Partitioning Multivariate Polynomial Equations via Vertex Cuts for Algebraic Cryptanalysis and other Applications

Kenneth Koon-Ho Wong, Gregory V. Bard and Robert H. Lewis

Abstract. The variable-sharing graph of a polynomial system of equations has one vertex for each variable, and an edge between two variables if and only if those variables appear together in at least one equation. If this graph is disconnected, then the system is actually two separate systems that can be solved individually. This can provide a huge speed-up, but is unlikely to occur either randomly or in applications. However, it may be the case that deleting a small number of vertices c disconnects the variable-sharing graph in a balanced fashion, so that the ratio of the sizes of the larger and smaller components is roughly less than two. If this is the case, then we demonstrate two techniques, one for small fields and one for large fields, to split and solve the system. For small finite fields GF(q), we simply iterate through all possible q^c guesses of the c variables, and solve the separated systems. If the field is infinite or finite but large (i.e. has more than roughly five members), we separate the system with resultants. We present two methods for detecting this condition and identifying the c variables.

First, when c is small, or when $|V|$ is small, one can run a large series of depth-first searches. Alternatively, when c or $|V|$ is large, we show how to use off-the-shelf balanced edge cut determining software to generate a "reasonable" vertex cut. Also, we state a condition for a system of equations to be immune to this approach. We identify a security criterion for ciphers that is based on this condition, and apply it to a discussion of QUAD, a provably secure stream cipher based on GF(2) polynomial systems, and to HB, an authentication mechanism based on GF(2) linear algebra. Lastly, we present experiments which show that this condition can occur in practice, with very sparse polynomial systems, and in applications to Euclidean Geometry and Game Theory.

Mathematics Subject Classification (2000). 05C90, 11T71, 68R10, 94A60, 14G50.

Keywords. Balanced vertex cut, Graph Partitioning, Polynomial System of Equations, QUAD, Resultants, Algebraic Cryptanalysis.

1. Introduction

Graph Theory has a long history of applications to equation solving. In solving linear systems, it has been used to reorder variables to reduce fill-in for sparse systems. It has been used to split very sparse linear systems of equations into smaller unrelated systems. In this paper, we apply graph theory methods to systems of multivariate polynomial equations, and develop methods of partitioning through finding vertex separators in the corresponding "variable-sharing" graphs.

A "variable sharing" graph has a vertex for each underlying variable in the system, and an edge between two vertices if and only if those variables appear together in some equation. If the graph is disconnected, obviously the system can be split into two distinct systems of smaller size, to be solved separately. However, even if the graph is connected, if its "vertex connectivity" (defined below) is low, we show that eliminating a few variables (for example by guessing their values if over a small finite field) can still split the system.

This suggests a divide-and-conquer approach to solving systems of equations. When the polynomial terms in the equations are very sparse, the system can be reduced to a small set of systems, whose solutions can be computed individually.

Unfortunately, in order to be a useful split, the two subsystems must be of approximately equal size. Otherwise, we will demonstrate that the split is non-productive (See Section 4.1). However, finding a balanced vertex cut is NP-Complete, as we will discuss in Appendix A.

Obviously, if a graph has a minor isomorphic to K_h, the complete graph of h vertices, then there is no cut of size less than h. But for a definition of "balanced" to be given later, there is a balanced cut of size less than $\sqrt{h^3|V|}$, as proven by Alon, Seymour and Thomas [8].

This yields a security criterion: If $\sqrt{h^3|V|}$ is small, then the cipher can be attacked by the methods in this paper. If $h > 128$ (roughly speaking) then the system would be immune to attack by the methods in this paper. See Section 6.2.1 and Section 6.2.

The rest of Section 1 introduces the background in graph theory and polynomial systems that we need. Section 2 details how a system of multivariate linear or polynomial equations can be split using graph partitioning methods. Section 3 describes how the actual vertex cuts can be found; we present two methods, one for graphs with very low vertex connection number, and one based on edge-cuts, which are converted to vertex cuts via the algorithm given in Appendix B. A third method, simulated annealing, is given in Appendix D but was found to be suboptimal and slow. Finding these cuts is an NP-hard problem in general as proven in Appendix A, but we demonstrate how to use them once found in Section 4. Experiments were performed to see how often these cuts can be found and with

what parameters on graphs of various sizes, and the results are given in Section 5 and Appendix E. We demonstrate the impact of the standard "degree dropping" method of relabeling monomials on the variable-sharing graph in Appendix C.

We offer two principal cryptographic applications in Section 6. The first is a method whereby a manufacturer of a sparse implementation of QUAD [18], a provably secure stream cipher, could "poison" the polynomial system at the heart of the cipher and thereby enable messages transmitted with it to be read by the manufacturer. Second, we present a security criterion that a system of equations (and thus a cipher) can have to render itself immune to the attacks of this paper, and we demonstrate that the authentication system HB [30] achieves this criterion with overwhelming probability in any particular instance. Finally, in Section 7 we also give two applications to other branches of mathematics: Euclidean Geometry and Game Theory. We offer our conclusions in Section 8.

Note: The method in this paper is not to be confused with the new method for solving polynomial systems of equations via graph theory proposed by Hårvard Raddum and Igor Semaev in 2006, called the "agreeing-gluing" method (see [46]). However, our method might be a good pre-processor for their method.

1.1. Graph Theory Preliminaries

We present a brief introduction of graph partitions. Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. The following terminology is mostly standard but different papers and books define them somewhat differently. We will use the following conventions:

A vertex $v_i \in V$ is said to be connected to another vertex if there is a path from one to the other using only edges in the edge set E. A pair of subgraphs is said to be connected if there is a path from any vertex of one subgraph to any vertex of the other. A graph is said to be disconnected if there exist vertices v_i and v_j such that v_i is not connected to v_j.

Two subgraphs H_1, H_2 of G are considered vertex-disjoint if they share no vertices, and edge-disjoint is defined likewise. An edge cut of G is a subset of the edges E, so that their removal results in vertex-disjoint subgraphs which are not connected to each other. A vertex cut of G is a subset of the vertices V, so that their removal, as well as the removal of all edges incident on them, results in vertex-disjoint subgraphs which are not connected to each other.

Graph Connectivity. The vertex-connectivity $\kappa(G)$ of a graph G is the size of the smallest vertex cut. If the graph is not connected then $\kappa(G)$ is zero, as the null-set is a vertex cut. The edge-connectivity $\lambda(G)$ of G is the size of the smallest edge cut. If the graph is not connected then it is zero as the null set is an edge cut.

The graph K_n is the complete graph of n vertices, which means it has n vertices and all possible edges. We will prove in Section 1.2 that the vertex-connectivity of

K_n is $n-1$. A graph is planar if it can be drawn on a plane such that no edges intersect except at vertices.

Let B be a graph with six vertices, namely a_1, a_2, a_3 and b_1, b_2, b_3. Each a_i has an edge to each b_j but there are no other edges. Any graph containing either K_5 or B as a minor is not planar [36, 52]. The converse is also true, namely any planar graph does not have K_5 or B as a minor [36, 52].

The graph consisting of one vertex and no edges is usually considered disconnected to make certain theorems have fewer hypotheses.

Menger's Theorem [41] states that if there are n vertex-distinct paths from any particular vertex to any other, then the vertex-connectivity is at least n. Furthermore, if there are n edge-distinct paths then the edge-connectivity is at least n.

Graph Cuts. The vertex- and edge-connectivity are closely related to the effort required to separate a graph into two or more components. The process of removing vertices and edges to make a graph disconnected is called vertex cutting and edge cutting, or vertex partitioning and edge partitioning, respectively.

Edge partitioning are widely used in scientific and engineering applications such as electric circuit design [49], parallel matrix computations [35], and mesh partitioning for finite element analysis [42]. Software implementations including [13, 14, 25, 28, 44, 45, 53] are available for computing edge cuts using a variety of algorithms.

On the other hand, vertex partitioning has fewer applications and we are not aware of any publicly available software implementation. However, vertex cuts can be derived from edge cuts through a simple algorithm. As we shall see, partitioning multivariate equations using graph theoretic methods requires vertex cuts. Therefore, we first compute edge cuts using available software and convert the results into vertex cuts. More details will be given in Section 3.2.

It can be seen that an obvious vertex cut of a graph G is to arbitrarily choose a vertex v_i and remove all neighbours. This results in a trivial cut, and is mostly useless for pratical applications. A process of computing a vertex or edge cut such that each subgraph is roughly the same in cardinality is called "balanced graph partitioning," and is known to be an NP-hard problem (See Appendix A). Heuristics can also be used to compute balanced graph partitions efficiently, which are used in several software packages mentioned above.

The measure of balance is given by α. It is the ratio of the vertex count of the largest disconnected component to the sum of the vertex counts of all of the disconnected components after the cut. Thus if we partition V into the mutually exclusive and collectively exhaustive sets V_1, V_2, C, with no edges between V_1 and V_2, then

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

The effects of vertex partition balance on α is shown in Figure 1.

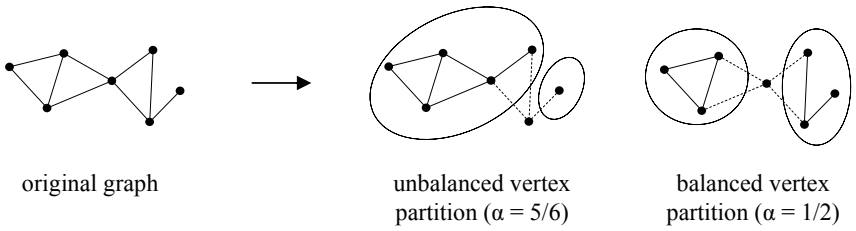

FIGURE 1. Balanced and Unbalanced Vertex Partitions

Obtaining disjoint graphs by balanced graph partitioning is not to be confused with graph k-partitioning, where the set of vertices V of a graph are decomposed into k disjoint sets $V_1, V_2, \ldots, V_k$ such that no two vertices in a set V_i share an edge with each other. Those are a generalization of "Maximum-Independent Set", another NP-Complete problem [19, Ch. 34]. In this paper, we do not consider this form of partitioning.

Throughout this paper, by graph partition or cut we mean balanced graph partition or cut, since the unbalanced case is not useful, for reasons to be explained later in this paper, in Section 4.1.

Finally, we will make use of the common algorithm "depth first search" (DFS), or equivalently its analog "breadth first search." This algorithm can be found in [19, Ch. 22] but for our purposes suffice it to say that it enumerates all vertices reachable from a particular vertex v, and does so in time upper-bounded by the number of edges in the graph. This is sometimes written as $O(|V|d)$ time, where d is the average degree of a vertex in the graph. Let $\delta(v)$ be the degree of a vertex v, then the average degree d of a graph $G(V, E)$ can be expressed as

$$d = \frac{1}{|V|} \sum_{v \in V} \delta(v)$$

1.2. The Vertex Separator Theorems

The vertex separator theorems guarantee the existence of vertex cuts in certain graphs. These theorems are usually stated in terms of graphs with weighted vertices, but setting the weight of each vertex to 1 allows the theorems to be used for unweighted graphs.

Let $G = (V, E)$ be a graph, and V_1, C, V_2 mutually exclusive and collectively exhaustive sets of vertices such that there are no edges between V_1 and V_2. Clearly, the deletion of the vertices in C would disconnect the graph, and so C is a vertex cut.

First, for planar graphs, Lipton and Tarjan [40] proved that a vertex cut of size less than $\sqrt{8|V|}$ exists, such that the size of $V_1 \cup C \leq (2/3)|V|$ and $V_2 \cup C$ likewise. A consequence of this is $(1/3)|V| \leq |V_1| \leq (2/3)|V|$ and V_2 likewise. This implies there is a cut with $|C| \leq (1/3)|V|$ and $1/2 \leq \alpha \leq 2/3$.

Second, for graphs of genus $g > 1$, Gilbert Hutchinson and Tarjan [27] proved that there is a cut with $|C| \in O(\sqrt{g|V|})$. A planar graph has genus $g = 0$.

Contracting an edge between v_i and v_j signifies creating a new vertex, such that any edge to either v_i or to v_j now goes to this new vertex instead, and then both v_i and v_j are deleted. A graph G is said to have a minor K if some subgraph of G is isomorphic to K after contracting zero or more edges. For example, the Kuratowski-Wagner theorem states that a graph is planar (genus zero) if and only if it has no K_5 or B minors [36, 52]. It is notable that K_h has, asymptotically, genus $g \in \Omega(h^2)$.

Third, in 1990 Alon, Seymour and Thomas [8] proved that if the graph has no K_h minor, then regardless of genus, there is a cut with $|C| \leq \sqrt{h^3|V|}$, and as before, $V_1 \cup C \leq (2/3)|V|$ and $V_2 \cup C$ likewise. This means that $(1/3)|V| \leq |V_1| \leq (2/3)|V|$ and V_2 likewise. Also this implies there is a cut with $|C| \leq (1/3)|V|$, and $1/2 \leq \alpha \leq 2/3$. Furthermore, they conjecture that the h^3 can become h^2 instead, and the conjecture remains open at the time of this writing, so far as the authors are aware.

For our problem, the $(1/3)|V| \leq |V_1| \leq (2/3)|V|$ condition would represent excellent balance, as we will show that highly unbalanced cuts, e.g. with $V_1 \approx |V|$, would be useless. However, $|C| \leq (1/3)|V|$ is not strict enough, as one method for solving a polynomial system of n variables over GF(2) would involve solving $2^{|C|+1}$ systems of $(n - |C|)/2$ variables. While $|C| \leq \sqrt{h^3|V|}$ is important, determining h might be infeasible. In fact, determining h is NP-hard, because an oracle that could detect if $h \geq h_0$ could decide if a graph contains a clique of size h_0, which is a known NP-Complete problem "Max-Clique" [19, Ch. 34].

An interesting case is to consider which graphs have the worse possible (highest) vertex-connectivities. The graph K_n, defined earlier, has vertex-connectivity $n-1$, which is optimal. In fact, it is the only graph with that vertex-connectivity and number of vertices.

Suppose a graph G with vertex-connectivity $n-1$ and n vertices differs from K_n. Since K_n has all possible edges by definition, this is only possible if G lacks an edge or edges that K_n has. Call one such edge (v_i, v_j). Then remove all $n-2$ vertices except v_i and v_j. Since there is no edge (v_i, v_j), and since there is no longer path from v_i to v_j, (as no vertices other than v_i and v_j remain in the graph), then the graph is disconnected. Therefore, the vertex-connectivity of G is at most $n-2$, which is a contradiction.

Obviously, no graph with n vertices can have vertex-connectivity greater than $n-1$ since removing any $n-1$ vertices will result in a graph with one vertex and therefore that graph must have no edges, and so it is disconnected.

1.3. Algebraic Cryptanalysis

In algebraic cryptanalysis, a cipher is first described as a system of equations, where its variables correspond to the key or plaintext, or other features of the encryption such as an initialization vector. Solving the system is equivalent to breaking the cipher. We discuss below the process of algebraic cryptanalysis of bit-based stream ciphers.

A bit-based stream cipher consists a pseudo-random number generator, which outputs a series of keystream bits k_i. Given plaintext bits p_i, the stream cipher encrypts in GF(2) as $c_i = p_i + k_i$, where c_i are bits of the ciphertext. The keystream bits k_i are usually generated by some function which could be written $k_{i+1} = f(s_i)$, where s_i is some internal state, often many bits long, and $s_{i+1} = g(s_i)$. Because stream ciphers are traditionally designed to be implemented in digital hardware composed of logic gates, converting f and g into a system of equations over GF(2) is relatively straight-forward.

Then, if both p_i and c_i are known for a few bits $i_1, i_2, i_3, \ldots$ one can attempt to write a system of equations based on $p_i + c_i = k_i = f(s_{i-1})$. This can be because of a known-plaintext or chosen-plaintext attack scenario, or because of just knowing the format of the plaintext. For example, in ASCII, the most-significant bit of each byte is zero, and the next is almost always 1.

Note further, it often occurs that as many equations as desired can be made available. For example, by using longer messages in a "chosen-plaintext attack." Systems with more equations than unknowns are called over-defined. The extra information, resulting from having more equations than unknowns, can have a significant effect on the efficiency of solving the system. For example, a quadratic system with n^2 equations in n unknowns can be solved via Linearization in polynomial time (in fact, n^6 time).

Alternatively, stream ciphers can output $2, 4, 8$ or any number r bits at a time, but this is equivalent either to outputting a field element of GF(2^r), or a vector from GF$(2)^r$, and modeling the system as above. for example, the stream cipher QUAD [18], which we address in Section 6.1, outputs 160 bits at a time, which we treat as a single 160-bit vector.

Algebraic attacks on stream ciphers have had successes on some basic stream ciphers, see for example [20, 21]. Since then, equation generation techniques for more complicated stream ciphers with different components were found [6, 17, 54]. However, finding efficient methods to solve the equation system obtained from stream ciphers is still an active area of research. In this paper, we investigate the

feasibility of using graph theory to partition systems of equations into smaller ones in order to reduce computation time for a solution. Details of this will be discussed in Section 2.2.

For these reasons, and also in the cryptanalysis of block ciphers, we might have a system of polynomial equations over GF(2^r). For cryptographic applications, note that we only require solution of the polynomial equations in GF(2^r), but not in its algebraic closure. Furthermore, one solution is usually enough to break a system. We assume that all systems of equations generated for cryptanalysis contain at least one solution, usually because some message was sent and so therefore there is at least one key which maps that plaintext to that ciphertext.

Usual choices for fields include GF(2^{32}), GF(256), GF(16), but most importantly GF(2), which is used for bit-based stream ciphers. For the moment, we will focus on GF(2), and address the extension fields afterward. Finally, systems over GF(2^r) can always be written as systems with r times as many variables and equations over GF(2), see for example [11], but the technique was well-known before. Therefore, it suffices to consider only GF(2).

If the system of equations is overdefined enough, it might be possible to eliminate the variables in the vertex cut set from the system by discarding those equations where these "forbidden" variables appear. Furthermore, we can perform an edge cut instead, and discard the equations corresponding to the edge cut set. Alternatively, the search for a "good" cut can be performed many times on random but fixed-sized subsets of the equations. We do not address such highly over-defined systems of equations here.

The cryptanalysis of each cipher is different, using unique properties of that cipher, and so it is not possible to give a universal recipe for attacking all ciphers. In Section 4.2, a general method of constructing a graph from a polynomial system of equation will be discussed. This method applies for all equation systems generated from bit-based stream ciphers, or on extension fields as explained above.

2. Partitioning Systems of Equations

By considering how linear systems are often partitioned for applications such as parallel computation, we can understand a great deal about how to partition polynomial systems.

2.1. Linear Equations

The need for efficient solutions of large sparse linear systems of equations is present in many engineering and modelling applications. In parallel environments, graph partitioning can be used to separate a system into pieces so that computation can be carried out in the most efficient manner. The following example gives a basic

idea of how graph partitioning can be used to separate a linear system. Suppose we are to solve the following linear system of equations using two processors.

$$\begin{bmatrix} 1 & 0 & -2 & 0 & 2 \\ 0 & -3 & 0 & 1 & 1 \\ 5 & 0 & -4 & 0 & 0 \\ 0 & 1 & 0 & 3 & 0 \\ 0 & 0 & -1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ -7 \\ 14 \\ 25 \end{bmatrix} \tag{1}$$

Create a graph $G(V, E)$ with vertices $v_i \in V$ representing each of the variables x_i, and edges $v_i, v_j \in E$ whenever x_i and x_j appear together in an equation. An edge partition can then be applied on G to separate it into two balanced parts V_1, V_2 with edge cut set $\{(v_2, v_5), (v_4, v_5)\}$. This process is shown in Figure 2.

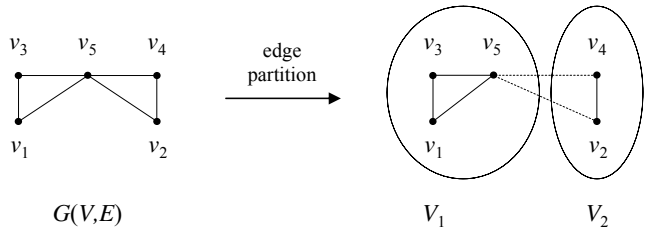

FIGURE 2. Graph corresponding to linear system (1) and an edge partition

The variables corresponding to vertices in V_1, V_2 can then be assigned to separate processors, and the only communication necessary between the two processors are among the variables corresponding to vertices with edges across V_1, V_2.

2.2. Polynomial Equations

Polynomial boolean equations naturally arise in algebraic attacks on symmetric ciphers, for example in inversion substitution boxes. Any map from any finite set to any finite set can be written as a polynomial system of equations over any finite field. This is called The Universal Mapping Theorem, and explains why finite field polynomials are a sort of "skeleton key" of combinatorics [12, Ch. 4].

Computing the solution to a system of quadratic equations is known to be NP-Complete [10, Ch. 3.9]. A variety of solution techniques exist, such as Linearisation, Gröbner, Resultants and Homotopy Methods [12, Ch. 5], as well as very new methods such as SAT-solvers [9] or the Raddum-Semaev Method [46], but if any ever become polynomial time, then $P = NP$. Thus all currently known methods run in time super-polynomial to the number of variables.

All of these techniques consider the equation system as a whole. In this paper, we develop a method of separating the equation system into smaller ones, so that the solutions can be computed more efficiently, by applying the known methods to the two smaller systems.

2.3. Underlying Complexity of Polynomial System Solving

Let $O(2^{\phi n})$ be the complexity of solving a polynomial system, over GF(2). Since we are seeking solutions over GF(2), we can iterate over all possible solutions in $\mathrm{GF}(2)^n$, which costs $O(2^n)$. This is the upper bound to the complexity, which means $\phi \leq 1$.

Of course, lower bounding $k \leq \phi$ for some constant $0 < k \leq 1$ would prove $P \neq NP$, and so is unlikely to be possible with currently known techniques.

In his PhD Thesis, Daniel Rolf proved that an n variable logical Satisfiability problem can be solved in time $2^{\phi n}$ where $\phi \leq 0.4029$, see [48]. This is not the same as solving an n variable polynomial system of equations, but the two problems are polynomially reducible to each other (because both are NP-hard and have decision problems in NP), and so the bound is probably comparable. In [9] it was shown that converting a quadratic system in n variables and m equations to a logical satisfiability problem would result in $mn^2\beta/12$ variables, but this conversion might not be the most efficient. Note β is the fraction of possible coefficients that are non-zero.

It is well known that systems of equations over $\mathrm{GF}(2^r)$ can be represented as systems over GF(2) with r times as many variables and r times as many equations. For an example, see [11]. Thus $2^{r\phi n}$ would be a lower-bound on the complexity of solving a system of equations over $\mathrm{GF}(2^r)$, and note the brute-force complexity would be 2^{nr}.

3. Finding the Cuts

3.1. Special Cases for Low Vertex-Connectivity

The special case of a vertex-connectivity of zero for G (i.e. $\kappa = 0$ or a disconnected graph) can be detected with a standard DFS. Just mark each vertex "unvisited" initially, and mark it "visited" as the DFS progresses. If not all nodes are visited by the end of the DFS, then G, being an undirected graph, is not connected. This has expected running time of $\Theta(|V|d)$ if the average degree of the graph is d.

The special case of $\kappa = 1$ can be detected by removing and restoring, one at a time, all vertices, and checking a DFS in each case. Likewise, the number two can be detected by removing all possible pairs. This will require, to detect a vertex-connectivity of at most k,

$$\sum_{i=0}^{k} \binom{|V|}{i} |V|d \approx \frac{(|V| + 1 - k/2)^{k+1}\, d}{k!}$$

based on the approximations

$$\binom{a}{b} + \binom{a}{b-1} = \binom{a+1}{b} \text{ and } \binom{a}{n} \approx \frac{(a-n/2)^n}{n!}$$

This approach also has the advantage of revealing the set of vertices that we must remove in order to cut the graph into disconnected components.

Since we desire a balanced cut, we can mark each subset of k or fewer vertices that cuts the graph, found during the search, with a difficulty number. The difficulty number should be the size of the largest connected component, since this will be the hardest system of equations to solve (presumably). At the conclusion of the search, we take the cut with the lowest difficulty number.

Thus, if $|V|$ is small or if the vertex-connectivity κ is thought to be small, one should test for $\kappa = 0$, followed by $\kappa = 1$, and $\kappa = 2$ until some computational time has elapsed, and then take the cut with the minimum difficulty number.

3.1.1. Justification of the Difficulty Number. For example, if a system has 100 variables, and a removal of 4 vertices slices it into three components of size 89, 4, and 3, then the running time to solve the system would be

$$2^4[2^{\phi 89} + 2^{\phi 4} + 2^{\phi 3}]$$

Note ϕ was defined in Section 2.3. On the other hand, if a removal of 5 vertices slices it into two components of size 47 and 48, then the running time to solve the system would be

$$2^5[2^{\phi 47} + 2^{\phi 48}]$$

Since $2^{\phi 3}$ and $2^{\phi 4}$ are trivial, the second one is faster if and only if $2^{\phi 89}/2 > 2^{\phi 47} + 2^{\phi 48}$, or more plainly, $2^{\phi 42 - 1} > 1 + 2^{\phi}$ which is almost certainly the case.

The purpose of this example is to show that neither the number of connected components nor the size of the cut are the crucial factor. The crucial factor is the size of the largest component after the cut.

3.1.2. Complexity of Pseudo-Brute Force Search. Therefore, if the vertex-connection number κ is small, an optimal cut can be found by trying the removal of all subsets of the vertices of size 1, followed by size 2, then size 3, and so on. Even for a problem of 100 variables, note that

$$\binom{100}{7} + \binom{100}{6} + \cdots + \binom{100}{2} + \binom{100}{1} + \binom{100}{0} = 17,278,988,696$$

and performing 17 billion DFSs might indeed be feasible.

By tagging each subset that disconnects the graph with a difficulty number, equal to the size of the largest connected component after the cut, we would be assured to find the optimal cut of all those possible, of size up to $|C| = 7$.

3.2. Multilevel Partitioning

While the problem of balanced graph partitioning is NP-hard, heuristic algorithms are generally very efficient.

A popular and efficient scheme for balanced graph partitioning is called multilevel partitioning, which will be used for our experiments. Suppose a graph G_0 is to be partitioned. G_0 is first coarsened progressively into simpler graphs $G_1, G_2, \ldots, G_r$ by collapsing adjacent vertices. The process of choosing vertices to collapse is called matching. After reaching a graph G_r with the desired level of simplicity, a partition is performed. The result is then progressively uncoarsened back through the chain of graphs G_{r-1}, G_{r-2}, G_0. At each uncoarsening step a refinement to the partition can be performed. The output is then a partition of G_0. More details multilevel partitioning can be found in [32, 29]. Examples of partitioning and refinement algorithms include Kerighan-Lin [33] and its variants such as Fiduccia-Mattheyses [24]. The multilevel partitioning scheme can be made to work with graphs with weighted vertices and edges, but for this paper we only consider the unweighted version.

As discussed in Section 1, it appears that implementations of vertex partitioning with constraints are not readily available. On the other hand, a few publicly available edge-partitioning software packages that allow constraints exist, which are mainly used for partitioning nodes for matrix reordering in finite element method problems, or telecommunications network reliability analysis. We have decided to use the edge partitioning program Metis [3] for our study. It implements the multilevel partitioning algorithms, which are currently among the most efficient partitioning algorithms. We then convert the edge partition to a vertex partition, but the vertex partition obtained will be suboptimal. The conversion is discussed in Section B.

The Matlab interface Meshpart for Metis is used. We have used the balanced edge cut implementation on unweighted undirected graphs. Random graphs of varying sizes and edge density were used, where the edge density ρ of a graph G with $|V|$ vertices is the ratio of the number of edges $|E|$ in G with to the maximum number of possible edges, i.e.

$$\rho = \frac{|E|}{\binom{|V|}{2}} = \frac{d}{|V| - 1}$$

where d is the average degree.

4. Exploiting the Cut

Assuming that a variable-sharing graph has a suitable vertex cut, we will now discuss how to exploit that fact to solve the system of equations faster. This turns

out to be different in very small finite fields from all other cases. However, the techniques for large fields will work for any field.

4.1. Exploiting Vertex Cuts in GF(2)

We now restrict our attention to quadratic systems over the field GF(2), and later will address its finite extensions. Suppose that the time complexity of computing the solution to a system of n quadratic equations is $O(2^{\phi n})$. If the system can be broken down into two separate but equal-sized systems, with a very small cut, then the time complexity becomes $O(2^{\phi n/2})$.

Suppose we have a system of 100 equations and unknowns. This would require $\sim 2^{\phi 100}$ time, which is probably not feasible. But suppose that if 3 particular variables were to become known, that the system would partition into two pieces of size 48 and 49 variables (note $48 + 49 + 3 = 100$). The three variables have $2^3 = 8$ possible settings, which can be exhausted. Then the running time would be $\sim 8(2^{\phi 48} + 2^{\phi 49})$, which may be quite feasible. Even into sizes 45 and 52 variables is acceptable, with time $\sim 8(2^{\phi 45} + 2^{\phi 52})$. But note also that a partition into sizes 95 and 2 variables would be useless since $\sim 8(2^{\phi 95} + 2^{\phi 2})$ is probably infeasible since $2^{\phi 95}$ is probably infeasible. For this reason, only balanced vertex cuts are useful. Here, "probably" refers to the uncertainty of the lower-bound for ϕ.

4.2. Constructing the Graph

The graph will be constructed as follows. Create a vertex for each variable in the system, and draw an edge between two vertices v_a and v_b if and only if there is some equation where both the variable a and b are present, regardless if in the same monomial or not. If the resulting graph is disconnected, then the system can be written as

$$\begin{aligned} f_1(x_1, \ldots, x_s) &= 0 \\ f_2(x_1, \ldots, x_s) &= 0 \\ &\vdots \\ f_q(x_1, \ldots, x_s) &= 0 \\ f_{q+1}(x_{s+1}, \ldots, x_n) &= 0 \\ f_{q+2}(x_{s+1}, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_{s+1}, \ldots, x_n) &= 0 \end{aligned}$$

but if the variable numbering and equation numbering were permuted, then it might be very hard to notice this.

But in all likelihood, this will not occur and the graph will be connected. Now suppose instead that a set of vertices C is a vertex cut, and furthermore that the

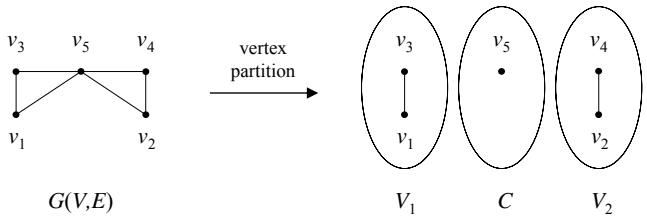

FIGURE 3. Graph of quadratic system (2) and a vertex partition.

two disconnected components after C is removed from V are roughly equal sized. Then one could simply guess all $2^{|C|}$ combinations of values for the variables in C. Then we have

$$2^{|C|}\left(2^{\phi\frac{(|V|-|C|)}{2}}+2^{\phi\frac{(|V|-|C|)}{2}}\right)=2^{(\phi/2)|V|+(1-\phi/2)|C|+1}$$

by previous work.

This is an improvement if and only if:

$$(\phi/2)|V|+(1-\phi/2)|C|+1<\phi|V|$$

or more simply

$$|C|<\frac{\phi|V|-2}{(2-\phi)}\approx\frac{|V|}{2/\phi-1}$$

which using Rolf's bound from [48] would be $|C|<0.32216|V|$.

Thus, to find the optimum way such that the system of equations splits into two independent subsystems with similar number of variables is equivalent to finding a vertex cut set S such that G splits into a disconnected graph with components H_1, H_2 of similar vertex count.

For example, suppose we have the following quadratic system of equations over GF(2).

$$\begin{aligned}x_1x_3+x_1+x_5&=1\\x_2x_4+x_4x_5&=0\\x_1x_5+x_3&=0\\x_2x_5+x_2+x_4&=0\\x_2+x_4x_5&=1\end{aligned}\tag{2}$$

The corresponding graph of the system and a vertex partition is shown in Figure 3.

The quadratic system can then be reduced into two systems of two equations with the common variable x_5, which becomes (3).

$$\begin{aligned} x_1x_3 + x_1 + x_5 &= 0 & x_2x_4 + x_4x_5 &= 0 \\ x_1x_5 + x_3 &= 0 & x_2x_5 + x_2 + x_4 &= 0 \\ & & x_2 + x_4x_5 &= 1 \end{aligned} \tag{3}$$

Applying guesses to x_5 in (2) and computing solutions to the reduced systems gives

$$\begin{aligned} x_5 &= 0 : \text{no solution} \\ x_5 &= 1 : x = (1, 0, 1, 0, 1) \end{aligned}$$

which is the same as if we have computed the solution to the full system of equations.

4.3. Exploiting in Large Finite Fields or Infinite Fields

If we had chosen GF(256) (used in the Advanced Encryption Standard [5]) or even GF(16) [47], then the number of possibilities for $|C|$ removed variables would be $256^{|C|}$ or $16^{|C|}$, which would be extremely infeasible by guess-and-check. In those cases, and for the case of an infinite field like $\mathbb{Q}$, we suggest the following method.

Suppose $|C| \subset |V|$ will partition V into C, H_1, H_2, with no edges between H_1 and H_2. This means that there are no equations which have a variable from the vertices of H_1 and also a variable from the vertices of H_2. Now relabel the variables as follows. First, denote the variables represented by vertices in C by xs, likewise those form H_1 as ys and those in H_2 as zs. Since no equation has both ys and zs then call those that have ys and possibly some xs as fs, and those with zs and possibly some xs as gs.

Then one has

$$\begin{aligned} f_1(x_1, \ldots, x_{|C|}, y_1, \ldots, y_{|H_1|}) &= 0 \\ f_2(x_1, \ldots, x_{|C|}, y_1, \ldots, y_{|H_1|}) &= 0 \\ &\vdots \\ f_\ell(x_1, \ldots, x_{|C|}, y_1, \ldots, y_{|H_1|}) &= 0 \\ g_1(x_1, \ldots, x_{|C|}, z_1, \ldots, z_{|H_2|}) &= 0 \\ g_2(x_1, \ldots, x_{|C|}, z_1, \ldots, z_{|H_2|}) &= 0 \\ &\vdots \\ g_{m-\ell}(x_1, \ldots, x_{|C|}, z_1, \ldots, z_{|H_2|}) &= 0 \end{aligned}$$

Then we can use a well-known technique of resultants. In the resultant technique, unknowns are divided into two classes: "variables" and "parameters".

Given m polynomials, one can label $m-1$ of the unknowns as "variables", and the remaining unknowns as "parameters". The resultant of these m polynomials will be a polynomial entirely in the "parameters", but with none of the "variables". This is a highly-non-trivial process, but for $m < 12$ it is quite feasible. This can be repeated for all subsets of size m among the equations.

We have two cases, the fs and the gs. The xs will be the "variables" in both cases, and the ys or zs will be the "parameters" in each case. We will then take $|C|$ equations from the fs, and calculate their resultant. This will be a polynomial entirely in terms of ys. This can be applied to all subsets of the fs consisting of exactly $|C|$ equations, or fewer as desired. This will also be done for all subsets of size $|C|$ among the gs, to produce polynomials entirely in terms of zs.

Observe now that the fs and gs are entirely disjoint, and can be solved separately. Their solutions can be plugged back into the original polynomials to obtain the xs when finished.

5. Partitioning Experiments

Vertex partitioning experiments have been performed on random graphs of different sizes. These were performed on a Pentium M 1.4 Ghz laptop with 1 gigabyte of RAM using the meshpart Matlab interface to the Metis partitioning software. The experimental results are shown in Table 1, Table 3, and Table 4. Note more data is given in Appendix E. In the tables $|V|, |E|$ are the number of vertices and edges in G respectively, ρ is the edge density, d is the average degree of vertices, $|C|$ is the size of the vertex separator, $|V_1|, |V_2|$ are the numbers of vertices in the separated disjoint subgraphs of G, and α is as defined before. The times taken for the partitioning are also shown.

It can be observed from Table 1 that the growth of α 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 α is too near to the upper limit of 1.0 for the method to be useful. Since the number of total edges for a graph of size n is $O(n^2)$, the edge density must be smaller with a larger graph. Note, the time taken for all vertex partitions in the experiments would be extremely negligible to the time taken to solve the partitioned systems.

Figure 4 shows the effect of varying number of edges on the balance of vertex partitions for graphs of different order $n = |V|$. The partition balance parameters α shown in the figure are obtained from the mean of 200 algorithm runs.

$\|V\|$	$\|E\|$	ρ	d	$\|C\|$	$\|V_1\|$	$\|V_2\|$	α	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

TABLE 1. Vertex Partitioning Experiments

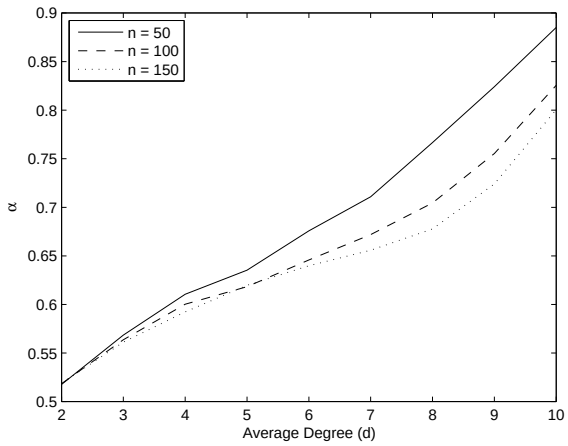

FIGURE 4. Partition Balance vs Average Degree of Graph

It can be observed that the partition balance parameters α increases approximately linearly with the average degree d of the graphs. Furthermore, the increase seems to be slower with graphs of larger cardinality. However Table 1 suggests that the increase becomes faster again for even larger graphs. From the vertex separator theorems, we assert that a reasonable balance is obtained when $1/2 \leq \alpha \leq 2/3$.

With the graph sizes considered in Figure 4, this reasonable balance can be obtained with $d \leq 6$. This also can mean that when $d \leq 6$, the graph can likely be made planar, since α satisfies the vertex separator theorem for planar graphs.

6. Applications to Cryptology

The technique of computing vertex separators to split quadratic systems of equations is applied to the stream cipher QUAD [18] and the authentication system HB [30].

6.1. The Stream-Cipher QUAD

An example of a cipher for which this vertex separator method would be useful is the QUAD family of stream ciphers [18]. 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[1] GF(2). This system of equations is not secret, but publicly known, and there are criteria (such as relating to rank) which we omit here. In a different context, QUAD has been analyzed in [15].

6.1.1. Background. The authors of QUAD recommend $k = 2$, and $n \geq 160$, so let us assume 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 $\mathrm{GF}(2)^n \to \mathrm{GF}(2)$, and 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 $\vec{s}_i$ of 160 bits. The first 160 equations are evaluated at this vector, and the resulting $\vec{f}_1(\vec{s}_i) = \vec{s}_{i+1}$ bits becomes the new state. The second 160 equations are evaluated to become the output of that stage $\vec{z}_i = f_2(\vec{s}_i)$. The $\vec{z}_i$ is added to the next n bits of the plaintext $\vec{p}_i$ using GF(2) addition, and is transmitted as a ciphertext $\vec{c}_i = \vec{p}_i + \vec{z}_i$.

There is an elaborate setup stage which maps the secret key and an initialization vector to the starting state $\vec{s}_1$.

Finding a pre-image under this map, that is to say, finding $\vec{s}_i$ given $\vec{s}_{i+1}$ and $\vec{z}_i$, is equivalent to solving a quadratic system of $2n$ equations in n unknowns, and that is NP-hard [10, Ch. 3.9]. This is further complicated by the fact that the adversary would not have $\vec{s}_{i+1}$, but rather only $\vec{z}_i + \vec{p}_i$.

[1] Mention is made of other fields in the paper [18] but in practice there is no question that GF(2) is intended.

Given a known-plaintext scenario, where the attacker knows both the plaintext $\vec{p}_1, \ldots, \vec{p}_n$ and ciphertext $\vec{c}_1, \ldots, \vec{c}_n$, allows one to write the following system of equations:

$$\begin{aligned}
\vec{c}_1 + \vec{p}_1 &= \vec{z}_1 = g(\vec{s}_1) \\
\vec{c}_2 + \vec{p}_2 &= \vec{z}_2 = g(\vec{s}_2) = g(f(\vec{s}_1)) \\
\vec{c}_3 + \vec{p}_3 &= \vec{z}_3 = g(\vec{s}_3) = g(f(f(\vec{s}_1))) \\
\vec{c}_4 + \vec{p}_4 &= \vec{z}_4 = g(\vec{s}_4) = g(f(f(f(\vec{s}_1)))) \\
&\vdots = \vdots \\
\vec{c}_i + \vec{p}_i &= \vec{z}_i = g(\vec{s}_i) = g(\underbrace{f(f(f(\cdots f(}_{i \text{ times}} \vec{s}_1) \cdots))))
\end{aligned}$$

What is interesting here is that $g(f(f(f(f(x)))))$ and higher iterates might be quite dense even if f is sparse. The authors of QUAD have excellent security arguments if the polynomial system is generated by fair coins, but then 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 the conference talk but not the paper, they mention that a slightly sparse f might still be secure furthermore because of repeated iteration and just the general difficulty of the MQ problem, there would probably be no feasible algebraic attack against the sparse version.

6.1.2. Poisoned Equations and QUAD. One could imagine the following scenario, which is inspired by Jacques Patarin's system "Oil and Vinegar" [34]. 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 cut with $\alpha \approx 0.6$. Our experiments show that this is a feasible cut and feasible α. 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 cut of 20 vertices divides the variable sharing graph into 56 and 84 vertices. This means that an attacker would need to only know the plaintext and ciphertext of one 160-bit sequence, and solve the equation

$$g(\underbrace{f(f(\cdots f(f(}_{i-1 \text{ times}} s_1)) \cdots))) = p_i + c_i$$

For any guess of the key, this would be solving 56 equations in 56 unknowns and 84 equations in 84 unknowns, trivial for a SAT-solver [9] [10, Ch. 3] [12, Ch. 7]. Only 2^{20} iterations would be required, and with a massive parallel network, such as BOINC [1], this would be quite feasible.

6.2. Immunity to these Attacks

Since the feasibility of computing a small vertex separator depends on the density of edges on the graph, for a cipher to be immune to attacks of this kind, the variable-sharing graph should contain a fully connected graph K_n. See Section 6.2.1 for an explanation.

Furthermore, the authentication code HB exhibits this property with overwhelming probability.

6.2.1. Security of the Authentication System HB. Using the vertex separator theorems from Section 1.2, we learn that if a graph has no K_h minor, then a cut of at most $\sqrt{h^3|V|}$ can be found, with α such that the cut is useable. This condition may be hard to detect, but the presence of such a cut means that the system of equations can be solved more easily than expected, using the techniques of this paper. However, if a graph does have a K_h minor, then no cut can be smaller than $h-1$ vertices.

Thus, deliberately embedding a K_{200} minor, such as K_{200} itself, would render a system of equations unsolvable by this method. This would require 200 variables in a set S such that each variable in S appears in the same equation as each other variable in S at least once in the system. We will show below that the authentication scheme HB achieves this requirement.

6.2.2. Background. One scheme could be as follows. The prover is in possession of a secret vector $\vec{s}$, of k bits length. The verifier shall challenge the prover with challenge vectors, also of $k = 128$ bits, and the prover shall respond with the dot product of the secret vector with each challenge vector.

An adversary who does not know the secret vector would be correct with probability 1/2 at each stage, but the true prover would always be correct. Thus, after 32 challenges, the adversary is admitted only with probability 2^{-32}. However, after 128 such challenges (4 runs of the protocol in this case), a linear algebra problem could be setup and the secret vector recovered. (In reality, a few more challenges would be required to ensure the system was full-rank).

The following scheme is an improvement of the above, presented by Blum and Hopper [30], thus the name HB. Let the prover tell the truth with probability 7/8, and lie with probability 1/8. The adversary who does not know the secret vector would still be wrong with probability 1/2, and yet the prover would be wrong with probability 1/8. Repeating the scheme 32 times, for example, would result in an expected number of errors of 4 for the prover, and 16 for the adversary. Thus setting a limit of 7 errors would allow the prover to frequently gain admission (96.05%) and the adversary rarely so (0.11%). More iterations would make these boundaries sharper, and the probability of 1/8 can actually be anything in the interval $(0, 1/2)$.

Amazingly, this minor change destroys the linear algebra, because the attacker does not know which equations are valid or not. One attack is to iterate on all possible arrangements of 3, 4, or 5, errors (the most likely values), but there are 242,296 of these. Alternatively, 0, ..., 7 errors brings the total to 4,514,873, certainly feasible. As the number of challenges increases, however, the number of possible arrangements of truth and lies grows super-polynomially.

As an algebraic attack, one could generate a variable for each interaction, ℓ_i, which is 1 if the prover lied in the ith interation and 0 if not. Since there are probably 5 or fewer lies, any product of six of the ℓ_i is zero. This generates $\binom{32}{6}$ very short polynomial equations, each consisting of a sixth degree monomial equal to zero. One can see that even omitting a few would introduce spurious solutions.

6.2.3. The Variable-Sharing Graph of HB. For each run of the protocol, there would be $\binom{32}{6}$ equations with the ℓ_i, but then 32 equations involving the dot-products, which would include the 128 unknowns that are the secret vector $s_1, \ldots, s_{128}$. If the challenge vector $c_1, \ldots, c_{128}$ had a bit $c_i = 1$, then s_i will be present as a variable in the corresponding equation. Thus, with probability 1/4, s_i and s_j will appear together in any given challenge. Since there are 32 challenges, s_i and s_j will have an edge between them with probability $1 - (3/4)^{32} \approx 1 - 10^{-4}$. Thus the variable-sharing graph will have K_{128} as a subgraph in all likelihood, or if not then K_{127}, K_{126}, or at absolute worse K_{125}. Thus the vertex connectivity is at least 125.

6.3. Remedy to Poisoned Systems for QUAD

While finding a balanced vertex-cut is NP-hard as proven in Appendix A, calculating the vertex-connection number $\kappa(G)$ is easier. If $\kappa(G) > 80$, for example, then there is no cut, balanced or otherwise, with fewer than 80 vertices. Then, by calculating κ, a manufacturer of QUAD could prove that they are not poisoning the quadratic system as explained in the previous subsection.

7. Mathematical Applications

Some interesting mathematical will also be described in this section, namely Nash Equilibria and the Appolonius Problems.

7.1. Nash Equilibria

This is a well known topic in economic game theory [50], [22]. Briefly, a strategy is a Nash equilibrium if for each player p, never could p attain a better payoff by changing only p's own strategy, leaving all other strategies fixed.

Nash equilibria can be characterized by systems of polynomial equations [50]. We investigate here a "cube game," based on a graph G = the edge graph of the 3-dimensional cube. The eight players are associated to the vertices of the cube:

$$\begin{pmatrix} Adam & Bob & Carl & Dick & Fran & Gary & Hugh & Jane \\ 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ a & b & c & d & f & g & h & j \end{pmatrix}$$

Thus Adam's neighbors are Bob, Carl and Fran. The choices made by Dick, Gary, Hugh and Jane are irrelevant for Adam's payoff. $a, b, c, \ldots$ are probabilities. b is the probability allocated by Bob to the first choice, etc. Thus Adam's equation is a trilinear form in the three variables b, c, and f; analagously for the others. Generalizing the specific game discussed by Sturmfels[50], we have these equations:

$$\begin{aligned} (u_1 b - 1)(u_1 c - 1)(u_1 f - 1) &= rr \\ (u_1 a - 1)(u_1 d - 1)(u_1 g - 1) &= rr \\ (u_2 a - 1)(u_2 d - 1)(u_1 h - 1) &= rr \\ (u_2 b - 1)(u_2 c - 1)(u_1 j - 1) &= rr \\ (u_3 a - 1)(u_2 g - 1)(u_2 h - 1) &= rr \\ (u_3 b - 1)(u_2 f - 1)(u_2 j - 1) &= rr \\ (u_3 c - 1)(u_3 f - 1)(u_3 j - 1) &= rr \\ (u_3 d - 1)(u_3 g - 1)(u_3 h - 1) &= rr \end{aligned} \tag{4}$$

Sturmfels used values $u_1 = 3, u_2 = 5, u_3 = 7, rr = 1/10$.

7.1.1. Experimental Findings. Our first experiments were performed on the machine[2] `sage.math.washington.edu` using MAGMA [2], and SINGULAR [4]. The machine has 64 gigabytes of RAM and 16 AMD Opteron cores. We chose the ordering "degrevlex". It is known that MAGMA uses the algorithm F4 [23], and SINGULAR uses the Buchberger algorithm [16].

Even after substituting in the above constants for u_2, u_3, and rr, keeping u_1 as symbolic, no solution was available from MAGMA after 63 minutes, using 585 megabytes of RAM. Neither was a solution forthcoming from SINGULAR after 1054 megabytes and 55 minutes.

[2]We would like to thank Prof. William Stein for access to this machine, and the NSF for purchasing it. The grant was DMS-0555776.

However, one can see that half the equations use the variables b, c, f, j and the other half use a, d, g, h. Thus we can split the system into first

$$\begin{aligned}(u_1b-1)(u_1c-1)(u_1f-1)-1/10,\\ (5b-1)(5c-1)(u_1j-1)-1/10,\\ (7b-1)(5f-1)(5j-1)-1/10,\\ (7c-1)(7f-1)(7j-1)-1/10,\end{aligned}$$

which was solved in 0.1 seconds by MAMGA and 0.15 seconds by SINGULAR, and second

$$\begin{aligned}(u_1a-1)(u_1d-1)(u_1g-1)-1/10,\\ (5a-1)(5d-1)(u_1h-1)-1/10,\\ (7a-1)(5g-1)(5h-1)-1/10,\\ (7d-1)(7g-1)(7h-1)-1/10\end{aligned}$$

which was solved in 0.1 seconds by MAGMA and 0.17 seconds by SINGULAR.

Since u_1 is a constant known parameter, it can be shared between the two broken systems. If the parameters u_2, u_3, and rr are left in the equations, the same splitting applies.

Using the Dixon-EDF resultant algorithm [37], Lewis [38] considered the fully symbolic set of eight equations above (4). After a small simplification using substitutions $a = a/u_1, g = g/u_3$, etc., Lewis computed the resultant for a in about ten minutes on a desktop computer. It has 2961 terms. Each of the fully symbolic split systems of four equations derived from (4) is handled by Dixon-EDF in about five seconds. The result is, of course, the same polynomial of 2961 terms. The systems in which there are substitutions $u_2 = 5, u_3 = 7, rr = 1/10$ are trivial for Dixon-EDF.

We see here, first, the great superiority of Dixon-EDF over Gröbner basis techniques, and secondly, the wonderful efficacy of the splitting idea to resultant computations.

7.2. Appolonius Problems

The Apollonius Circle Problem dates to Greek antiquity. Given three circles in the plane, find or construct a circle tangent to all three. This can be generalized by replacing some circles with straight lines, by considering spheres in three dimensions, or even further with ellipsoids or lines [39].

Consider the classic three circles in Euclidean Two-Space. Everything we say here immediately generalizes to four spheres in Three-Space. Let the circles be S_1, S_2, and S_3. We want a circle S_0 that is tangent to each one. The centers of the circle S_i, denoted (h_i, k_i), are known for the three "input" circles, but not for S_0. Then denote by (x_i, y_i) the points of tangency between S_i and S_0, none of which are

known, for $i \in \{1, 2, 3\}$. The radius of each circle shall be r_i, which is unknown for S_0. Thus there are nine unknowns. The problem will be precisely defined by nine equations. We will now determine the variable-sharing graph.

The first constraint is that the point (x_i, y_i) must be on the circle S_i, for $i \in \{1, 2, 3\}$. This results in

$$(x_i - h_i)^2 + (y_i - k_i)^2 = r_i^2$$

and so we have an edge between x_i, y_i and y_i, z_i as well as x_i, z_i. All other terms of the above equation are known.

The second constraint is that the point (x_i, y_i) must be on the circle S_0. This results in

$$(x_i - h_0)^2 + (y_i - k_0)^2 = r_0^2$$

and so each of the variables $\{h_0, k_0, r_0\}$ is connected to each other but also to the variables $\{x_i, y_i\}$ for all $i \in \{1, 2, 3\}$. Thus the graph is connected.

The third constraint is that the slope of S_i at the point of tangency (x_i, y_i) must be equal to the slope of S_0 at that point. Namely, via implicit differentiation, we obtain

$$(x_i - h_i)(y_i - k_0) = (x_i - h_0)(y_i - k_i)$$

which only involves $\{x_i, y_i, h_0, k_0\}$, which are already connected in the graph. There will be one equation for each circle for this constraint, or nine total equations in the system for all the constraints.

Removal of the vertices $C = \{h_0, k_0, r_0\}$ will partition the graph, into three connected components of size two vertexes, namely $\{x_i, y_i\}$ for $i \in \{1, 2, 3\}$. This should make sense, because once the "output circle" S_0 is known, finding the tangency points is trivial. Furthermore, no proper subset of C will partition the graph, because if any vertex in C remains, then it is connected to all the other variables, and the graph is connected. One can further easily see that this is the optimal vertex cut.

At first it may seem strange to in effect "discard" the variables in C, as they define the solution circle, while keeping the (x_i, y_i), as they are just intermediaries. But once two of the tangency points are known (not three!), it is trivial to find the center of the solution circle: it is just the intersection of the lines through the correspondng centers (h_i, k_i) and tangencies (x_i, y_i). The desired radius is also then trivial.

Thus, the theory of this paper has application to problems even in Euclidean Geometry. Applying the theory of resultants to solving this problem in practice (and more complex generalizations of the problem) can be found in [39].

8. Analysis and Conclusion

The NP-hard problem of finding a vertex cut that is balanced can be used to assist in solving polynomial systems of equations. We have shown how to find such a cut by several methods, including converting edge-cuts from METIS via a greedy algorithm, or via repeated Depth First Searches. Meanwhile, we have shown how to exploit such a cut if it is found, either in the case of very small finite fields like GF(2) or in the case of larger fields.

This is useful for algebraic cryptanalysis in general, but also for two particular cryptographic applications. We demonstrate a security criterion: if the graph has a minor isomorphic to K_n, then it is immune to the attacks presented in this paper. Furthermore, we show that this criteria is achievable by proving that HB achieves it with overwhelming probability in any particular case. Also, we demonstrate a method by which a malicious manufacturer could create a poisoned polynomial system for QUAD, that would allow the manufacturer to be able to break the encryption and read messages transmitted with it. Finally, we show applications to Euclidean Geometry and Game Theory.

References

[1] Boinc: Berkeley open infrastructure for network computing. http://boinc.berkeley.edu/.

[2] Magma computational algebra system. http://magma.maths.usyd.edu.au/.

[3] Metis - serial graph partitioning and fill-reducing matrix ordering. http://glaros.dtc.umn.edu/gkhome/views/metis/.

[4] SINGULAR is a Computer Algebra System for polynomial computations with special emphasis on the needs of commutative algebra, algebraic geometry, and singularity theory. http://www.singular.uni-kl.de/.

[5] Publication u.s. fips pub 197, Nov. 2001.

[6] S. Al-Hinai, L. Batten, B. Colbert, and K. K.-H. Wong. Algebraic attacks on clock-controlled stream ciphers. In L. M. Batten and R. Safavi-Naini, editors, *11th Australasian Conference on Information Security and Privacy — ACISP 2006*, volume 4058 of *Lecture Notes in Computer Science*, pages 1–16, Melbourne, Australia, 2006. Springer.

[7] B. Albright. An introduction to simulated annealing. *The College Math Journal*, 38(1):37–42, 2007.

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

[9] G. Bard, N. Courtois, and C. Jefferson. 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, 2006. `http://eprint.iacr.org/2007/024.pdf`.

[10] G. V. Bard. *Algorithms for solving linear and polynomial systems of equations over finite fields with applications to cryptanalysis*. PhD thesis, University of Maryland at College Park, Applied Mathematics and Scientific Computation, 2007.

[11] G. V. Bard. Extending sat-solvers to low degree extension fields of gf(2). In *Central European Conference on Cryptography 2008*, 2008.

[12] G. V. Bard. *Algebraic Cryptanalysis*. Springer, 2009.

[13] R. Baños, C. Gil, J. Ortega, and F. G. Montoya. Multilevel heuristic algorithm for graph partitioning. In *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*. Springer, 2003.

[14] J. Berry, N. Dean, M. Goldberg, G. Shannon, and S. Skiena. Graph computation with link. *Software Practice and Experience*, 2000.

[15] D. J. B. Bo-Yin Yang, Owen Chia-Hsin Chen and J.-M. Chen. Analysis of quad. In *Fast Software Encryption*, volume 4593 of *Lecture Notes in Computer Science*, pages 290–308. Springer, 2007.

[16] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal*. Phd thesis, University of Innsbruck, 1965.

[17] J. Y. Cho and J. Pieprzyk. Algebraic attacks on SOBER-t32 and SOBER-t16 without stuttering. In B. Roy and W. Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 49–64, Delhi, India, 2004. Springer.

[18] H. G. Côme Berbain and J. Patarin. Quad: A practical stream cipher with provable security. In *Advances in Cryptology - Eurocrypt 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2006.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[20] N. Courtois. Algebraic attacks on combiners with memory and several outputs. In C. Park and S. Chee, editors, *Information Security and Cryptology - ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, Seoul, Korea, 2004. Springer.

[21] N. Courtois and W. Meier. Algebraic attacks on stream cipher with linear feedback. In E. Biham, editor, *Advances in Cryptology - Eurocrypt 2003*, volume 2656, Warsaw, Poland, 2003. Springer.

[22] R. Datta. Finding all nash equilibria of a finite game using polynomial algebra, 2006. http://arxiv.org/abs/math.AC/0612462.

[23] J.-C. Faugère. A new efficient algorithm for computer Gröbner bases (f_4). *Journal of Pure and Applied Algebra*, 139:61–88, 1999.

[24] C. Fiduccia and R. Mattheyses. A linear time heuristic for improving network partitions. In *19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[25] C. Fremuth-Paeger. Goblin: A graph object library for network programming problems, 2007. http://goblin2.sourceforge.net/.

[26] M. R. Garey, P. S. Johnson, and L. Stockmeyer. Simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[27] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan. A separation theorem for graphs of bounded genus. *Journal of Algorithms*, 5:391–407, 1984.

[28] B. Hendrickson and R. Leland. The chaco user's guide: Version 2.0the chaco user's guide: Version 2.0the chaco user's guide: Version 2.0the chaco user's guide: Version 2.0the chaco user's guide: Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, 1994.

[29] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Supercomputing*, 1995.

[30] N. J. Hopper and M. Blum. Secure human identification protocols. In *ASIACRYPT*, pages 52–66, 2001.

[31] D. S. Johnson. The NP-completeness column: An on-going guide. *J. Algorithms*, 8:438–448, 1987.

[32] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. In *International Conference on Parallel Processing*, pages 113–122, 1995.

[33] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphics. *Bell Systems Technical Journal*, 49:291–307, 1970.

[34] A. Kipnis, J. Patarin, and L. Goubin. Unbalanced oil and vinegar signature schemes. In *EUROCRYPT*, pages 206–222, 1999.

[35] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings Publishing Company, Redwood City, CA, 1994.

[36] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:271–283, 1930.

[37] R. H. Lewis. Heuristics to accelerate the dixon resultant. *Mathematics and Computers in Simulation*, 77(4):400–407, 2008.

[38] R. H. Lewis. Polynomial equations arising in global positioning systems and in nash equilibria, 2008. ACA Conference, RISC, Linz, Austria.

[39] R. H. Lewis and S. Bridgett. Conic tangency equations and apollonius problems in biochemistry and pharmacology. *Mathematics and Computers in Simulation*, 61(2):101–114, 2003.

[40] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, Apr. 1979.

[41] K. Menger. Zur allgemeinen Kurventheorie. *Fund. Math.*, 10:96–115, 1927.

[42] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In A. George, J. Gilbert, , and J. Liu, editors, *Graph Theory and Sparse Matrix Computation*, volume 56 of *The IMA Volumes in Mathematics and its Application*, pages 57–84. Springer, 1993.

[43] R. Müller and D. Wagner. α-vertex separator is NP-hard even for 3-regular graphs. *J. Computing*, 46:343–353, 1991.

[44] F. Pellegrini and J. Roman. SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *HPCN'96*, volume 1067 of *LNCS*, pages 493–498, Brussels, Belgium, 1996. Springer.

[45] R. Preis and R. Diekmann. The PARTY partitioning-library, user guide - version 1.1. Technical Report tr-rsfb-96-024, University of Paderborn, 1996.

[46] H. Raddum and I. Semaev. New technique for solving sparse equation systems. Cryptology ePrint Archive, Report 2006/475, 2006. `http://eprint.iacr.org/2006/475`.

[47] J. Rejeb, V. Ramaswamy, and K. Ghadiri. Hardware implementation of the rijndael algorithm for high-speed networks. In *International Signal Processing Conference (ISPC '03)*, March 2003.

[48] D. Rolf. *Algorithms for the Satisfiability Problem*. PhD thesis, Humboldt-Universität zu Berlin, 2006.

[49] D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *9th workshop on Design automation*, pages 57–92. ACM, 1972.

[50] B. Sturmfels. *Solving Systems of Polynomial Equations*. American Mathematical Society, October 2002.

[51] D. Wagner and F. Wagner. Between min-cut and graph bisection. Technical Report B-91-1, Freie Universität Berlin, 1991.

[52] K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Math. Ann.*, 114, 1937.

[53] C. Walshaw and M. Cross. JOSTLE: Parallel Multilevel Graph-Partitioning Software - An Overview. Technical report, Civil-Comp Ltd., 2007.

[54] K. K.-H. Wong, B. Colbert, L. Batten, and S. Al-Hinai. Algebraic attacks on clock-controlled cascade ciphers. In R. Barua and T. Lange, editors, *Progress in Cryptology - Indocrypt 2006*, volume 4329, pages 32–47, Kolkata, India, 2006. Springer.

Appendix A. NP-Completeness of the Problem

Both the edge cut and vertex cut problems are NP-hard. Specifically, the question if a graph G has a vertex cut with $|C| \leq k$ vertices, and α as defined before, is NP-Complete. Finding the smallest k such that this question answers true is therefore NP-hard. These are the α-vertex-separator decision and optimization problems, and were proven NP-Complete/NP-hard respectively by Müller and Wagner in [43]. For $\alpha = 1/2$, the problems were proven NP-Complete/NP-hard by Johnson in [31], by reduction to the known NP-Complete problem named "The Balanced Complete Bipartite Subgraph Problem."

Determining the existence of a set of edges E of cardinality $\leq k$, whose deletion from a graph will cause it to become disconnected into components V_1 and V_2, such that α is as defined before, is NP-Complete. Finding the smallest k such that this question answers true is therefore NP-hard, as before. These are the α-edge-separator decision and optimization problems, and were proven NP-Complete/NP-hard respectively by Wagner and Wagner in [51]. The $\alpha = 1/2$ case was shown to be NP-Complete/NP-hard by Garey Johnson and Stockmeyer, in [26], as "The Minimum-Bisection Problem."

Appendix B. Greedy Algorithm from Edge Cut to Vertex Cut

Given a set of c edges, that form an edge cut, there may be several sets of vertices of various cardinalities which will perform a similar vertex cut. In particular, this will happen if there is a vertex which is incident on (touches) several edges in the cut.

The following loop is a "greedy algorithm" approach to finding a fairly efficient vertex cut that is equivalent to a given edge cut. The algorithm is given in pseudocode in Table 2. One starts with a set of edges D representing the edge cut. Then, at each iteration, choose the vertex which is incident on the largest number of edges in D. Mark that vertex as "to be deleted", and then delete the edges in D that are incident upon that vertex. Repeat until D is empty.

Obviously this will produce disconnected components that are subsets of the original edge cut, but there may possibly be smaller vertex subsets which could have accomplished the same or better.

Input: C, a set of edges that forms an edge cut for the graph $G = (V, E)$.
Output: R a set of vertices that forms a vertex cut, that will produce identical disconnected components

1. $D \leftarrow C$
2. $R \leftarrow \{\}$.
3. While $D \neq \{\}$ do:
   (a) Pick the vertex in V that is incident on the highest number of edges in D. Call it v.
   (b) Insert v into R.
   (c) Remove from D any edge that v is incident upon.
4. Return R.

TABLE 2. The Greedy-Algorithm Approach to Converting a Balanced edge cut to a Balanced vertex cut.

It is unclear how to break ties in the selection of v during each iteration of the loop, as it is quite possible to imagine scenarios where several vertices are incident upon 2 edges in the edge cut, and no vertices are incident on three. One option is to break ties randomly, and run the greedy-algorithm a few hundred times and take the best outcome. This would probably be a very small difference in practice, but could conceivably reduce the cut by one vertex. Note the running times in our experiments (See Table 1) are in milliseconds, so this is quite feasible.

Appendix C. Degree Dropping by Relabeling

In the case where there are systems of degree three or higher, variable relabelling can be used to convert these systems into quadratic ones. It can be observed that any multivariate system of maximum degree d can be converted to an equivalent (but larger) quadratic system by re-labelling, in time polynomial to the size of the system for any fixed d [12]. This algorithm has been well-known and well-used, but the running time analysis and other details can be found in [12].

The mechanism is quite simple. Given any two variables in the system, that appear together in some degree 3 or higher monomial, for example x_1 and x_2, one can write a new variable x_{n+1} and the equation $x_{n+1} = x_1 x_2$. Then all degree 3 monomials $x_1 x_2 x_j$ now become $x_{n+1} x_j$. Likewise for monomials of higher degree. Thus the number of monomials of degree greater than 2 has been strictly reduced. No solutions to the old system are excluded, because whatever x_1 and x_2 are, the value of x_{n+1} can be its product. No new solutions are introduced, because every equation in the old system is still a constraint, and the new equation prevents $x_{n+1} \neq x_1 x_2$.

Therefore, repeating this algorithm a finite number of times will result in a system with no monomials above degree 2, because the number of such monomials is strictly reduced with each pass, and starts as some finite positive integer.

C.1. Preservation of Connectivity Zero

Consider now the variable-sharing graph. If it were disconnected, then surely x_1 and x_2 are in the same connected component, because they were in the same monomial. This means that an edge will be drawn between x_1 and x_{n+1} and also between x_2 and x_{n+1}. But it will only be the case that some x_j gets an edge to x_{n+1} if the x_j previously appeared in a degree 3 (or higher) monomial such as $x_1 x_2 x_{n+1}$. This means that x_j would have been in the same connected component as x_1 and x_2. Therefore, no edges are drawn between the two distinct connected components, only edges in the component containing x_1 and x_2. Thus we claim that this algorithm preserves vertex connectivity when $\kappa = 0$ (a disconnected graph).

C.2. Increasing Connectivity

Suppose that the vertex connectivity is two, and the shared variables are x_1 and x_2 between the two sets of equations. Suppose further that monomials of the form $x_1 x_2 x_j$ occur frequently, and so $x_{n+1} = x_1 x_2$ is chosen as a substitution. Then an edge from x_{n+1} to each of the two "halves" will be drawn, in addition to x_1 and x_2. Thus the smallest cut has gone from $\{x_1, x_2\}$ to $\{x_1, x_2, x_{n+1}\}$. Therefore, it is possible that the vertex connectivity could increase if it were not zero to begin with.

C.3. Practical Scenarios

However, we argue that this will be extremely rare. In fact, if the variables are divided among V_1, C, V_2, where there are no edges between V_1 and V_2, then the choice of x_i from V_1 and x_j from V_2 would never be made, because they never appear together by virtue of the fact that there is no edge between them in the variable-sharing graph.

If one variable is chosen from C and the other from V_1, then it will never appear in any equation that takes variables from C and V_2. Thus the new variable can be placed in the V_1 set. Surely this is also true if both variables are chosen from V_1.

Thus the only time the vertex connectivity of the variable-sharing graph can increase is if both of the variables are chosen from the cut C. This could be avoided explicitly, but it is also rare probabilistically if C is relatively small compared to the size of V.

Heuristically, one would expect the graph after this operation to be easier to cut, because there is not much of an increase in edges, but a significant increase in vertices. Thus the average degree of the graph is decreasing. Likewise, the average number of monomials per equation is dropping as the new equations have only two monomials. Thus this process makes the graph simpler, and one could hope, easier to cut.

Appendix D. Simulated Annealing

This vague set of techniques is quite classic, and an excellent expository article intended for mathematicians is [7].

D.1. Introduction

Imagine a problem that can be defined on a vector of integers, where there is some non-negative real function that we are trying to make zero. This non-negative function is called, for historical reasons, the "energy" function. One approach would be to simulate Gradient Descent (also known as "Steepest Descent"), by checking all vectors that are distance one away from the current vector, and taking the smallest energy vector, provided that it is an improvement. Here "distance" means the "Manhattan Norm" or L-1 norm.

Repeating this process is much like doing Gradient Descent to minimize a non-negative multivariate function, where the function output is the analog of the energy, and the gradient vector is the analog of the vector that is distance one away with the lowest energy. The analog of a local minimum is a vector whose energy is lower than all of the vectors that are distance one from it.

Note, without some property analogous to continuity, i.e. that decreasing energies lead to a path toward a lowest energy, this process is misguided and will be unlikely to work. A suitably fine-grained discretization of a continuous problem would have this pseudo-continuity property.

Instead, however, imagine we look at all vectors distance 2 or 3 away, and then take the vector with lowest energy. This can get one around "local minima". For example, if $\vec{x}_0$ is the current vector with energy 5, and $\vec{x}_1$ is distance 1 from $\vec{x}_0$ with energy 6, yet $\vec{x}_2$ is distance 2 from $\vec{x}_0$ and distance 1 from $\vec{x}_1$, with energy 4, then moving from $\vec{x}_0$ to $\vec{x}_2$ lowers the net energy. It is a "good move". But, if all vectors distance one away from $\vec{x}_0$ have energy roughly 6, then using the method described above, we would never arrive at $\vec{x}_2$, but stay at $\vec{x}_0$ forever. Thus one bypasses local minima of small "basin[3] size".

This process is known as "simulated annealing", and is inspired by the annealing in metalurgy. Annealing helps diffuse atoms of metals in alloying processes, by raising the energy level (heating the metal), and letting the energy level slowly migrate to the lowest possibility (letting the metal cool). This can be repeated. For example, the "energy" went up when we moved from $\vec{x}_0$ to $\vec{x}_1$ (from 5 to 6), but this allowed us to reach $\vec{x}_2$ where we had energy 4, a lower final state. The analogy is weak at best, because we consider all points of distance 2 or 3 away from $\vec{x}_0$, and select $\vec{x}_2$ from the neighborhood of $\vec{x}_0$ without considering its relationship with $\vec{x}_1$.

The tradeoff is that if the distance chosen is larger than 2, the search may take a very long time. This may result in fewer iterations for the same length of computing time. If small local minima are expected, a larger search may bypass them. But if few local minima are expected, then a higher number of iterations might reach a lower energy level than a smaller number of iterations.

D.2. Application to Finding Vertex Cuts

In the problem of this paper, a vector is replaced instead by a subset of the vertices S. (This can be thought of as a vector of zeros and ones, of length $|V|$, where zero means "exclude" and one means "include"). When we say "search the vectors distance 2 away", we mean try swapping out any pair of vertices in S with any two in $V - S$. Thus the distance is the cardinality of the symmetric difference of the sets, S before and S after. Or using the model of a vector of ones and zeros, it is the Hamming Distance of those vectors, before and after.

The initial condition is a valid vertex cut (balanced or not) plus a few random vertices. The goal is to convert it from an unbalanced cut to a perfectly balanced

[3]The basin of a local minimum in traditional numerical analysis is the set of points such that gradient descent begun there converges eventually to the local minimum under discussion. In this algorithm, the analog of the basin is the set of vectors for which a distance-1 search will eventually converge to the low-energy local minimum vector under discussion.

cut, if possible. The energy level is the size of the largest connected-component after deleting all the vertices in S. If S no longer contains a valid cut, then the energy is $|V - S|$, which is maximal. But if S is a valid vertex cut, it will be lower. A perfectly balanced cut will have energy $\lceil |V - S|/2 \rceil$, which is optimal.

This method was far too slow and inefficient, especially compared to sophisticated tools like Metis. The set of vectors "distance two" away was too large, and so we randomly sampled it, rather than searching it exhaustively. This improved matters but it was far too slow in any case, and so we dropped the entire approach.

Appendix E. More Experimental Results

$\|V\|$	$\|E\|$	ρ	d	$\|C\|$	$\|V_1\|$	$\|V_2\|$	α	Time
20	20	0.0952	2	1	9	10	0.5263	74.66 ms
20	30	0.1429	3	3	10	7	0.5882	70.35 ms
20	40	0.1905	4	5	9	6	0.6000	69.46 ms
20	50	0.2381	5	7	10	3	0.7692	70.98 ms
20	60	0.2857	6	9	9	2	0.8182	76.19 ms
20	70	0.3333	7	9	10	1	0.9091	80.58 ms
20	80	0.3810	8	10	7	3	0.7000	72.95 ms
20	90	0.4286	9	11	0	9	1.0000	74.25 ms
20	100	0.4762	10	10	0	10	1.0000	58.92 ms
40	40	0.0488	2	4	19	17	0.5278	58.11 ms
40	60	0.0732	3	10	19	11	0.6333	57.72 ms
40	80	0.0976	4	10	19	11	0.6333	59.28 ms
40	100	0.1220	5	14	20	6	0.7692	58.39 ms
40	120	0.1463	6	15	19	6	0.7600	57.96 ms
40	140	0.1707	7	16	16	8	0.6667	69.02 ms
40	160	0.1951	8	18	19	3	0.8636	69.51 ms
40	180	0.2195	9	18	7	15	0.6818	72.84 ms
40	200	0.2439	10	20	1	19	0.9500	70.32 ms
60	60	0.0328	2	0	32	28	0.5333	69.42 ms
60	90	0.0492	3	8	30	22	0.5769	70.18 ms
60	120	0.0656	4	16	24	20	0.5455	70.58 ms
60	150	0.0820	5	19	24	17	0.5854	71.00 ms
60	180	0.0984	6	20	29	11	0.7250	71.66 ms
60	210	0.1148	7	22	24	14	0.6316	71.33 ms
60	240	0.1311	8	26	12	22	0.6471	72.26 ms
60	270	0.1475	9	27	30	3	0.9091	61.75 ms
60	300	0.1639	10	27	30	3	0.9091	72.81 ms
80	80	0.0247	2	4	40	36	0.5263	61.71 ms
80	120	0.0370	3	13	39	28	0.5821	58.73 ms
80	160	0.0494	4	21	37	22	0.6271	58.83 ms
80	200	0.0617	5	25	34	21	0.6182	70.27 ms
80	240	0.0741	6	27	36	17	0.6792	74.13 ms
80	280	0.0864	7	32	35	13	0.7292	72.76 ms
80	320	0.0988	8	34	29	17	0.6304	76.81 ms
80	360	0.1111	9	34	15	31	0.6739	75.50 ms
80	400	0.1235	10	39	38	3	0.9268	72.55 ms
100	100	0.0198	2	5	49	46	0.5158	69.30 ms
100	150	0.0297	3	14	48	38	0.5581	71.41 ms
100	200	0.0396	4	19	49	32	0.6049	71.74 ms
100	250	0.0495	5	29	40	31	0.5634	72.78 ms
100	300	0.0594	6	32	41	27	0.6029	75.59 ms
100	350	0.0693	7	40	47	13	0.7833	73.98 ms
100	400	0.0792	8	44	31	25	0.5536	74.35 ms
100	450	0.0891	9	44	47	9	0.8393	79.04 ms
100	500	0.0990	10	48	48	4	0.9231	88.69 ms

TABLE 3. More Experiments

$\|V\|$	$\|E\|$	ρ	d	$\|C\|$	$\|V_1\|$	$\|V_2\|$	α	Time
120	120	0.0165	2	6	59	55	0.5175	70.06 ms
120	180	0.0248	3	21	59	40	0.5960	70.90 ms
120	240	0.0331	4	26	54	40	0.5745	72.52 ms
120	300	0.0413	5	36	54	30	0.6429	73.43 ms
120	360	0.0496	6	39	45	36	0.5556	63.93 ms
120	420	0.0579	7	50	47	23	0.6714	65.50 ms
120	480	0.0661	8	50	35	35	0.5000	65.07 ms
120	540	0.0744	9	52	31	37	0.5441	66.05 ms
120	600	0.0826	10	54	22	44	0.6667	66.48 ms
140	140	0.0142	2	8	69	63	0.5227	70.93 ms
140	210	0.0213	3	20	68	52	0.5667	73.76 ms
140	280	0.0284	4	33	66	41	0.6168	73.80 ms
140	350	0.0355	5	41	63	36	0.6364	78.15 ms
140	420	0.0426	6	47	60	33	0.6452	74.65 ms
140	490	0.0496	7	54	50	36	0.5814	79.88 ms
140	560	0.0567	8	59	67	14	0.8272	67.15 ms
140	630	0.0638	9	61	63	16	0.7975	66.69 ms
140	700	0.0709	10	65	57	18	0.7600	75.43 ms
160	160	0.0124	2	8	80	72	0.5263	71.23 ms
160	240	0.0186	3	25	79	56	0.5852	73.84 ms
160	320	0.0248	4	36	74	50	0.5968	75.24 ms
160	400	0.0311	5	43	66	51	0.5641	75.41 ms
160	480	0.0373	6	59	66	35	0.6535	77.78 ms
160	560	0.0435	7	61	67	32	0.6768	77.57 ms
160	640	0.0497	8	65	65	30	0.6842	71.26 ms
160	720	0.0559	9	72	46	42	0.5227	67.61 ms
160	800	0.0621	10	75	28	57	0.6706	76.35 ms
180	180	0.0110	2	6	89	85	0.5115	64.23 ms
180	270	0.0166	3	28	87	65	0.5724	65.60 ms
180	360	0.0221	4	34	85	61	0.5822	73.36 ms
180	450	0.0276	5	50	86	44	0.6615	64.37 ms
180	540	0.0331	6	65	84	31	0.7304	68.36 ms
180	630	0.0387	7	69	82	29	0.7387	68.46 ms
180	720	0.0442	8	79	87	14	0.8614	67.67 ms
180	810	0.0497	9	80	83	17	0.8300	71.01 ms
180	900	0.0552	10	84	88	8	0.9167	76.07 ms
200	200	0.0100	2	13	99	88	0.5294	61.56 ms
200	300	0.0149	3	30	99	71	0.5824	72.98 ms
200	400	0.0199	4	46	95	59	0.6169	74.56 ms
200	500	0.0249	5	54	85	61	0.5822	75.95 ms
200	600	0.0299	6	67	91	42	0.6842	69.84 ms
200	700	0.0348	7	73	72	55	0.5669	69.95 ms
200	800	0.0398	8	80	90	30	0.7500	73.63 ms
200	900	0.0448	9	86	48	66	0.5789	77.18 ms
200	1000	0.0498	10	93	96	11	0.8972	73.69 ms

TABLE 4. Still More Experiments

Kenneth Koon-Ho Wong
Information Security Institute
Queensland University of Technology
2 George Street
Brisbane 4001, Australia

e-mail: kk.wong@qut.edu.au

Gregory V. Bard
Department of Mathematics
Fordham University
The Bronx, NY, 10428

e-mail: bard@fordham.edu

Robert H. Lewis
Department of Mathematics
Fordham University
The Bronx, NY, 10428

e-mail: rlewis@fordham.edu