Graph theoretic obstacles to perfect hashing

George Havas* and Bohdan S. Majewski†
Department of Computer Science
Key Centre for Software Technology
The University of Queensland
Queensland 4072, Australia

Abstract

A number of algorithms based on quasi-random graphs for generating perfect hash functions have been published. These include Sager's mincycle algorithm, a modification by Fox *et al.* of it and finally probabilistic methods due to Czech, Havas and Majewski. Each of these algorithms exploits different properties of graphs, such as being bipartite or being acyclic. In this paper we formally justify the significance of these properties. Also we indicate causes of failure for some methods. In particular we show that acyclicity of a graph plays a crucial role in finding order preserving perfect hash functions. It is a sufficient, but not necessary, condition for algorithms to actually find a perfect hash function. We provide some examples for which various published methods methods fail, taking exponential time to do so. Finally, based on our considerations of graph properties, we propose yet another method for generating perfect hash functions.

1 Introduction

A *hash function* is a function $h : W \to I$ that maps the set of keys $W \subset U$ into some given interval of integers I, say $[0, k-1]$, where $k > 0$. U is the universe from which keys in W have been selected, $U = \{0, \ldots, u-1\}$. The hash function, given a key, computes an address (an integer from I) for the storage or retrieval of that item. The storage area used to store items is known as a *hash table*. Keys for which the same address is computed are called *synonyms*. Due to the existence of synonyms a situation called *collision* may arise in which two keys are mapped into the same address. Several schemes for resolving collisions are known. A *perfect* hash function is an injection $h : W \to I$, where W and I are sets as defined above, $k \geq m$. If $k = m$, then h is a *minimal* perfect hash function. As the definition implies, a perfect hash function transforms each key of W into a unique address in the hash table. Since no collisions occur each item can be retrieved from the table in a single probe. If for any two keys from W, w_i and w_j, $i < j$ implies that $h(w_i) < h(w_j)$ then the hash function is order preserving. (In other words, the relative order in the input set is preserved in the hash table.)

Minimal perfect hash functions are used for memory efficient storage and fast retrieval of items from a static set, such as reserved words in programming languages, command names in operating systems, commonly used words in natural languages, etc. An overview of perfect hashing is given in [GBY91, §3.3.16], and the area is surveyed in [LC88, HM92].

*E-mail: havas@cs.uq.oz.au. The author was supported in part by Australian Research Council grant A49030651

†E-mail: bohdan@cs.uq.oz.au

Various algorithms with different time complexities have been presented for constructing perfect or minimal perfect hash functions. Some, arguably the most important algorithms, are based on mapping the set W into a graph, which then is processed to obtain a perfect hash function. These include Sager's mincycle algorithm [Sag85], its modifications due to Fox, Heath, Chen and Daoud [FCHD88, FHC89, FCDH91, FHCD92] and Czech and Majewski [CM92], and probabilistic methods due to Czech, Havas, Majewski and Wormald [CHM92, MWCH92]. Even the famous FKS scheme [FKS84] can be viewed as a method of constructing and coding a special type of acyclic graph.

In this paper we investigate these methods. In particular we concentrate our attention on the properties of graphs used by the algorithms. We formally justify the significance of such properties as being bipartite or being acyclic in relation to perfect and order preserving perfect hash functions. We identify a graph theoretic cause for failure of some of the methods. We provide examples for which the methods of Sager, of Fox *et al.* and of Czech and Majewski fail and run in exponential time. Finally, based on our considerations of graph properties, we suggest yet another method for generating perfect hash functions.

2 Keys, graphs and hash tables

Graphs prove to be an excellent tool in building hash tables for a given set of keys, especially if these tables need to be perfect. Let us briefly look at some methods which comply with this approach.

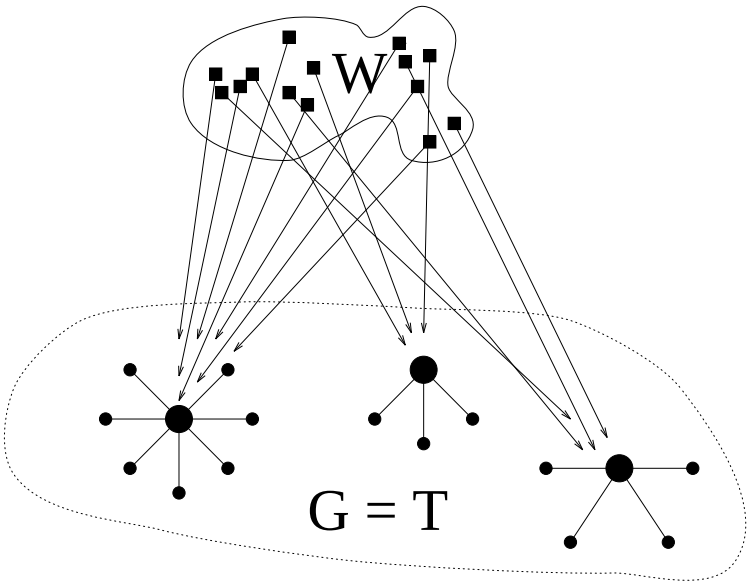

Figure 1: The FKS method. The keys from W are mapped into primary vertices. Keys with the same primary vertex are mapped into unique vertices forming a star. The resulting graph G defines the hash table T for W.

The earliest method which used graphs was published by Fredman, Komlós and Szemerédi [FKS84]. Although they did not cast their algorithm in graph theoretic terms, it is easy to do so. In the first step, a set of m keys is mapped into m primary vertices using a polynomial of degree 1, $f_1(x) = (kx \bmod u) \bmod m$. For each group of keys that have been mapped into the same primary vertex, a second level polynomial is selected, $f_{2,i}(x) = (k'_i x \bmod u) \bmod 2b_i^2$, where b_i is the size of the group and $0 \leq i < m$. The second function maps each key in the

group into a unique vertex. The resulting structure is an acyclic graph, which is a union of star shaped trees, as shown in Fig. 1. Fredman, Komlós and Szemerédi provide a structure that efficiently encodes the resulting graph and prove that the number of the vertices, which determines the size of the structure, does not exceed $11m$. As the m primary vertices need to store parameters of the second level polynomials, the total size of the structure is $13m$. Fredman, Komlós and Szemerédi mention some refinements that allow a reduction in the size of the encoding.

The next four methods, which use significantly less space than FKS, use similar approaches. First a set of keys is mapped explicitly into a graph. Then, using different methods, the authors try to solve the following problem, which we call the perfect assignment problem. For a given undirected graph $G = (V, E)$, $|E| = m$, $|V| = n$ find a function $g : V \to [0, m-1]$ such that the function $h : E \to [0, m-1]$ defined as

$$h\left(e = (u, v) \in E\right) = \Big(\alpha(e) + g(u) + g(v)\Big) \bmod m$$

is a bijection. The α function maps edge e into a constant. In other words the problem asks for an assignment of values to vertices so that, for each edge, the sum of values associated with its endpoints (plus a constant), taken modulo the number of edges, is a unique integer in the range $[0, m-1]$. A perfect hash function generated by each of these methods can be stored in $n + O(1)$ words.

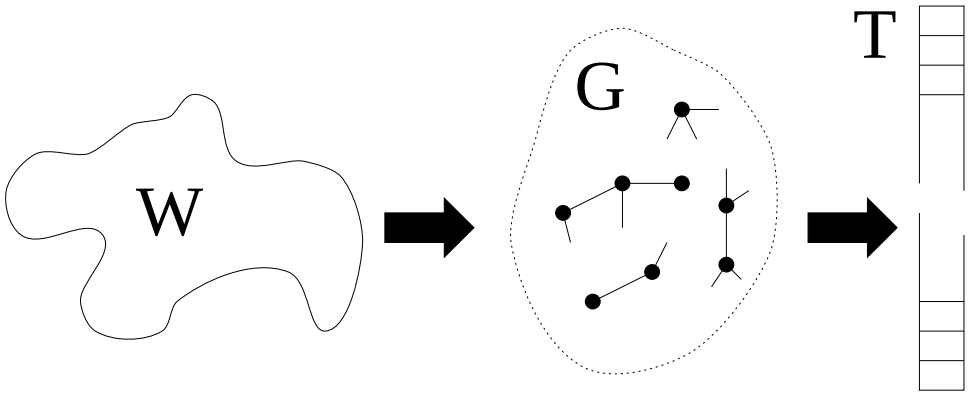

Figure 2: A graph based method of computing a perfect hash function. The input set W is mapped into graph G which then, after some processing, yields the hash table T for the keys in W.

Sager's mincycle algorithm consists of three steps. In step 1, the set W is mapped into a graph G, called the dependency graph. Sager chooses the number of vertices to be twice the smallest power of 2 greater than $m/3$. The dependency graph is always generated to be bipartite. The graph may contain cycles. In practice, because of its density and the poor quality of the mapping functions used by Sager, most of the edges belong to one or more cycles. Cycles, as we will see in the next section, may cause the perfect assignment problem to be unsolvable and they certainly make it difficult. To find a solution to the assignment problem, Sager uses an exhaustive search procedure to go through all feasible functions g. To speed up the search a certain order is imposed on the edges of the dependency graph in the second step of the algorithm. This arrangement dictates the order in which the searching procedure tries to solve the perfect assignment problem. The search may result in success or, after examining all feasible functions g, report a failure. Sager argues that the ordering step causes the exhaustive search to run in polynomial time. However his algorithm runs in

exponential time whenever there is no solution to the assignment problem, since the number of feasible functions is exponentially large.

Fox, Heath, Chen and Daoud [FHCD92] introduce three major changes to Sager's method. Better mapping functions are introduced, which mimic random functions more closely than the old functions. Consequently, the mapping step creates a graph that is closer to a random graph. Fox *et al.* keep the bipartite property of the graph, without elaborating on its importance. The ordering step uses a less time consuming method, which, despite the lower quality of its result, gives reasonable performance. In the searching step the authors employ a probabilistic algorithm, which generates random 'probe sequences' determining a possible arrangement of the keys in the hash table. The Fox *et al.* algorithm, in the form described in [FHCD92], runs in exponential time if no solution to the perfect assignment problem can be found. The authors mention a modification that may avoid an exponential time search, discussed in our Section 4. Fox, Heath, Chen and Daoud choose the number of vertices to be roughly $0.6m$.

Czech and Majewski [CM92] suggest yet another heuristic for the ordering step. They also modify the exhaustive search, introducing backtrack pruning in order to make it more efficient. Otherwise their method follows the same path as the methods of Sager and Fox *et al.*

Czech, Havas and Majewski [CHM92] set $\alpha(e) = 0$ for all $e \in E$ and insist on G being acyclic. They show that if $n = cm$ with $c > 2$ the probability of obtaining an acyclic mapping is a non-zero constant. Hence this can be done in linear time by a probabilistic algorithm. They also provide a method of finding function g in linear time for acyclic graphs. Because of the acyclicity of G, the resulting perfect hash function can be made order preserving and minimal. In [MWCH92] they extend their method to hypergraphs. This allows a reduction in the size of the program that evaluates the hash function to about 58% of the size of the program obtained in the case of standard graphs. We stress that despite the fact that graphs in the FKS and CHM methods are acyclic in both cases, the methods are not equivalent. Fredman *et al.* require that for each edge in G there is a unique vertex. This is not so in the CHM method, and therefore we need an auxiliary tool, like function g, to distinguish between the keys. The greater density of graphs in the CHM method allows a substantial reduction in the size of the resulting perfect hash function. Moreover maintaining the acyclicity property makes the method very flexible.

Thus different authors use different properties to achieve their goal of finding a perfect hash function. The question we address in this paper is how important are these properties. Can some of them be weakened, or may we gain something extra by exploiting them more thoroughly?

3 The perfect assignment problem is not always solvable

Fredman, Komlós and Szemerédi proved that their method is guaranteed to work. (One major disadvantage of their method is its relatively large space requirements.) Hence we concentrate on the perfect assignment problem. The problem, perhaps not surprisingly, cannot be always solved. This section provides a few examples of infinite classes of graphs for which we prove that no solution exists. For simplicity we assume that $\alpha(e) = 0$ for all $e \in E$.

A very simple example of a graph for which there is no solution to the assignment problem is a unicyclic graph, C_{4j+2}, $j > 1$. Such a graph has $4j + 2$ edges and $4j + 2$ vertices. The edges are assigned numbers between 0 and $m - 1 = 4j + 1$. Hence the total sum of the values assigned to the edges is $(4j + 2)(4j + 1)/2 = (2j + 1)(4j + 1)$. This number is clearly odd. On the other hand, each vertex is adjacent to two edges. Thus the sum of values on the edges is $\sum_{\{v,u\}\in E}(g(u) + g(v)) \bmod m = (2\sum_{v\in V} g(v)) \bmod m$ which must be an even number — a

contradiction.

Another infinite family is formed by complete graphs on $8k+5$ vertices, $k>0$. Again, by the parity argument, we can show that there is no assignment of values $0\,..\,m-1$ to edges such that a suitable function g can be found. The essence of the above examples can be stated as the following theorem.

Theorem 1 *For any graph G with $m=4j-2$ edges, $j>0$, such that all the vertices of G have even degrees, there is no solution to the assignment problem.*

Proof. Consider the sum of the values assigned to the edges, $\sum_{\{v,u\}\in E}(g(v)+g(u)) \bmod m$. If the number of edges is even, that is $m=2k$, and the degrees of all the vertices of G are even clearly the above sum must be even. On the other hand this sum must be equal to $m(m-1)/2 = k(2k-1)$. In order to make the assignment problem unsolvable, the last expression needs to be odd, that is k must be odd, $k=2j-1$. Hence $m=4j-2$, and for such m we always get a parity disagreement. □

An elegant test can be constructed to verify if the assignment problem can be solved for a cycle with particular values assigned to its edges. Given a cycle $C_k=(v_1,v_2,\ldots,v_k)$, let the edges of cycle C_k be assigned the values $t_1, t_2, \ldots, t_k$, so that the edge $\{v_j, v_{j+1}\}$ is assigned t_j. For vertex v_1 construct any permutation $\pi(v_1)=\langle i_1^1, i_2^1, \ldots, i_k^1\rangle$, of the numbers from 0 to $k-1$. For vertex v_2 compute the permutation $\pi(v_2)=\langle i_1^2, i_2^2, \ldots, i_k^2\rangle$, such that $i_j^2=(t_1-i_j^1) \bmod k$. For vertex v_p, $1<p\le k+1$ construct the permutation $\pi(v_p)=\langle (t_{p-1}-i_1^{p-1}) \bmod k, \ldots, (t_{p-1}-i_k^{p-1}) \bmod k\rangle$. The cycle has a solution to the assignment problem if there exists a j such that $i_j^{k+1}=i_j^1$. Tracing the whole process back we quickly discover that the cycle has a solution if there is a j such that $(t_k-t_{k-1}+\cdots+(-1)^{k-1}t_1+(-1)^k i_j) \bmod k = i_j^1$. To show that the last condition is both sufficient and necessary assume that the i-th vertex of cycle C_k is assigned the value $g(v_i)$. The values on the edges are $(g(v_i)+g(v_{i \bmod k+1})) \bmod m = t_i$, for $m>k$ and $1\le i\le k$. We observe that $g(v_{i \bmod k+1}) = t_i - g(v_i) \pmod m$, $1\le i\le k$. In particular we have

$$g(v_j) = t_{j-1} - g(v_{j-1}) = \sum_{p=1}^{j-1}(-1)^{j-1-p}t_p + (-1)^{j+1}g(v_1) \pmod m$$

for $2\le j\le k$. Using the above to evaluate $g(v_k)$ in $(g(v_k)+g(v_1)) = t_k \pmod m$ yields

$$\sum_{p=1}^{k-1}(-1)^{k-1-p}t_p + (-1)^{k+1}g(v_1) + g(v_1) = t_k \pmod m$$

so

$$(-1)^{k+1}g(v_1) + g(v_1) = \sum_{p=1}^{k}(-1)^{k-p}t_p \pmod m$$

Hence we have the following technical result, which is used to derive a practical corollary and effective applications:

Theorem 2 *For a cycle $C_k=(v_1,v_2,\ldots,v_k)$ and an integer m there exists a function g which maps each vertex into some number in the range $[0,m-1]$ such that for the edge $\{v_p,v_{p+1}\}$, $(g(v_p)+g(v_{p+1})) \bmod k = t_p$, where each t_p is a number in the range $[0,m-1]$, if and only if for some integer $j\in[0,m-1]$*

$$\left(\sum_{p=1}^{k}(-1)^{k-p}t_p + (-1)^k j\right) \equiv j \pmod m$$

From the above theorem we easily deduce the following corollary:

Corollary 3 *Let* $x = \sum_{p=1}^{k}(-1)^{k-p}t_p$, *where* $t_p \in \{0, \ldots, m-1\}$ *for* $1 \leq p \leq k$ *are the values assigned to the edges of some cycle* C_k. *If* k *is even then there exists a solution to the assignment problem if and only if* $x \equiv 0 \pmod{m}$. *If* k *is odd then a solution exists if and only if* x *is even or both* x *and* m *are odd.*

The above corollary provides an quick and easy test. Unfortunately it does not give us a method for choosing t_p's so that the final condition is fulfilled. Notice that cycles of even length are in some sense more difficult to deal with than cycles of odd length. However, once the condition on the alternating sum for an even length cycle is fulfilled, it is much easier to assign values to its vertices. A similar procedure to that presented for acyclic graphs may be used (see [CHM92]). For odd length cycles, it is much easier to satisfy the condition for the alternating sum, but then there are at most 2 valid assignments to the vertices.

Corollary 3 provides us with another proof of Theorem 1 for the case of unicyclic graphs. First we notice that any alternating sum can be split into two parts, the positive part P and the negative part N. From any sum, to obtain a different sum we must exchange at least one element from the positive part with another element from the negative part. Call these elements π and ν. The initial sum is equal to $P - N$, while the modified sum will be equal to $P - N + 2(\nu - \pi)$. Thus the minimum change must equal 2 or be a multiple of 2. Now consider the maximum sum for the case of a unicyclic graph C_{4j+2}. The maximum alternating sum is equal to $(6j+2)(2j+1)/2 - (2j+0)(2j+1)/2 = (2j+1)^2$. This sum is odd. Hence it is not divisible by $4j+2$. Moreover, as we may modify it only by exchanging elements between the positive and the negative part, all other sums will be odd too. Consequently none of them is divisible by $4j+2$. Hence by Corollary 3 there is no solution for unicyclic graphs if $m = k$ and $k = 4j+2$. Without actually trying to do so, we have discovered the fact that an alternating sum of the integers $\{0, 1, \ldots, k-1\}$, for any permutation of them, must either always be odd (if $k = 4j+2$ or $k = 4j+3$) or always be even (if $k = 4j$ or $k = 4j+1$).

We can actually prove a stronger version of Theorem 2. Before we do this we need some definitions from graph theory. A *path* from v_1 to v_i is a sequence $P = v_1, e_1, v_2, e_2, \ldots, e_{i-1}, v_i$ of alternating vertices and edges such that for $1 \leq j < i$, e_j is incident with v_j and v_{j+1}. If $v_1 = v_i$ then P is said to be a *cycle*. In a simple graph (a graph with no self-loops or multiple edges) a path or a cycle $v_1, e_1, v_2, e_2, \ldots, e_{i-1}, v_i$ can be more simply specified by the sequence of vertices $v_1, \ldots, v_i$. If in a path each vertex appears once, then the sequence is called a *simple* path. If each vertex appears only once except that $v_1 = v_i$ then P is called a *simple* cycle. Because we are interested only in simple paths and cycles, these are abbreviated and the word "simple" is understood. A graph is *connected* if there is a path joining any pair of its distinct vertices. A vertex v is said to be an *articulation point* of a connected graph G if the graph obtained by deleting v and all its adjacent edges is not connected. G is said to be *biconnected* if it is connected and it contains no articulation points, that is, any two vertices of G are connected by at least two distinct paths. Two cycles, C_1 and C_2, can be added to form their sum, $C_3 = C_1 \oplus C_2$, where operation $\oplus$ is the operation of ring-sum. The ring sum of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, is the graph $((V_1 \cup V_2), ((E_1 \cup E_2) - (E_1 \cap E_2))$. The set of all cycles is closed under the operation of ring-sum, forming the *cycle space*. A *cycle basis* of G is a maximal collection of independent cycles of G, or a minimal collection of cycles on which all cycles depend. In other words, no cycle in a cycle basis can be obtained from the other cycles by adding any subset of them. There are special cycle bases of a graph which can be derived from spanning trees of a graph. Let F be a spanning forest of a graph G. Then, the set of cycles obtained by inserting each of the remaining edges of G into F is a

fundamental cycle set of G with respect to F.

Theorem 4 *Let $G = (V, E)$ be a biconnected graph with $m = |E| > 2$ edges. Let each edge $e \in E$ be assigned a value $t_e \in [0, m-1]$. Let B be a cycle basis of G with the additional property that each cycle in B goes through a selected vertex in G, say v_1. If there exists an integer $j \in [0, m-1]$ such that for every cycle $C_k \in B$, $C_k = (e_1, e_2, \ldots, e_k)$, $(\sum_{p=1}^{k}(-1)^{k-p}t_{e_p} + (-1)^k j) \equiv j \pmod{m}$, then there exists a function $g : V \to \{0, \ldots, m-1\}$ such that for each edge $e = \{u, v\}$, $e \in E$ the function $(g(u) + g(v)) \bmod m$ is equal to t_e.*

Proof. The above result is easy to prove by induction. By Theorem 2, it is true if G consists of one cycle. Suppose it is true if B of G contains $i > 1$ cycles. Add a new cycle to B that has at least one edge, but at most one path in common with some cycle in B. (In the following argument we restrict our attention to cycles with the above specified property. In doing so we lose no generality, as a cycle that has more than one disjoint path in common with some other cycle or cycles can always be modelled by a number of cycles, such that each of them shares just one path with one other cycle. The case when a new cycle has just one vertex in common with a basis cycle is trivial, therefore we omit it in our analysis.) If the sum test fails for the new cycle then G has no solution to the assignment problem. Hence suppose that the sum test is true. Graph G has no solution to the assignment problem if and only if there exists a cycle that does not belong to B for which the sum test fails. To show that this is impossible consider the graph in Fig. 3.

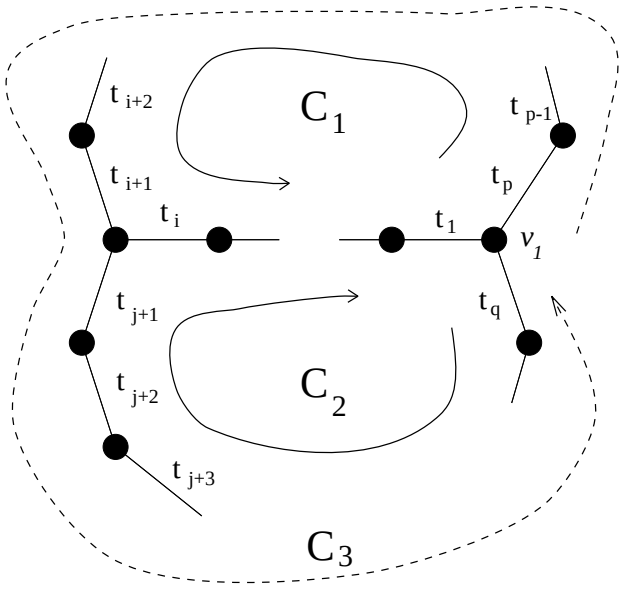

Figure 3: The alternating sum for $C_3 = C_1 \oplus C_2$

Cycles C_1 and C_2 are basis cycles, cycle C_3 can be created by adding C_1 and C_2. The alternating sum for C_1 is $s_1 = \sum_{k=0}^{p-1}(-1)^k t_{p-k}$ and the alternating sum for C_2 is $s_2 = \sum_{k=0}^{q-j-1}(-1)^k t_{q-k} + \sum_{k=0}^{i-1}(-1)^{q-j+k} t_{i-k}$. The alternating sum for C_3 is $s_3 = \sum_{k=0}^{p-i-1}(-1)^k t_{p-k} + \sum_{k=j+1}^{q}(-1)^{p-i-1+k-j} t_k$. If the length of cycle C_1 is odd and the length of cycle C_2 is even then $s_3 = s_1 + s_2$. By Corollary 3 and the statement of Theorem 4 we know that s_1 was either even or of the same parity as m, and s_2 was a multiple of m. Cycle C_3 must be of odd length and naturally $s_1 + s_2$ is either even or of the same parity as m. If the lengths of both cycles are of the same parity, then $s_3 = s_1 - s_2$ and again the condition on s_3 is fulfilled. It is also

easy to see that the combination even/odd ($s_3 = s_1 + s_2$, again) preserves the condition on the alternating sum of C_3. Therefore, if all cycles in a basis pass the sum test, so will any non-basis cycle. □

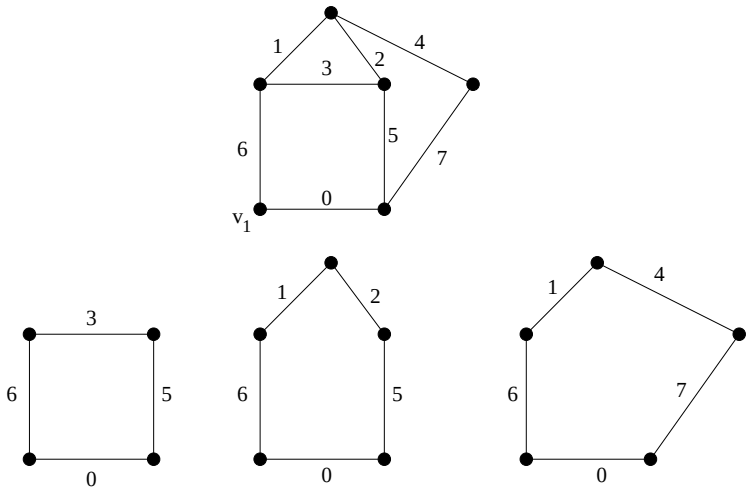

Figure 4: A sample graph and its cycle basis

Example. Consider the graph in Fig. 4. Its cycle basis based on v_1 consists of three cycles, shown at the bottom. To verify if for the given values on the edges it is possible to find the appropriate values for vertices we need to calculate the alternating sums for the three basis cycles. They are equal to $6 - 3 + 5 - 0 = 8$, $6 - 1 + 2 - 5 + 0 = 2$ and $6 - 1 + 4 - 7 + 0 = 2$. Using Theorem 4 we promptly discover that for all three cycles a suitable j is 1 or 5. Hence we conclude that there exists a suitable assignment for the graph in Fig. 4. Indeed we may even give a method for obtaining such an assignment. We start by setting $g(v_1)$ to j. Then we perform a regular search on the graph, say depth-first search, starting with vertex v_1. Whenever we visit a new vertex u, which we have reached via the edge $e = \{v, u\}$ we set $g(u) := (t_e - g(v)) \bmod m$. The method is a straightforward modification of the method given for acyclic graphs (see [CHM92]). Using this method we can easily find function g for the graph in Fig. 4. □

The requirement that each cycle in B has to run through a specific vertex is necessary because of the odd length cycles. For each of them we not only need the permutation to agree on some position, but additionally two permutations for two different cycles have to agree on the same position. One way to ensure that is to make all cycles have at least one vertex in common, hence for all of them the starting permutation is the same. If a graph is a bipartite graph, that is all its cycles are of even length, then we may restrict our attention only to its cycles. Once for each cycle the condition on the alternating sum is satisfied we may freely choose the the values for edges that do not belong to any cycle and still be able to solve the assignment problem. This is expressed in the next theorem which we state without a proof, as it follows easily from the two theorems and the corollary given before.

Theorem 5 *Let $G = (V_1 \cup V_2, E)$ be a bipartite graph with $m = |E|$ edges. Let each edge $e \in E \subseteq V_1 \times V_2$ be assigned a unique value $t_e \in [0, m-1]$. If for every biconnected component H_i of G and each cycle $C_k = (e_1, e_2, \ldots, e_k) \in B_i$, where B_i is a cycle basis of H_i, $\sum_{p=1}^{k} (-1)^{k-p} t_{e_p} \equiv 0 \pmod m$ and G has no multiple edges, then there exists a solution to the assignment problem for G.*

4 Consequences

Theorem 5 justifies the use of bipartite graphs by the algorithms of Sager [Sag85], Fox *et al.* [FHCD92] and Czech and Majewski [CM92]. All of them consider cyclic edges independently of the other edges. A solution of the perfect assignment problem for all of the cyclic components can be trivially extended to acyclic parts of the dependency graph if the graph is bipartite. If the graph has cycles of odd length it might be impossible to do so. Hence the process of finding a perfect hash function is greatly simplified when bipartite graphs are used. Otherwise the whole structure needs to be considered.

In the case of order preserving hash functions, acyclicity is a sufficient condition for the function h to be order preserving. In the presence of cycles the order preserving property cannot be guaranteed. Some graphs (like K_4) resist some assignments, while allowing others. Others, C_6 might serve as a simple example, allow no solution to the perfect assignment problem. The class of graphs which are unsolvable for some assignments is much larger than the class of graphs for which there is no solution to the assignment problem. Note that inability for a graph to have arbitrary numbers assigned to its edges excludes it from the class of graphs suitable for generating an order preserving perfect hash function. The probabilistic method [CHM92], by rejecting these types of graphs, can always succeed in solving the assignment problem for the graph output by the mapping step.

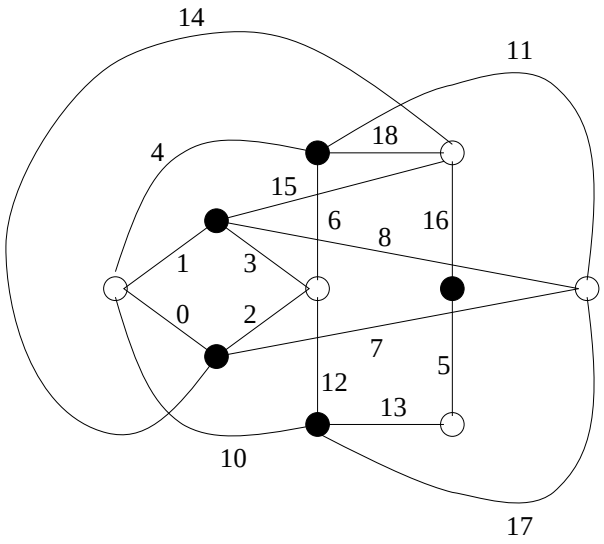

Figure 5: An example of the dependency graph for which the methods of Sager, Fox *et al.* and Czech and Majewski are certain to fail. The numbers on the edges give a minimum possible assignment.

By Theorem 1 we know that if the number of edges in the dependency graph is $4j - 2$, $j > 0$, and all vertices have even degrees there is no solution to the assignment problem. This allows us to construct examples of dependency graphs for which the methods of Sager [Sag85], Fox *et al.* [FHCD92] and Czech and Majewski [CM92] are certain to fail. One such an example, with $m = 18$ edges and $n \approx 0.6m = 10$ vertices, is given in Fig. 5. We ran an exhaustive search along the lines of those specified by Sager [Sag85] and Czech and Majewski [CM92] on this example. It made $16,259,364$ iterations before discovering that there is no solution to the assignment problem for this graph. An upper bound on the number of iterations executed by the exhaustive search procedure in the above mentioned algorithms is $\sum_{i=1}^{m-n+p(G)} m!/(m-i)!$,

where $p(G)$ is the number of connected components of the dependency graph G.

We have also found an optimal perfect mapping for this graph, shown in Fig 5. The mapping maps the 18 edges into the 19 element range [0, 18]. It is easy to verify that for any cycle of the graph in Fig. 5 the given assignment meets the conditions specified by the theorems in the previous section.

Let us investigate the probability that a dependency graph generated in the mapping step belongs to the class of graphs characterized by Theorem 1. From now on we assume that the number of edges m is even but not divisible by 4. Notice that our question can be reformulated as what is the probability that each connected component of a random graph has an Eulerian cycle. Such graphs are called even graphs.

Even graphs have been enumerated by R. Read [Rea62]. Modifying his approach for bipartite graphs with $n = 2r$ vertices, r vertices in each part, we get that the number of even graphs is given by the following expression:

$$E(G_b) = 2^{-2r} \sum_{s=0}^{r} \binom{r}{s} \sum_{t=0}^{r} \binom{r}{t} \sum_{j=0}^{m} (-1)^j \binom{s(r-t)+t(r-s)}{j} \binom{st+(r-s)(r-t)}{m-j}$$

for bipartite graphs, and

$$\begin{aligned} E(G'_b) &= 2^{-2r} \sum_{s=0}^{r} \binom{r}{s} \sum_{t=0}^{r} \binom{r}{t} \\ &\times \sum_{j=0}^{m} (-1)^j \binom{s(r-t)+t(r-s)+j-1}{j} \binom{st+(r-s)(r-t)+m-j-1}{m-j} \end{aligned}$$

for bipartite multigraphs with no loops. The above expression allow us to compute that there are 1479792175 even bipartite multigraphs with 18 edges and 10 vertices. Consequently, finding the one in Fig. 5 was a relatively simple task. However, the even bipartite multigraphs constitute only about 0.42% of the entire population of bipartite multigraphs with 18 edges and 10 vertices.

We were not able to find a reasonable general approximation of the formula for $E(G'_b)$. However, for $2r \leq m$, we observe that $E(G'_b)/\binom{r^2+m-1}{m} \approx 2^{-2r+2}$. This approximation can be justified by the following simple argument. The probability that a given vertex v in a bipartite graph with r vertices in each part has degree d is $\Pr(\deg(v) = d) = \binom{m}{d} \left(\frac{1}{r}\right)^d \left(1 - \frac{1}{r}\right)^{m-d}$. Consequently, the probability that this degree is even is

$$\Pr(\deg(v) \bmod 2 = 0) = \sum_{d \geq 0} \binom{m}{2d} \left(\frac{1}{r}\right)^{2d} \left(1 - \frac{1}{r}\right)^{m-2d}$$

As $(x+y)^m = \sum_{k=0}^{m} \binom{m}{k} x^k y^{m-k}$, selecting $x = 1/r$ and $y = (1 - 1/r)$ and $x = -1/r$ and $y = (1 - 1/r)$ and adding both equations gives

$$\begin{aligned} \left(\frac{1}{r} + \left(1 - \frac{1}{r}\right)\right)^m + \left(\frac{-1}{r} + \left(1 - \frac{1}{r}\right)\right)^m &= 2 \sum_{k \geq 0} \binom{m}{2k} \left(\frac{1}{r}\right)^{2k} \left(1 - \frac{1}{r}\right)^{m-2k} \\ \frac{1 + \left(1 - \frac{2}{r}\right)^m}{2} &= \sum_{k \geq 0} \binom{m}{2k} \left(\frac{1}{r}\right)^{2k} \left(1 - \frac{1}{r}\right)^{m-2k} \end{aligned}$$

Thus, as $r \to \infty$, any vertex has approximately $\frac{1}{2}$ chances of having an even degree. However, once the degrees for $r - 1$ vertices in each part of a bipartite graph are fixed, the degree of

the last vertex in each part is determined. (Actually, 'fixing' the degree of any vertex modifies the probabilities for the remaining vertices. But as long as we have 'enough' edges, the above argument gives a reasonable approximation.)

As a consequence, the claimed polynomial time complexities of the algorithms of Sager [Sag85], Fox, Heath, Chen and Daoud [FHCD92] and Czech and Majewski [CM92] are false. The expected time complexity of these algorithms is $O(n^n/2^n)$ rather than polynomial. Even graphs, however sparse, are still too common for these algorithms. A simple solution that avoids the above problem is provided by a probabilistic approach. If the claimed complexity of the algorithm is, say n^k, for some constant k, the algorithm should count the number of operations executed in the searching step. As soon as this number exceeds, say, n^{2k} the algorithm returns to the mapping step and, by changing some parameters, generates a new dependency graph. Notice that the trouble caused by even graphs was very unlikely to be detected by experimental observations. Fox *et al.* provide experimental data for $m \geq 32$. For 32 keys they would require close to a quarter of a million experiments to have a 50% chance of observing the phenomenon.

5 Yet another algorithm

In this section we propose a new algorithm for generating minimal perfect hash functions. The algorithm is presented as an open problem, as some details need to be worked out.

Consider the following algorithm. Given a graph G assign m numbers between 0 and $m-1$ in any order to the edges of G. In the next step construct a cycle basis of G so that each cycle has a unique edge (that type of a cycle basis is often called a fundamental cycle set). Find a cycle in the basis for which the alternating sum does not meet the conditions specified in this section. Find the minimum value $v > m$ such that if it is assigned to the unique edge of the cycle the conditions on the alternating sum are met. We require that such a minimum must not be equal to any numbers assigned to other edges. Repeat the whole process until no such cycle can be found. Solve the assignment problem for the graph with the values computed by the above method. Question: what is the maximum value on any edge of G? If, for a graph with m edges and $n = m/\log m$ vertices, all numbers were expected to be bounded by m, then the above specified algorithm could be used to generate a perfect hash function that uses $O(m)$ bits. The time required to complete such a process is proportional to the total length of cycles in a cycle basis.

References

[CHM92] Z.J. Czech, G. Havas, and B.S. Majewski. An optimal algorithm for generating minimal perfect hash functions. *Information Processing Letters*, 43(5):257–264, October 1992.

[CM92] Z.J. Czech and B.S. Majewski. Generating a minimal perfect hashing function in $O(m^2)$ time. *Archiwum Informatyki Teoretycznej i Stosowanej*, 4:3–20, 1992.

[FCDH91] E. Fox, Q.F. Chen, A. Daoud, and L. Heath. Order preserving minimal perfect hash functions and information retrieval. *ACM Transactions on Information Systems*, 9(3):281–308, July 1991.

[FCHD88] E. Fox, Q.F. Chen, L. Heath, and S. Datta. A more cost effective algorithm for finding perfect hash functions. Technical Report TR-88-30, Virginia Polytechnic Institute and State University, Blacksburg, VA, September 1988.

[FHC89] E. Fox, L. Heath, and Q.F. Chen. An $O(n \log n)$ algorithm for finding minimal perfect hash functions. Technical Report TR-89-10, Virginia Polytechnic Institute and State University, Blacksburg, VA, April 1989.

[FHCD92] E.A. Fox, L.S. Heath, Q. Chen, and A.M. Daoud. Practical minimal perfect hash functions for large databases. *Communications of the ACM*, 35(1):105–121, January 1992.

[FKS84] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, July 1984.

[GBY91] G.H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley, Reading, Mass., 1991.

[HM92] G. Havas and B.S. Majewski. Optimal algorithms for minimal perfect hashing. Technical Report 234, The University of Queensland, Key Centre for Software Technology, Queensland, July 1992.

[LC88] T.G. Lewis and C.R. Cook. Hashing for dynamic and static internal tables. *Computer*, 21:45–56, 1988.

[MWCH92] B.S. Majewski, N.C. Wormald, Z.J. Czech, and G. Havas. A family of generators of minimal perfect hash functions. Technical Report 16, DIMACS, Rutgers University, New Jersey, USA, April 1992.

[Rea62] R.C. Read. Euler graphs on labelled nodes. *Canadian Journal of Mathematics*, 14:482–486, 1962.

[Sag85] T.J. Sager. A polynomial time generator for minimal perfect hash functions. *Communications of the ACM*, 28(5):523–532, May 1985.