Biham, E., Handschuh, H., Lucks, S., Rijmen, V. (eds.) Symmetric Cryptography. Dagstuhl Seminar Proceedings, vol. 07021. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2007)

[5] Bard, G.V.: Algorithms for solving linear and polynomial systems of equations over finite fields with applications to cryptanalysis. Ph.D. thesis, Department of Applied Mathematics and Scientific Computation, University of Maryland at College Park (Aug 2007), available at `http://www.math.umd.edu/~bardg/bard_thesis.pdf`

[6] Bard, G.V.: Algebraic Cryptanalysis. Springer (2009)
[7] Bard, G.V., Courtois, N., Jefferson, C.: Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-Solvers. Cryptology ePrint Archive, Report 2007/024 (2007), http://eprint.iacr.org/2007/024.pdf
[8] Baños, R., Gil, C., Ortega, J., Montoya, F.G.: Multilevel heuristic algorithm for graph partitioning. In: Applications of Evolutionary Computing, Lecture Notes in Computer Science, vol. 2611. Springer (2003)
[9] Berbain, C., Gilbert, H., Patarin, J.: QUAD: A practical stream cipher with provable security. In: Vaudenay, S. (ed.) Advances in Cryptology - Eurocrypt 2006. Lecture Notes in Computer Science, vol. 4004, pp. 109–128. Springer (2006)
[10] Bernstein, D.: Response to slid pairs in Salsa20 and Trivium. Tech. rep., The University of Illinois at Chicago (2008), http://cr.yp.to/snuffle/reslid-20080925.pdf
[11] Berry, J., Dean, N., Goldberg, M., Shannon, G., Skiena, S.: Graph computation with LINK. Software: Practice and Experience 30, 12851302 (2000)
[12] Black, M., Bard, G.: SAT over BOINC: Satisfiability solving over a volunteer grid. Draft Article, Submitted for Publication (2010), http://www.math.umd.edu/~bardg/publications.html
[13] Bosma, W., Cannon, J., Playoust, C.: The MAGMA algebra system. I. The user language. Journal of Symbolic Computation 24(3-4), 235–265 (1997)
[14] Chase, M., Lysyanskaya, A.: Simulatable vrfs with applications to multi-theorem nizk. In: Advances in Cryptology - CRYPTO 2007. Lecture Notes in Computer Science, vol. 4622, pp. 303–322. Springer (2007)
[15] Cho, J.Y., Pieprzyk, J.: Algebraic attacks on SOBER-t32 and SOBER-t16 without stuttering. In: Roy, B., Meier, W. (eds.) Fast Software Encryption. Lecture Notes in Computer Science, vol. 3017, pp. 49–64. Springer, Delhi, India (2004)
[16] Courtois, N.: Algebraic attacks on combiners with memory and several outputs. In: Park, C., Chee, S. (eds.) Information Security and Cryptology - ICISC 2004. Lecture Notes in Computer Science, vol. 3506. Springer, Seoul, Korea (2004)
[17] Courtois, N., Meier, W.: Algebraic attacks on stream cipher with linear feedback. In: Biham, E. (ed.) Advances in Cryptology - Eurocrypt 2003. vol. 2656. Springer, Warsaw, Poland (2003)
[18] Courtois, N., Shamir, A., Patarin, J., Klimov, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) Advances in Cryptology—Proc. of EUROCRYPT. Lecture Notes in Computer Science, vol. 1807, pp. 392–407. Springer-Verlag (2000)
[19] Davis, T.A.: Direct methods for sparse linear systems, Fundamentals of Algorithms, vol. 2. SIAM, Philadelphia, USA (2006)
[20] De Cannière, C., Preneel, B.: Trivium specifications. Tech. rep., Katholieke Universiteit Leuven (2007), http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf
[21] Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Advances in Cryptology - Eurocrypt 2009. Lecture Notes in Computer Science, vol. 5479, pp. 278–299. Springer (2009)
[22] Eén, N., Sörensson, N.: Minisat — a SAT solver with conflict-clause minimization. In: Bacchus, F., Walsh, T. (eds.) Proc. Theory and Applications of Satisfiability Testing (SAT'05). Lecture Notes in Computer Science, vol. 3569, pp. 61–75. Springer-Verlag (2005)

[23] Eibach, T., Pilz, E., Völkel, G.: Attacking Bivium using SAT solvers. In: Büning, H.K., Zhao, X. (eds.) Theory and Applications of Satisfiability Testing (SAT '08). Lecture Notes in Computer Science, vol. 4996, pp. 63–76. Springer-Verlag (2008)
[24] Faugère, J.C.: A new efficient algorithm for computer Gröbner bases (f_4). Journal of Pure and Applied Algebra 139, 61–88 (1999)
[25] Fiduccia, C., Mattheyses, R.: A linear time heuristic for improving network partitions. In: 19th ACM/IEEE Design Automation Conference. pp. 175–181 (1982)
[26] Fremuth-Paeger, C.: Goblin: A graph object library for network programming problems (2007), http://goblin2.sourceforge.net/
[27] Gilbert, J.R., Hutchinson, J.P., Tarjan, R.E.: A separation theorem for graphs of bounded genus. Journal of Algorithms 5, 391–407 (1984)
[28] Gilbert, J.R., Teng, S.H.: Meshpart: Matlab mesh partitioning and graph separator toolbox (2002), `http://www.cerfacs.fr/algor/Softs/MESHPART`
[29] Hendrickson, B., Leland, R.: The Chaco user's guide: Version 2.0. Tech. Rep. SAND94-2692, Sandia National Laboratories (1994)
[30] Hendrickson, B., Leland, R.: A multilevel algorithm for partitioning graphs. In: 1995 ACM/IEEE Supercomputing Conference. ACM (1995)
[31] Johnson, D.S.: The NP-completeness column: An on-going guide. J. Algorithms 8, 438–448 (1987)
[32] Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 20(1), 359–392 (1999)
[33] Karypis, G., et al.: Metis — Serial graph partitioning and fill-reducing matrix ordering (1998), `http://glaros.dtc.umn.edu/gkhome/views/metis/`
[34] Kernighan, B., Lin, S.: An efficient heuristic procedure for partitioning graphics. Bell Systems Technical Journal 49, 291–307 (1970)
[35] Khovratovich, D., Biryukov, D., Nikolic, I.: Speeding up collision search for byte-oriented hash functions. In: Advances in Cryptology - Crypto 2009. Lecture Notes in Computer Science, vol. 5473, pp. 164–181. Springer (2009)
[36] Kipnis, A., Patarin, J., Goubin, L.: Unbalanced oil and vinegar signature schemes. In: EUROCRYPT. pp. 206–222 (1999)
[37] Kumar, V., Grama, A., Gupta, A., Karypis, G.: Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings Publishing Company, Redwood City, CA (1994)
[38] Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics 36(2), 177–189 (Apr 1979)
[39] Maximov, A., Biryukov, A.: Two trivial attacks on Trivium. In: Adams, C.M., Miri, A., Wiener, M.J. (eds.) Proc. Selected Areas in Cryptography (SAC07). Lecture Notes in Computer Science, vol. 4876, pp. 36–55. Springer-Verlag (2007), available from `http://eprint.iacr.org/2007/021`
[40] McDonald, C., Charnes, C., Pieprzyk, J.: An algebraic analysis of Trivium ciphers based on the boolean satisfiability problem. Cryptology ePrint Archive, Report 2007/129 (2007), `http://eprint.iacr.org/2007/129`. Presented at the International Conference on Boolean Functions: Cryptography and Applications (BFCA2008)
[41] Menger, K.: Zur allgemeinen Kurventheorie. Fundamenta Mathematicae 10, 96–115 (1927)
[42] Miller, G.L., Teng, S.H., Thurston, W., Vavasis., S.A.: Automatic mesh partitioning. In: George, A., Gilbert, J., , Liu, J. (eds.) Graph Theory and Sparse Matrix Computation. The IMA Volumes in Mathematics and its Application, vol. 56, pp. 57–84. Springer (1993)

[43] Müller, R., Wagner, D.: α-vertex separator is NP-hard even for 3-regular graphs. J. Computing 46, 343–353 (1991)
[44] Pellegrini, F., Roman, J.: SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In: HPCN'96. LNCS, vol. 1067, pp. 493–498. Springer, Brussels, Belgium (1996)
[45] Preis, R., Diekmann, R.: The PARTY partitioning-library, user guide - version 1.1. Tech. Rep. tr-rsfb-96-024, University of Paderborn (1996)
[46] Priemuth-Schmid, D., Biryukov, A.: Slid pairs in Salsa20 and Trivium. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) Progress in Cryptology—INDOCRYPT'08. Lecture Notes in Computer Science, vol. 5365, pp. 1–14. Springer-Verlag (2008)
[47] Raddum, H., Semaev, I.: New technique for solving sparse equation systems. Cryptology ePrint Archive, Report 2006/475 (2006), `http://eprint.iacr.org/2006/475`
[48] Raddum, H.: Cryptanalytic results on Trivium. Tech. Rep. 2006/039, The eSTREAM Project (27 March 2006), `http://www.ecrypt.eu.org/stream/papersdir/2006/039.ps`
[49] Schweikert, D.G., Kernighan, B.W.: A proper model for the partitioning of electrical circuits. In: 9th workshop on Design automation. pp. 57–92. ACM (1972)
[50] Vielhaber, M.: Breaking One.Fivium by AIDA an algebraic IV differential attack. Cryptology ePrint Archive, Report 2007/413 (2007), `http://eprint.iacr.org/2007/413`
[51] Walshaw, C., Cross, M.: JOSTLE: Parallel Multilevel Graph-Partitioning Software - An Overview. Tech. rep., Civil-Comp Ltd. (2007)
[52] Wong, K.K.H.: Application of Finite Field Computation to Cryptology: Extension Field Arithmetic in Public Key Systems and Algebraic Attacks on Stream Ciphers. PhD Thesis, Information Security Institute, Queensland University of Technology (2008)
[53] Wong, K.K.H., Bard, G., Lewis, R.: Partitioning multivariate polynomial equations via vertex separators for algebraic cryptanalysis and mathematical applications. Draft Article (2008), `http://www.math.umd.edu/~bardg/publications.html`
[54] Wong, K.K.H., Colbert, B., Batten, L., Al-Hinai, S.: Algebraic attacks on clock-controlled cascade ciphers. In: Barua, R., Lange, T. (eds.) Progress in Cryptology - Indocrypt 2006. vol. 4329, pp. 32–47. Springer, Kolkata, India (2006)
[55] Yang, B.Y., Chen, O.C.H., Bernstein, D.J., Chen, J.M.: Analysis of QUAD. In: Fast Software Encryption. Lecture Notes in Computer Science, vol. 4593, pp. 290–308. Springer (2007)

A New Criterion for Symmetric Ciphers?

In this paper, we have shown that the graph partitioning method works particularly well when the equation systems describing these ciphers are sparse. This, in turn, means that the vertex connectivities of their variable-sharing graphs are low. Therefore, for maximum protection against an algebraic attack of this kind, a cipher should be designed such that its variable-sharing graph is close to being a complete graph, and hence does not admit any useful balanced vertex partition. The vertex connectivity could become the measure of this criterion, although care must be taken to account for the effects of variable relabelling techniques. Further research would be needed to refine this possible new criterion.