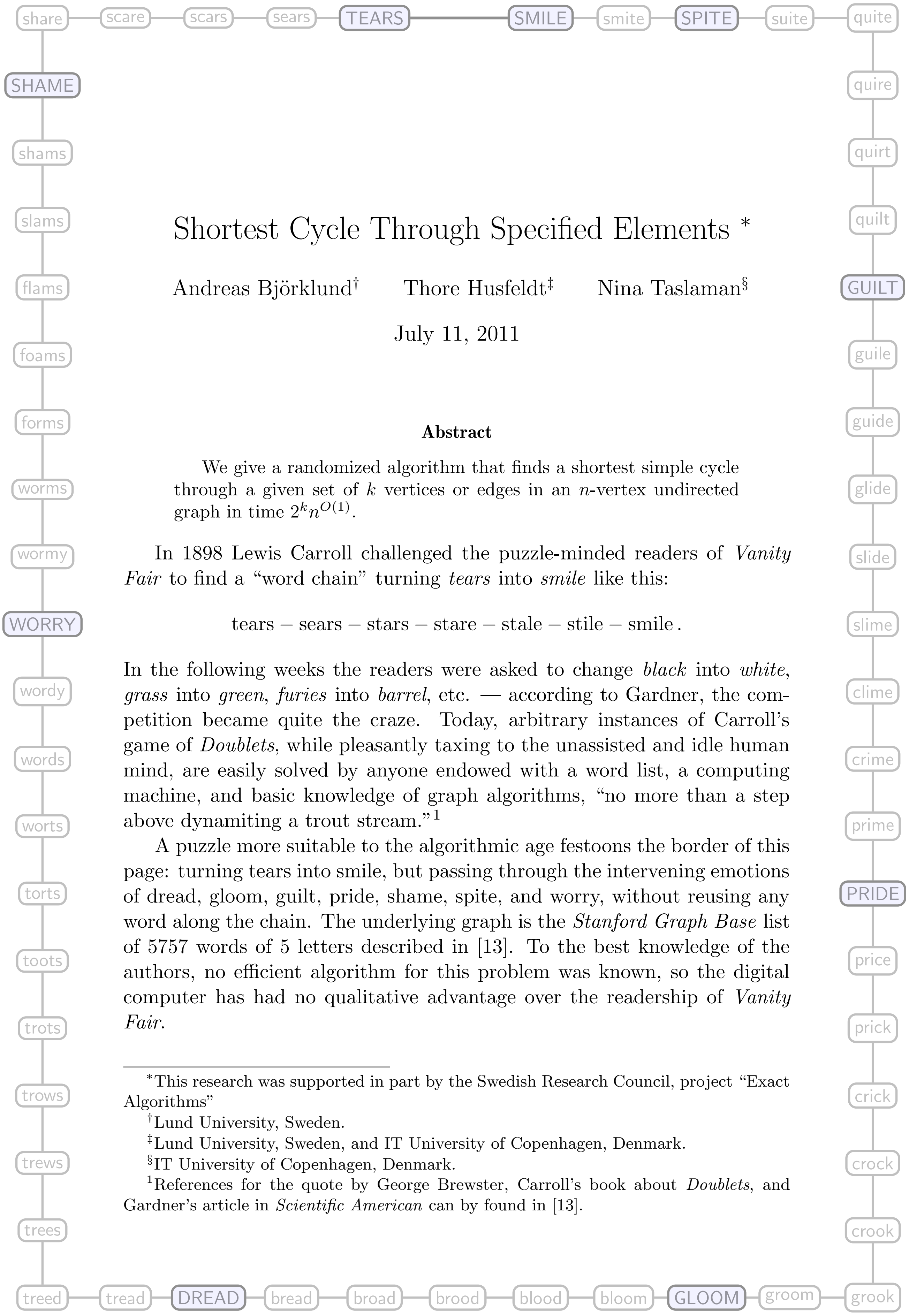


Shortest Cycle Through Specified Elements *

Andreas Björklund[†] Thore Husfeldt[‡] Nina Taslaman[§]

July 11, 2011

Abstract

We give a randomized algorithm that finds a shortest simple cycle through a given set of k vertices or edges in an n-vertex undirected graph in time $2^k n^{O(1)}$.

In 1898 Lewis Carroll challenged the puzzle-minded readers of *Vanity Fair* to find a "word chain" turning *tears* into *smile* like this:

$$\text{tears} - \text{sears} - \text{stars} - \text{stare} - \text{stale} - \text{stile} - \text{smile}\,.$$

In the following weeks the readers were asked to change *black* into *white*, *grass* into *green*, *furies* into *barrel*, etc. — according to Gardner, the competition became quite the craze. Today, arbitrary instances of Carroll's game of *Doublets*, while pleasantly taxing to the unassisted and idle human mind, are easily solved by anyone endowed with a word list, a computing machine, and basic knowledge of graph algorithms, "no more than a step above dynamiting a trout stream."[1]

A puzzle more suitable to the algorithmic age festoons the border of this page: turning tears into smile, but passing through the intervening emotions of dread, gloom, guilt, pride, shame, spite, and worry, without reusing any word along the chain. The underlying graph is the *Stanford Graph Base* list of 5757 words of 5 letters described in [13]. To the best knowledge of the authors, no efficient algorithm for this problem was known, so the digital computer has had no qualitative advantage over the readership of *Vanity Fair*.

*This research was supported in part by the Swedish Research Council, project "Exact Algorithms"

[†]Lund University, Sweden.

[‡]Lund University, Sweden, and IT University of Copenhagen, Denmark.

[§]IT University of Copenhagen, Denmark.

[1]References for the quote by George Brewster, Carroll's book about *Doublets*, and Gardner's article in *Scientific American* can by found in [13].

Behind the whimsical brain teaser lies of course a clean combinatorial problem: For a graph $G = (V, E)$ and a set $K \subseteq V \cup E$ of specified vertices or edges, a *K-cycle* is a simple cycle in G that includes all elements of K. The existence of such cycles through given elements has been a central topic of graph theory since the 1960s, see [11] for some references.

We present a randomized algorithm for this problem:

Theorem 1. *A shortest simple cycle through k given vertices or edges in an undirected n-vertex graph can be found by a randomized algorithm in time $2^k n^{O(1)}$ with one-sided errors of exponentially small probability in n.*

In particular, we can detect if a K-cycle exists at all. For this decision problem, the previous best time dependency on k was doubly exponential in k^{10}. For the optimization problem of determining the length of the shortest K-cycle, no efficient algorithm was known for $k > 3$, to the best of our knowledge. (In the language of parameterized complexity, the problem was not known to be fixed-parameter tractable.)

We stress that our algorithm is an improvement not only in the theoretical sense. Previous algorithms would outperform the brute force solution only for inputs of galactic size. A straightforward implementation of our algorithm is able to find cycles through several specified elements in a graph with thousands of vertices. Also, our algorithm is short and conceptually simple, using nothing more complicated than dynamic programming. Our correctness argument is a bit more subtle, but except for the DeMillo–Lipton–Schwartz–Zippel lemma, the presentation is self-contained.

The error in our randomized algorithm is one-sided in the sense that it never reports a false positive. The error probability is made exponentially small by repeating the algorithm a polynomial number of times.

Related work

Cycle through given vertices. For $k = 1$, the problem is solved by breadth first search, and for $k = 2$ it corresponds to finding a flow of size 2 between two vertices. For $k = 3$, it can be solved in linear time [8, 18]. It follows from the work of Robertson and Seymour on the disjoint path problem [19] that the problem can be solved in polynomial time for constant k. Kawarabayashi [11] finds a cycle through k specified vertices in polynomial time provided $k = O((\log \log n)^{1/10})$. This remains the best known deterministic algorithm. The arguments needed to establish the correctness of previous algorithms for $k \geq 3$ are not easy. The $k = 3$ algorithm by [18] requires extensive case analysis partially omitted from the journal version and appearing only in [17]. The other algorithms rely on combinatorial results, in the extreme case of Robertson and Seymour's algorithm [19] the correctness proof requires hundreds of pages.

For the optimization version of finding a *shortest* K-cycle, little was known. Dean's list of open questions from a 1991 conference on graph minors [4] asks if the problems can be solved in polynomial time for fixed k; in modern terms, if the problem is fixed parameter tractable. For $k = 2$, the problem is a special case of minimum-cost network flow and solvable by textbook algorithms; see [20] for some early results. According to [4], the case $k = 3$ is solved by Fleischner and Woeginger in [8], though this is not made explicit. An algorithm for fixed $k > 3$ does not follow from Robertson–Seymour techniques, and to the best of our knowledge, the question has remained open.

For directed graphs, the case $k = 1$ are solved as for undirected graphs, but already the detection problem for $k = 2$ is NP-hard [7].

Related problems. Kawarabayashi, Li, and Reed [12] give an algorithm for detecting a K-cycle whose length has a given parity; for fixed k, the running is polynomial in n, but the dependency on k is unspecified. Kobayashi and Kawarabayashi [14] give an algorithm for detecting if an *induced* K-cycle in a planar graph in time $O(\text{poly}(w^w)n^2)$, where $w = k^{2/3}$, in particular their algorithm runs in polynomial time for $k = o((\log n/\log\log n)^{2/3})$.

Long paths. The brute-force way to find a K-cycle of length l is of course to consider all $\binom{n}{l}$ candidate vertex subsets in G and see if they describe a K-cycle. Algorithms for long paths whose running time is exponential in the path length are known, and it is easy to change these algorithm to consider only such paths that visit K. For example, the algorithm in [3] can be modified to detect a K-cycle of length l in time $1.66^l n^{O(1)}$. While this may be competitive with previous algorithms for K-cycle for $k > 3$, it would be time-consuming to find the solution on the title page, which has $l = 58$. Moreover, it seems difficult to modify these algorithms to be able to detect the *absence* of a K-cycle in time subexponential in n.

Techniques. Our algorithm associates a polynomial over a finite field with the structure we want to find, an idea whose algorithmic importance was recognized with Edmonds's method for detecting a matching [6]. The present paper uses an idea involving cancellation of monomials, which was introduced by Koutis [15] to find long paths in graphs, with other papers exploring the same idea [2, 3, 16, 22]. Compared to these recent papers, our construction is quite simple, but the analysis is delicate.

One can view the determinant summation technique of [2] as detecting a cycle through specified vertices. The running time is exponential in the number of vertices *between* the specified ones. In the present paper the situation is reversed: the specified vertices are exponentially expensive, and the vertices in-between are cheap.

1 Algorithm

An easy observation is that we can restrict our attention to the case where K contains only vertices. Indeed, a specified edge uv can be replaced by adding a fresh vertex w to K and V and adding the edges uw and wv to E. This increases n by at most k.

Terminology. We let $N(v)$ denote the set of neighbours of v. A *walk* of length l is a sequence of vertices $v_0, v_1, \ldots, v_l$ with $v_i v_{i+1} \in E$ for $0 \leq i < l$. (Our graphs are simple, so a walk is uniquely defined by its sequence of vertices.) For a walk W we let $V(W)$ and $E(W)$ denote its set of vertices and edges, respectively. The walk's *internal vertices* are $v_1, \ldots, v_{l-1}$. A walk is *closed* if $v_0 = v_l$.

For vertex subset S, an S*-walk* is a walk that includes every vertex from S exactly once. A *digon* is a walk of the form u, v, u. A K*-digon* is a digon u, v, u where $v \in K$.

Closed walks Fix an arbitrary 'starting' vertex $a \in K$ and an arbitrary total order $\prec$ of the vertices in the neighbourhood $N(a)$. For given length l ($2 \leq l \leq n$), define the set $\mathcal{C}_l$ of closed walks $W = (v_0, v_1, \ldots, v_{l-1}, v_0)$ with the following properties:

P1 (start) $v_0 = a$,

P2 (K-walk) every vertex in K appears exactly once on W,

P3 (oriented) $v_1 \prec v_{l-1}$

P4 (no K-digons) every internal vertex on W that belongs to K has different predecessors and successors on W; formally, if $v_i \in K$ with $1 \leq i < l$ then $v_{i-1} \neq v_{i+1}$. See fig. 1(iii).

This set includes the K-cycles of length l, but can contain other, self-intersecting closed walks as well. Property P3 is used to ensure that a cycle and its reversal are considered only once; we arbitrarily decide to consider the cycle's direction that goes from a to the lower-ordered neighbour. Property P4 is more technical.

A function on sets of walks. With foresight, we will work in $\mathbf{F}_q$ with $q = 2^{1+\lceil \log_2 l \rceil}$, a finite field of characteristic 2 and size $q \geq 2l$. Associate values $f(e_1), \ldots, f(e_m) \in \mathbf{F}_q$ with the edges of G. Extend the definition of f to walks by

$$f(v_0, v_1, v_2, \ldots, v_l) = f(v_0 v_1) f(v_1 v_2) \cdots f(v_{l-1} v_l)\,.$$

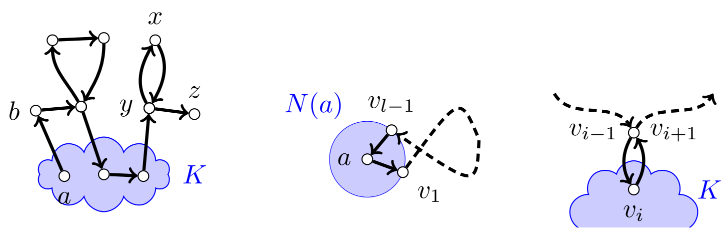


Figure 1: (i) A K-walk. (ii) A closed walk $a = v_0, v_1, \ldots, v_l = a$. (iii) A K-digon. All K-digons are forbidden, while the digon y, x, y in (i) is allowed.

and to sets $\mathcal{W}$ of walks by

$$f(\mathcal{W}) = \sum_{W \in \mathcal{W}} f(W)\,.$$

Algorithm. We are ready to present our main algorithm. Essentially, we define f by choosing $f(e_1), \ldots, f(e_m)$ at random and then check if $f(\mathcal{C}_l)$ is nonzero for increasing l.

Algorithm M (*Find the length of a shortest K-cycle.*). The input is an undirected, simple graph $G = (V, E)$ and a vertex subset $K \subseteq V$.

M1. [Initialize.] Choose $f(e) \in \mathbf{F}_q$ for all $e \in E$ uniformly at random. Choose a starting vertex $a \in K$ and an ordering of $N(a)$ arbitrarily. Set $l = |K|$

M2. [Iterate over all lengths.] Compute $f(\mathcal{C}_l)$ using dynamic programming (algorithm F in the next section). If $f(\mathcal{C}_l) \neq 0$ answer that G contains a K-cycle of length l. Otherwise increase l and repeat step M2 until $l = |V| + 1$.

M3. [Admit defeat.] Answer that no K-cycle was found.

This algorithm establishes theorem 1. The proof of correctness is in sec. 2.

Dynamic programming for sequencing problems. The values $f(\mathcal{C}_l)$ in step M2 can be computed using dynamic programming over the subsets of K and the length of the walk's prefix; this is a standard application of dynamic programming to sequencing problems [1, 9]. We only need to maintain some extra information about the last two vertices on a walk's prefix (in order to avoid building an K-digon) and the second vertex (in order to determine the orientation of the final closed walk).

For completeness, we present the dynamic programming construction in detail. For every vertex subset $S \subseteq K$, vertices $b, y, z \in V$, and length r ($2 \le r \le n$), define the values

$$T(S, r, b, y, z) = \sum_{W \in \mathcal{W}_r} f(W)\,,$$

where the sum is taken over all walks $W = v_0, \ldots, v_r$ with the properties

S1 (start and end) $v_0 = a$, $v_1 = b$, $v_{r-1} = y$, and $v_r = z$,

S2 (S-walk) every vertex in S appears exactly once on W,

S3 (no S-digons) if $v_i \in S$ with $1 \le i < r$ then $v_{i-1} \ne v_{i+1}$.

The values $T(S, r, b, y, z)$ can be computed for all arguments by dynamic programming in time $O(2^k n^5)$; the details are given below.

Algorithm F (*Compute* $f(\mathcal{C}_l)$). The input is an undirected, simple graph $G = (V, E)$, a vertex subset $K \subseteq V$ with $a \in K$ a fixed start vertex, an integer l ($2 \le l \ne n$), and values $f(e)$ for each $e \in E$.

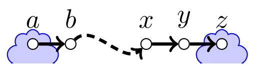
F1. [Initialize table.] Set all table entries to 0. Set $T(\{a\}, 2, b, a, b) = f(ab)$ for all $b \in N(a)$. Set $r = 3$.

F2. [Dynamic programming.] Update $T(S, r, b, y, z)$ as follows, for every $b, y, z \in V$ with $ab \in E$ and $yz \in E$ and every $S \subseteq K$. (These are the S-walks of length r of the form $a, b, \ldots, y, z$.)

If $z \in K$ then for each $S \ni z$ set

$$T(S, r, b, y, z) = f(yz) \sum_{x \in V} T(S - \{z\}, r-1, b, x, y)\,.$$

(The situation looks somewhat like this:



The prefix $a, b, \ldots, x, y$ must be an $(S - \{z\})$-walk because of property S2. Since $z \in S$, it has not appeared earlier on the walk, and in particular, $z \ne x$. Thus the new vertex z cannot violate property S3 even if $y \in S$.)

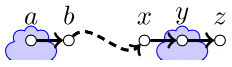
If $z \notin K$ then for each $y \notin S$ set

$$T(S, r, b, y, z) = f(yz) \sum_{x \in V} T(S, r-1, b, x, y)\,,$$

and for $y \in S$ set

$$T(S, r, b, y, z) = f(yz) \sum_{\substack{x \in V \\ x \neq z}} T(S, r-1, b, x, y)\,.$$

(The condition $x \neq z$ ensures that in a situation like



property S3 is satisfied.) All other $T(S, r, b, y, z)$ remain at 0. Increment r and repeat F3 until $r = l$.

F3. [Add relevant contributions.] Return

$$f(\mathcal{C}_l) = \sum_{y \in V} \sum_{b \in N(a)} \sum_{\substack{y \in N(a) \\ b \prec y}} T(K, l, b, y, a)\,. \tag{1}$$

Implementation details. The dynamic programming solution above is presented without attention to efficiency. Thus, the polynomial factor is prohibitive. The program can be sped up considerably, for example by iterating over $x \in N(y)$ instead of $x \in V$, or by treating outgoing and incoming edges around $N(a)$ differently to break symmetry. Our implementation runs in $O(2^k n^2 l)$.

For *finding* a cycle, rather than merely reporting its existence, we search through $v \in N(a)$ with binary search to detect a simple K-walk from v to a of length l, then through $N(v)$ for a simple K-walk of length $l-1$, etc. This increases the running time by a factor $l \log n$.

The space requirement of algorithm F is exponential in k. Using inclusion-exclusion in the style of [10] instead of dynamic programming, the space requirement can be reduced to polynomial in n and k.

2 Correctness

To see that algorithm M is correct, it is useful to view the value of $f(\mathcal{C}_l)$ as a function in the m choices of values $f(e_1), \ldots, f(e_m)$. To be precise, we will consider the polynomial $p_l \in \mathbf{F}_q[x_1, \ldots, x_m]$ defined for a given graph $G = (V, E)$ and $K \subseteq V$ by

$$p_l(x_1, \ldots, x_m) = \sum_{W \in \mathcal{C}_l} \prod_{e_i \in E(W)} x_i\,. \tag{2}$$

Then $f(\mathcal{C}_l) = p_l(f(e_1), \ldots, f(e_m))$. From the definition, it is clear that p_l is a polynomial in m variables of total degree l.

The following result implies correctness of algorithm M.

Lemma 1. *Let $G = (V, E)$ be an undirected, simple graph with $K \subseteq V$, and let $p_l \in \mathbf{F}_q[x_1, \ldots, x_m]$ be defined as in (2). If G has no K-cycle of length at most $l - 1$, then it has one of length l if and only if $p_l \neq 0$.*

We break this lemma into lemma 3 and lemma 4 below.

Since algorithm M chooses the values $f(e_1), \ldots, f(e_m)$ at random, we can view its behaviour as evaluating $p_l(x_1, \ldots, x_m)$ at a random point in $\mathbf{F}_q^m$. If $p_l = 0$, then algorithm M never reports a nonzero value. Conversely, the probability of a false negative, i.e., reporting 0 when $p_l \neq 0$, is bounded by the following lemma.

Lemma 2 (DeMillo–Lipton–Schwartz–Zippel [5, 21])**.** *Let $p \in \mathbf{F}_q[x_1, \ldots, x_m]$ be nonzero of total degree at most d. Then, for $f_1, f_2, \ldots, f_m \in \mathbf{F}_q$ selected independently and uniformly at random,*

$$\Pr[\, p(f_1, f_2, \ldots, f_m) \neq 0 \,] \geq 1 - \frac{d}{q}.$$

It remains to establish lemma 1.

Lemma 3. *If G has a shortest K-cycle of length l, then p_l is nonzero.*

Proof. A K-cycle $C \in \mathcal{C}_l$ contributes the term

$$\prod_{e_i \in E(C)} x_i$$

to p_l. This term depends only on the set of edges on C. With properties P3 and P1, the simple cycle C can be recovered from $E(C)$, so the contribution is unique. □

We now argue that all walks in $\mathcal{C}_l$ must pair up and cancel whenever G has no K-cycle of length at most l. To show this, we define a *fixed-point-free involution* on $\mathcal{C}_l$, i.e., a mapping $\phi : \mathcal{C}_l \to \mathcal{C}_l$ such that $\phi(\phi(W)) = W$ and $\phi(W) \neq W$ for all $W \in \mathcal{C}_l$. Such a mapping partitions $\mathcal{C}_l$ into pairs of walks $\{W, \phi(W)\}$.

Lemma 4. *If G has no K-cycle of length at most l, then there is a fixed-point-free involution $\phi : \mathcal{C}_l \to \mathcal{C}_l$ such that $f(W) = f(\phi(W))$ for all $W \in \mathcal{C}_l$.*

The basic idea of the proof is to define ϕ like so: Every walk in $\mathcal{C}_l$ that is not a K-cycle must contain a repeated internal vertex. For example, the closed walk 123567541 contains the repeated internal vertex 5. We now want to map this walk to the walk $123\overleftarrow{5675}41 = 123576541$ obtained from reversing the cycle between the first and last occurrence of 5. The resulting closed walk is different, yet corresponds to the same monomial since it contains the same edges. For this idea to work for general closed walks, we need to be careful about internal palindromes ($123\overleftarrow{56765}41$) and how to choose the internal vertex.

Proof of lemma 4. A *subwalk* of a walk $W = v_0, v_1, \ldots, v_l$ is a walk of the form $v_i, v_{i+1}, \ldots, v_j$ for some i, j with $0 \leq i \leq j \leq l$. It is a *prefix* of W if $i = 0$, and a *suffix* of W if $j = l$. If the end vertex of W and the start vertex of another walk W' are neighbours, we write WW' for the concatenated walk. We let $\overleftarrow{W}$ denote the reversal of W, and say that W is a *palindrome* if $W = \overleftarrow{W}$. A palindrome is *nontrivial* if it contains more than one vertex.

A repeated internal vertex on a walk is *critical.* If u is critical in W, we let $[uWu]$ denote the subwalk in W of maximal length starting- and ending at u. We can then decompose W as

$$W = X[uWu]Z\,, \tag{3}$$

for some prefix X and suffix Z of W such that neither contain u. By *contraction* of $[uWu]$ in W we refer to the operation $X[uWu]Z \mapsto XuZ$.

Let G be a graph, having no K-cycle of length at most l. We define the mapping $\phi : \mathcal{C}_l \to \mathcal{C}_l$ as follows. Let v be the output from the following procedure:

```
Algorithm R (Find v.). The input is a walk $W \in \mathcal{C}_l$.
R1. Let $i = 0$ and $W_0 = W$.
R2. Let $v$ be the first critical vertex in $W_i$.
R3. Decompose $W_i$ as $X[vW_iv]Z$. If $[vW_iv]$ is palindromic, set
      $W_{i+1} = XvZ$, increment $i$, and go to R2.
R4. Return $v$. ($[vW_iv]$ is not a palindrome.)
```

Decompose W as $X[vWv]Z$ for this v and define ϕ as

$$\phi : X[vWv]Z \mapsto X\overleftarrow{[vWv]}Z\,.$$

For example, on input 12345432345467861, algorithm R returns $v = 6$:

$$W_0 = 1[2345432]345467861\,, \quad W_1 = 123[454]67861\,, \quad W_2 = 1234[6786]1\,, \tag{4}$$

so $\phi(12345432345467861) = 12345432345\overleftarrow{467861} = 12345432345468761$.

To see that ϕ is well-defined, we need to show show that algorithm R gives a critical vertex v on input W. We first show that the following invariant holds for all i:

I1. $W_i \in \mathcal{C}_m$ for some $m \leq l$.

Assume $W_{i-1} \in \mathcal{C}_{m'}$ for some $m' \leq l$. Note that W_i is obtained from W_{i-1} by contracting a nontrivial palindromic subwalk, which maps closed walks to shorter closed walks. Since W_{i-1} satisfies P2, no vertex from K appears twice on W_{i-1}. In particular, no vertex from K is critical in W_{i-1}, and by property P4 also cannot occur in a nontrivial palindrome. Thus, W_i must contain any vertex in K present in W_{i-1}, so W_i satisfies P2, and also P1, since $a \in K$. Note that if the jth node v_j on W_{i-1} is removed in W_i, then v_{j-1} and v_{j+1} are either critical in W_{i-1}, or part of a nontrivial palindrome. This means that neighbors on W_{i-1} of any vertex in K are preserved in W_i, so that W_i also satisfies P4. Finally, W_i satisfies P3, since the given orientation is never changed. We conclude that $W_i \in \mathcal{C}_m$ for some $m \leq m' \leq l$.

It follows that algorithm R must terminate; otherwise, W_i would eventually have no critical vertex, and by *I1* be a K-cycle in G of length $m \leq l$. Also, the output vertex v must be critical in W, since $V(W_i) \subseteq V(W)$.

To see that ϕ is an involution on $\mathcal{C}_l$, write

$$W = X[vWv]Z \text{ and } \phi(W) = X\overleftarrow{[vWv]}Z\,.$$

We have to show that algorithm R outputs v on input $\phi(W)$. This follows because W and $\phi(W)$ have the same prefix X, so algorithm R will perform the same contractions until it reaches v. It then terminates, returning v, since $\overleftarrow{[vWv]}$ is not a palindrome if $[vWv]$ is not a palindrome. Thus

$$\phi(\phi(W)) = X\overleftarrow{\overleftarrow{[vWv]}}Z = W\,.$$

To see that ϕ is fixed point-free, it suffices to show that $[vWv]$ is not a palindrome. The following invariant provides proof by contrapositive, since we know that if algorithm R outputs v, then at some step $[vW_iv]$ is not a palindrome.

I2. If u is a critical vertex in W_i such that $[uW_iu]$ is a palindrome, then $[uW_{i+1}u]$ will also be a palindrome.

We verify *I2* by considering cases. If $[vW_iv]$ is not a palindrome, or contains no copy of u, then step R3 of the algorithm leaves $[uW_iu]$ unaffected, so $[uW_{i+1}u] = [uW_iu]$, a palindrome. If u is the first critical vertex in W_i (i.e., $u = v$) then $[uW_{i+1}u] = u$, a palindrome. Otherwise we can write W_i in two ways using (3),

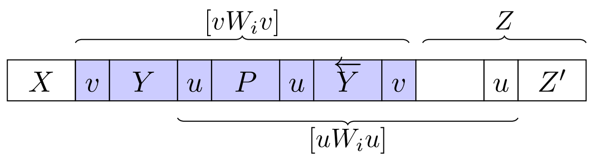
$$W_i = X[vW_iv]Z = X'[uW_iu]Z'\,, \qquad (u \neq v)\,,$$

where $[vW_iv]$ is a palindrome containing u. Thus, for some (possibly empty) subwalk Y not containing u, and some palindrome P

$$[vW_iv] = \begin{cases} vYu\overleftarrow{Y}v\,, & \text{if } u \text{ appears only once on } [vW_iv]\,; \\ vYuPu\overleftarrow{Y}v\,, & \text{otherwise}\,. \end{cases}$$

We can handle both cases at once with the convention that $u = uPu$ for empty P. Since v is the first critical vertex in W_i, the critical vertex u

does not appear in the prefix X. If u also does not appear in the suffix Z, then $W_{i+1} = XvZ$ contains no copy of u, so $[uW_{i+1}u]$ is the empty walk, pathologically a palindrome. The final, and interesting, case is when u appears on the suffix Z. (An example is $u = 4$ in (4).) Pictorially,



Since uW_iu is a palindrome, and the suffix Z does not contain v, it must be that $Z = YuPuZ'$. Thus, after the contraction in step R3, we have $W_{i+1} = XvZ = XvYuPuZ'$. Since neither X, Y or Z' contain u, this gives $[uW_{i+1}u] = uPu$, a palindrome.

We have established that ϕ is a fixed-point-free involution on $\mathcal{C}_l$. Finally, $f(W) = f(\phi(W))$, since $\phi(W)$ is just a permutation of W. $\square$

References

[1] R. Bellman, Dynamic programming treatment of the travelling salesman problem. *J. Assoc. Comput. Mach.* **9** 61–63 (1962).

[2] A. Björklund, Determinant sums for undirected Hamiltonicity. 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010 (October 23–26, 2010, Las Vegas, Nevada, USA). IEEE Computer Society 2010, pp. 173–182

[3] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto, Narrow sieves for parameterized paths and packings. Manuscript, arXiv:1007.1161, 2010.

[4] N. Dean, Open problems. In Graph Structure Theory, Proceedings of a AMS-IMS-SIAM Joint Summer Research Conference on Graph Minors (June 22 to July 5, Washington, Seattle, 1991). Contemporary Mathematics 147, American Mathematical Society 1993, pp. 677–688.

[5] R. A. DeMillo and R. J. Lipton, A probabilistic remark on algebraic program testing. *Inform. Process Lett.* **7** 193–195, 1978.

[6] J. Edmonds, Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards Sect. B* **71B** 241–245, 1967.

[7] S. Fortune, J. Hopcroft, and J. Wyllie, The directed subgraph homeomorphism problem. *Theor. Comput. Sci.* **10** 111-121 (1980)

[8] H. Fleischner and G. J. Woeginger, Detecting cycles through three fixed vertices in a graph. *Inf. Proc. Lett.* **41** 29–33 (1992).

[9] M. Held, R.M. Karp, A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.* **10** 196–210 (1962).

[10] R. M. Karp, Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.* **1** 49–51 (1982).

[11] K. Kawarabayashi, An improved algorithm for finding cycles through elements. In: Integer Programming and Combinatorial Optimization, 13th International Conference, IPCO 2008 (Bertinoro, Italy, May 26–28, 2008), Proceedings. Lecture Notes in Computer Science 5035, Springer 2008, pp. 374–384.

[12] K. Kawarabayashi, Z. Li, and B. A. Reed, Recognizing a totally odd K_4-subdivision, parity 2-disjoint rooted paths and a parity cycle through specified elements. Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010 (Austin, Texas, USA, January 17-19, 2010). SIAM 2010, pp. 318–328.

[13] D. E. Knuth, Combinatorial Algorithms, Part 1. *The Art of Computer Programming*, vol. 4A. Addison-Wesley, 2011.

[14] Y. Kobayashi and K. Kawarabayashi, Algorithms for finding an induced cycle in planar graphs and bounded genus graphs. Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009 (New York, NY, USA, January 4-6, 2009). SIAM 2009, pp. 1146–1155.

[15] I. Koutis, Faster algebraic algorithms for path and packing problems. In *Proc. 35th International Colloquium on Automata, Languages and Programming, ICALP* (Reykjavik, Iceland, July 7–11, 2008), Springer LNCS 5125, pages 575–586, 2008.

[16] I. Koutis and R. Williams, Limits and applications of group algebras for parameterized problems. In *Proc. 36th International Colloquium on Automata, Languages and Programming, ICALP* (Rhodes, Greece, July 5–12, 2009), Springer LNCS 5555, pages 653–664, 2009.

[17] A.S. LaPaugh, The subgraph isomorphism problem. Massachusetts Institute of Technology, Laboratory for Computer Science TM 99, 1978.

[18] A.S. LaPaugh and R.L. Rivest, The subgraph homomorphism problem. *J. Comput. Sys. Sci.* **20** 133–149 (1980).

[19] N. Robertson and P. D. Seymour, Graph minors XIII. The disjoint paths problem. *J. Combin. Theory Ser. B* **63** 65–110 (1995).

[20] J. W. Suurballe and R. E. Tarjan, A quick method for finding shortest pairs of disjoint paths. *Networks* **14** 325–336 (1984).

[21] J. T. Schwartz, Fast probabilistic algorithms for verification of polynomial identities. *J. Assoc. Comput. Mach.* **27** 701–717, 1980.

[22] R. Williams, Finding paths of length k in $O^*(2^k)$. *Inform. Process Lett.* **109** 301–338, 2009.