Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 61 (2001)

ECCC

The Complexity of Counting Self-Avoiding Walks in Two-Dimensional Grid Graphs and in Hypercube Graphs

Maciej Liśkiewicz[1]
Universität zu Lübeck
Institut für Theoretische Informatik

Mitsunori Ogihara[2]
Department of Computer Science
University of Rochester

Seinosuke Toda[3]
Department of Applied Mathematics
Nihon University

October 16, 2001

[1]Universität zu Lübeck, Institut für Theoretische Informatik, Wallstraße 40, D-23560, Lübeck, Germany. email: liskiewi@tcs.mu-luebeck.de. On leave from Instytut Informatyki, Uniwersytet Wrocławski, Poland.

[2]Department of Computer Science, University of Rochester, Rochester, NY 14627-0226, USA. Email: ogihara@cs.rochester.edu. Supported in part by NSF grants CCR-9701911, CCR-9725021, DUE-9980943, INT-9726724, an NIH/NIA grant RO1-AG18231 and a DARPA grant F30602-98-2-013.

[3]Department of Applied Mathematics, Nihon University, 3-25-40 Sakurajyou-shi, Setagaya-ku, Tokyo 156, Japan. Email: toda@am.chs.nihon-u.ac.jp.

ISSN 1433-8092

Abstract

Valiant (*SIAM Journal on Computing* 8, pages 410–421) showed that the problem of computing the number of simple s-t paths in graphs is #P-complete both in the case of directed graphs and in the case of undirected graphs. Valiant then asked whether the self-avoiding walk problem on the two-dimensional grid, the problem of computing the number of self-avoiding walks of a given length in the two-dimensional grid is complete for $\#P_1$, the tally-version of #P. This paper offers a partial answer to the question of Valiant. It is shown that computing the number of self-avoiding walks of a given length in the two-dimensional grid graph is #P-complete. The paper also studies several variations of the prolem and shows that all of them are #P-complete.

This paper also studies the problem of computing the number of self-avoiding walks in graphs embedded in a hypercube. Similar completeness results are shown for hypercube graphs. By scaling the computation time to exponential, it is shown that computing the number fo self-avoiding walks in the hypercubes is complete for #EXP in the case when a hypercube graph is specified by its dimension and a boolean circuit that accepts the nodes.

Finally, this paper studies the complexity of testing whether a given word over the four letter alphabet $\{U, D, L, R\}$ is a self-avoiding walk. A linear-space lower bound is shown for nondeterministic Turing machines with a one-way input head to recognize self-avoiding walks.

1 Introduction

A self-avoiding walk (SAW, for short) is a random walk that does not intersect itself. Computing the exact number of self-avoiding walks of a given length n on the complete two-dimensional lattice is a well-known problem and has been extensively studied (see Madras and Slade [MS93] and Welch [Wel93]). Although the exact formulas have been found for some special cases of the problem (see, e.g. [Wil96]), no formulas for the general case are known. The exact number for n up to 51 has been calculated (see Conway and Guttman [CG96]). Also, Randall and Sinclair [RS93] present an algorithm for generating self-avoiding walks under uniform distribution.

Valiant [Val79b] is the first to find connections between self-avoiding walks and computational complexity theory. Valiant showed that the problem of counting the number of all simple s-t paths is #P-complete under polynomial parsimonious reductions (polynomial-time function reductions with no post-computation) both in the case of directed graphs and in the case of undirected graphs. He then asked whether the exact counting problem for the two-dimensional grid is complete for $\#\mathrm{P}_1$, the "tally" version of #P, if the length of walks is specified by a tally string.

While it is straightforward to prove that the problem belongs to $\#\mathrm{P}_1$, settling the question of whether the problem is $\#\mathrm{P}_1$-hard appears to be difficult. The difficulty seems to lie in the fact that the two-dimensional grid has a very rigid, regular structure. An approach for proving the hardness would be to embed the computation of a nondeterministic Turing machine in the grid with no holes, thereby creating a correspondence between the ID's of a Turing machine and the grid points. However, every line segment of the grid may be traversed in either direction and the graph that represents transitions between machine configurations is multidimensional. Thus, finding such an embedding seems hard to find.

So we question whether the counting problem is $\#\mathrm{P}_1$-hard if it is permitted to create holes, i.e., if the graphs in which SAW's are count are those composed of the nodes and the edges of the complete two-dimensional grid. Since there are exponentially many choices for the locations of the holes, given a finite two-dimensional grid, here we should be rather thinking about #P-hardness than $\#\mathrm{P}_1$-hardness. Thus, we ask:

> *Is counting SAW's of a specified length in two-dimensional grid graphs* #P*-complete under some polynomial-time function reductions?*

In this paper, we resolve this question positively. The problem of counting SAW's in two-dimensional grid graphs is #P-complete under polynomial-time function reductions where the post-computation that is performed to recover the value of a #P-function from the count is simply right-bit-shifting the count in its binary representation, i.e., truncating the binary representation at the right end.

The proof uses as a basis a result by Garey, Johnson, and Tarjan [GJT76] that the problem of testing whether a planar 3-regular graph has a Hamiltonian cycle is NP-complete. For an NP-complete problem its NP-completeness proof actually shows that the counting version of the problem (i.e., the problem of computing the number of witnesses) is #P-complete. It is not the case for the Garey–Johnson–Tarjan proof. We show that by using different gadgets the Garey–Johnson–Tarjan proof can be made to show that the problem of counting Hamiltonian cycles in a certain kind of planar 3-regular graphs is #P-complete under the aforementioned "right-bit-shift" reductions. We then transform the problem of counting Hamiltonian cycles in the special kind of planar 3-regular graphs to the problem of counting SAW's in two-dimensional grid graphs. In this transformation an edge of the planar graph is cut in the middle and then the edges are mapped on the two-dimensional grid so that they are stretched to paths of the same length. Then

the Hamiltonian cycles in the input graph can be counted by computing the number of longest paths in the grid graph. This establishes that the problem of counting SAW's of a given length in two-dimensional grid graphs is #P-complete regardless of whether the end points of SAW's are specified.

This observation raises immediately the question of whether similar flexibility holds if SAW's of any length is to be counted. We show indeed this is the case. Namely, the problem counting SAW's in two-dimensional grid graphs is #P-complete regardless of whether length is specified and regardless of whether the end of points of the SAW's are specified. This completeness result has an added bonus. In the "right-bit-shift" reductions from #P-function to the counting problem of SAW's in a hypercube graph, the dimension of the hypercube graphs increases logarithmically. We ask how complex the problem is if the dimension is allowed to grow polynomially. Since there are 2^n nodes in the n-dimensional hypercube, we assume that hypercube graphs are specified by boolean circuits (each point in a hypercube can be naturally viewed as a binary string). Furthermore, we stipulate that the hypercube graphs have a special property that for each pair of neighboring nodes in the hypercube it holds that if both of them belong to the hypercube graph then the edge joining them also belongs to the graph. Then we show that the problem of counting SAW's with this kind of specification is the exponential-time version of #P.

Finally, we study computational complexity of detecting whether a given string over a four-letter alphabet (the four letters in the alphabet mutually exclusively correspond to the four directions in traversals on the two-dimensional grid) encodes a self-avoiding walk in the complete two-dimensional grid. A motivating question to study this problem concerns lower space-bounds for counting SAW's of a given length in such grid. An interesting open question is e.g. whether this counting problem belongs to #L? In this paper we show that to detect SAW's on-line any nondeterministic Turing machine needs at least linear space.

This paper is organized as follows: In Section 2, we set down notation and notions that are used throughout the paper. In Section 3, we present our reduction from SAT to the the Hamiltonian circuit problem in 3-regular planar graphs. In Section 4, we study the complexity of computing the number of self-avoiding walks in the two-dimensional grid graphs. In Section 5, we study the complexity of computing the number of self-avoiding walks in the hypercube graphs. Finally, in Section 6, we prove our complexity lower-bound results.

2 Preliminaries

2.1 Counting Complexity Classes

Let M be a nondeterministic Turing machine. $\#acc_M$ denotes the function that maps each string x to the number of accepting computation paths of M on input x. We are interested in two classes of functions, a class #P of Valiant [Val79a], defined as $\{\#acc_M \mid M \text{ is a polynomial-time nondeterministic Turing machine}\}$, and a class #EXP of Papadimitriou and Yannakakis [PY86], defined as $\{\#acc_M \mid M \text{ is an exponential-time nondeterministic Turing machine}\}$. For a language class $\mathcal{C}$, Valiant [Val79a] proposes to define $\#\mathcal{C}$ as the class of all functions f such that there exist some polynomial-time nondeterministic oracle Turing machine M and some language $A \in \mathcal{C}$ such that $f = \#acc_{M^A}$. The class #EXP by Papadimitriou and Yannakakis, which we study in this paper, is strictly larger than the one defined by way of Valiant. This is because each function in the Valiant version of #EXP has polynomial output length while functions in the Papadimitriou–Yannakakis version may have exponential output length.

2.2 Reductions Between Counting Functions

Next we define polynomial-time reductions between counting functions. Let f and g be functions from Σ^* to $\mathbf{N}$. We say that f is *polynomial-time one-Turing reducible* to g, denoted by $f \leq^p_{1\text{-}T} g$, if there is a pair of polynomial-time computable functions, $R_1 : \Sigma^* \to \Sigma^*$ and $R_2 : \Sigma^* \times \mathbf{N} \to \mathbf{N}$, such that for all x, $f(x) = R_2(x, g(R_1(x)))$. We consider two special cases of polynomial-time one-Turing reductions. We say that f is *polynomial-time parsimoniously reducible* to g, denoted by $f \leq^p_{\text{parsimonious}} g$, if for all x and y the above R_2 satisfies $R_2(x, y) = y$, i.e., for all x, $f(x) = g(R_1(x))$. We say that the function f is *polynomial-time right-bit-shift reducible* to g, denoted by $f \leq^p_{r\text{-}shift} g$, if there is a polynomial-time computable function $R_3 : \Sigma^* \to \mathbf{N} - \{0\}$ such that for all x and y it holds that $R_2(x, y) = y \operatorname{div} 2^{R_3(x)}$, i.e., for all x, $f(x) = g(R_1(x)) \operatorname{div} 2^{R_3(x)}$, where div is integer division. It is easy to see that the following proposition holds.

Proposition 2.1 *Both $\leq^p_{r\text{-}shift}$-reductions and $\leq^p_{\text{parsimonious}}$-reductions are transitive.*

2.3 Counting Problems

Here we define the problems that we are concerned with. A 3CNF formula (respectively, a $\hat{3}$CNF) formula is a boolean formula in the conjunctive normal form such that each clause has exactly three (respectively, at most three) literals. 3SAT (respectively, $\hat{3}$SAT) is the problem of testing satisfiability of 3CNF (respectively, $\hat{3}$CNF) formulas. #SAT (respectively, #3SAT and #$\hat{3}$SAT) is the problem of computing the number of satisfying assignments of boolean formulas (respectively, 3CNF formulas and $\hat{3}$CNF formulas). By the standard reduction from nondeterministic Turing machine computations to 3CNF formulas (see, for example [Pap94]), these three problems are each complete for #P under polynomial-time parsimonious reductions.

Proposition 2.2 *[Val79b] #SAT, #3SAT, and #$\hat{3}$SAT are each complete for #P under polynomial-time parsimonious reductions.*

For a CNF formula φ and its satisfying assignment α, we say that α is a *not-all-equal satisfying assignment of φ* if for every clause C of φ with more than one literal it holds that α falsifies at least one literal of C. Not-All-Equal-SAT (NAE3SAT for short) [Sch78] is a a special case of 3SAT in which it is asked whether a given 3CNF formula can be satisfied by a not-all-equal satisfying assignment. Schaefer [Sch78] is the first to study this variation of the satisfiability problem. He also proves that the problem is NP-complete. Schaefer restricted that the input formulas are 3CNF formulas, but we allow them to be $\hat{3}$CNF formulas.

The following lemma, which will play a crucial role later, states that there is a polynomial-time parsimonious reduction from #SAT to #NAE3SAT such that each formula belonging to the range of the reduction is composed of three-literal clauses and a single one-literal clause and is satisfiable only by not-all-equal satisfying assignments.

Lemma 2.3 *There is a polynomial-time computable function f that maps each 3CNF formula to a $\hat{3}$CNF formula such that for all integers $n, m \geq 1$ and for all formula φ of n variables and m clauses, $\psi = f(\varphi)$ has the following properties:*

1. *The number of variables of ψ is $n + m + 1$ and the number of clauses of ψ is $8m + 1$, out of which $8m$ are three-literal clauses and one is a single-literal clause.*

2. *Every satisfying assignment of ψ is a not-all-equal satisfying assignment. More precisely, for every satisfying assignment α of ψ, α satisfies exactly two literals for exactly $4m$ three literal clauses and satisfies exactly one literal for exactly $4m$ three-literal clauses.*

3. $\#\mathrm{SAT}(\varphi) = \#\mathrm{SAT}(\psi)$.

Proof. Let $n, m \geq 1$ be integers. Let φ be a 3CNF formula of n variables and m clauses. Let $C_1, \ldots, C_m$ be the clauses of φ. We construct ψ, which will be the value of the reduction on input φ, as follows: First, we introduce one new variable w and a single-literal clause $(\overline{w})$. Due to this clause, every satisfying assignment of ψ sets the value of w to false. Next, for each i, $1 \leq i \leq m$, we construct from C_i an eight-clause formula C_i' as follows: Let x, y, z be the three literals of C_i. We introduce a new variable u_i and some clauses so that every satisfying assignment of ψ sets the value of u_i to the value of $x \vee y$. This requirement can be fulfilled by introducing three clauses:

$$(\overline{u_i} \vee x \vee y), (u_i \vee \overline{x}), (u_i \vee \overline{y}).$$

This turns C_i to

$$(\overline{u_i} \vee x \vee y) \wedge (u_i \vee \overline{x}) \wedge (u_i \vee \overline{y}) \wedge (u_i \vee z).$$

We insert the literal w to each of the last three clauses to make them three-literal clauses. Thus, we obtain

$$(\overline{u_i} \vee x \vee y) \wedge (u_i \vee \overline{x} \vee w) \wedge (u_i \vee \overline{y} \vee w) \wedge (u_i \vee z \vee w).$$

Then every satisfying assignment of this four-clause formula that also satisfies the single-literal clause $(\overline{w})$ is a not-all-equal satisfying assignment. With this current form, how many out of the four three-literals clauses in the above have exactly two satisfied literals by such a not-all-equal satisfying assignment is dependent on the satisfying assignment itself. So, we add the literal-wise complement of each of the four clauses:

$$(u_i \vee \overline{x} \vee \overline{y}) \wedge (\overline{u_i} \vee x \vee \overline{w}) \wedge (\overline{u_i} \vee y \vee \overline{w}) \wedge (\overline{u_i} \vee \overline{z} \vee \overline{w}).$$

Note for all three-literal clauses D, for all assignments α, and all integers k, $0 \leq k \leq 2$, it holds that α satisfies exactly k literals of D if and only if α satisfies exactly $3 - k$ literals of the literal-wise complement of D. This is C_i', i.e.,

$$\begin{aligned} C_i' = {} & (\overline{u_i} \vee x \vee y) \wedge (u_i \vee \overline{x} \vee w) \wedge (u_i \vee \overline{y} \vee w) \wedge (u_i \vee z \vee w) \\ & \wedge (u_i \vee \overline{x} \vee \overline{y}) \wedge (\overline{u_i} \vee x \vee \overline{w}) \wedge (\overline{u_i} \vee y \vee \overline{w}) \wedge (\overline{u_i} \vee \overline{z} \vee \overline{w}). \end{aligned}$$

Now every satisfying assignment α of C_i' that satisfies $(\overline{w})$ satisfies exactly two literals for four three-literal clauses and exactly one literal for four three-literal clauses.

The formula ψ is defined as

$$C_1' \wedge \ \ldots \ \wedge C_m' \wedge (\overline{w}).$$

Then ψ has $n + m + 1$ variables, consists of $8m$ three-literal clauses and one single-literal clause, and has as many satisfying assignment as φ. Furthermore, for every satisfying assignment α of ψ, there are precisely $4m$ three-literal clauses such that α satisfies exactly one literal and precisely $4m$ three-literal clauses such that α satisfies exactly two literals. Clearly, the number of satisfying assignments of ψ is equal to that of φ. This proves the lemma. □

Hamiltonian Path is the problem of deciding, given a graph G with two specified nodes s and t, whether there is a Hamiltonian path from s to t in G. #HamPath is the problem of computing, given a graph G and a node pair (s, t), the number of Hamiltonian paths from s to t in G. On the other hand, Hamiltonian Cycle is the problem of deciding, given a graph G, whether there is a Hamiltonian cycle in G. #HamCycle is the problem of computing, given a graph G, the number of Hamiltonian cycles in G.

3 Polynomial Time Reducibility of #3SAT to #HamCycle and to #HamPath

To prove #P-completeness of the problem of counting SAW's in two-dimensional grid-graphs and the problem of counting SAW's in hypercube graphs, we use a polynomial-time right-bit-shift reduction from #3SAT to #HamCycle, where each graph in the range of the reduction is planar and 3-regular (i.e., each node has three incident edges). With a slight modification this reduction becomes also a polynomial-time right-bit-shift reduction from #3SAT to #HamPath and has maximum-degree three.

The reduction is a variation of the polynomial-time many-one reduction due to Garey, Johnson, and Tarjan [GJT76] from 3SAT to the Hamiltonian Cycle decision problem of planar, 3-regular graphs. The reduction has the property that for each graph in the range of the reduction there is at least one edge that every Hamiltonian cycle of the graph must traverse and such an edge is easy to identify. So, by simply removing one of the edges common to all Hamiltonian cycles, the Garey–Johnson–Tarjan reduction becomes a polynomial-time many-one reduction from 3SAT to the Hamiltonian Path Problem of planar graphs having maximum degree three.

One cannot directly use the Garey–Johnson–Tarjan reduction to prove #P-completeness of #HamCycle or #HamPath, because the number of Hamiltonian paths representing a satisfying assignment depends on how the assignment satisfies each clause. More precisely, let φ be a satisfiable 3CNF formula and G be the planar 3-regular graph that the Garey–Johnson–Tarjan reduction produces on input φ. Let A be the set of all satisfying assignments of φ and P be the set of all Hamiltonian cycles in G. The Garey–Johnson–Tarjan reduction defines an onto mapping from P to A so that from each element path in P an element in A can be recovered in polynomial time. For each $a \in A$, let P_a denote the set of all Hamiltonian cycles in P that map to a by this onto mapping. Clearly, $\|P\| = \sum_{a \in A} \|P_a\|$. It holds that $\|P_a\| = 2^p 3^q$, where both p and q are linear functions of the following three quantities: the number of clauses such that a satisfies all the three literals, the number of clauses such that a satisfies exactly two literals, and the number of clauses such that a satisfies exactly one literal. By considering only instances of NAE3SAT and applying slight modifications to the Garey–Johnson–Tarjan reduction, we can make the multitude $\|P_a\|$ equal to 2^r such that r depends only on the input.

Let φ be a 3CNF formula such that #SAT(φ) needs to be evaluated. Suppose that φ has some $n \geq 1$ variables and some $m \geq 1$ clauses. Let ψ be the formula generated by applying the transformation in Lemma 2.3 to φ. ψ is defined on $n + m + 1$ variables and has $8m$ three-literal clauses and one single-literal clause. We construct a graph G from ψ by applying the Garey–Johnson–Tarjan reduction with slight modifications.

Basic components of the construction are the Tutte-gadget (see Figure 1), the XOR-gadget (Figure 2), and the OR-gadget (Figure 3). Here the first two gadgets are taken without changes from the Garey–Johnson–Tarjan reduction while the OR-gadget is our own device.

The Tutte-gadget is used to force branching. To visit c without missing a node, one has to either enter from a and visit b on its way or enter from b and visit a on its way. There are four ways to do the former and two ways to do the latter (see Figure 1 (b)).

The XOR-gadget is a ladder built using eight copies of the Tutte-gadget. In order to go through all the nodes in an XOR-gadget one has to enter and exit on the same vertical axis. For each of the two vertical axes there are $(4 \cdot 2)^4 = 2^{12}$ Hamiltonian paths. XOR-gadgets can be crossed without losing planarity by inflating the number of Hamiltonian paths (see Figure 2). Since four XOR-gadgets are added, the number of Hamiltonian paths is increased by a multiplicative factor of 2^{48}.

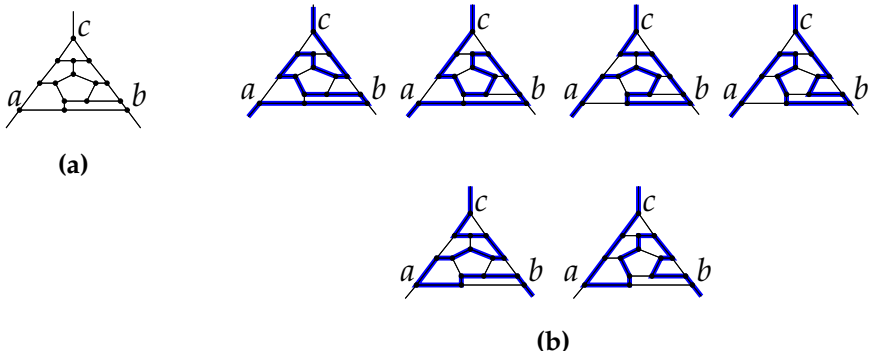

Figure 1: The Tutte Gadget. (a) The gadget. (b) Hamiltonian traversals of the nodes in the Tutte gadget. The top four are traversals connecting nodes a and c with b in the middle. The bottom two are traversals connecting nodes b and c with a in the middle.

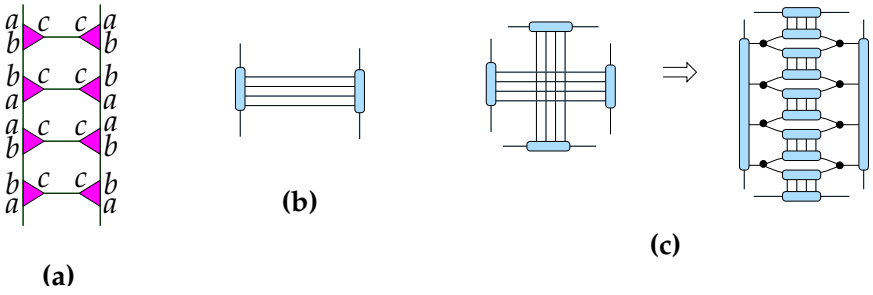

Figure 2: The XOR Gadget. (a) The structure of the XOR gadget. (b) A symbol to denote an XOR gadget. (c) Crossing of two XOR gadgets. On the left the four horizontal lines in one XOR gadget and the four vertical lines in the other XOR gadget need to be crossed. On the right the crossing of the lines are resolved by introduction of four additional XOR gadgets.

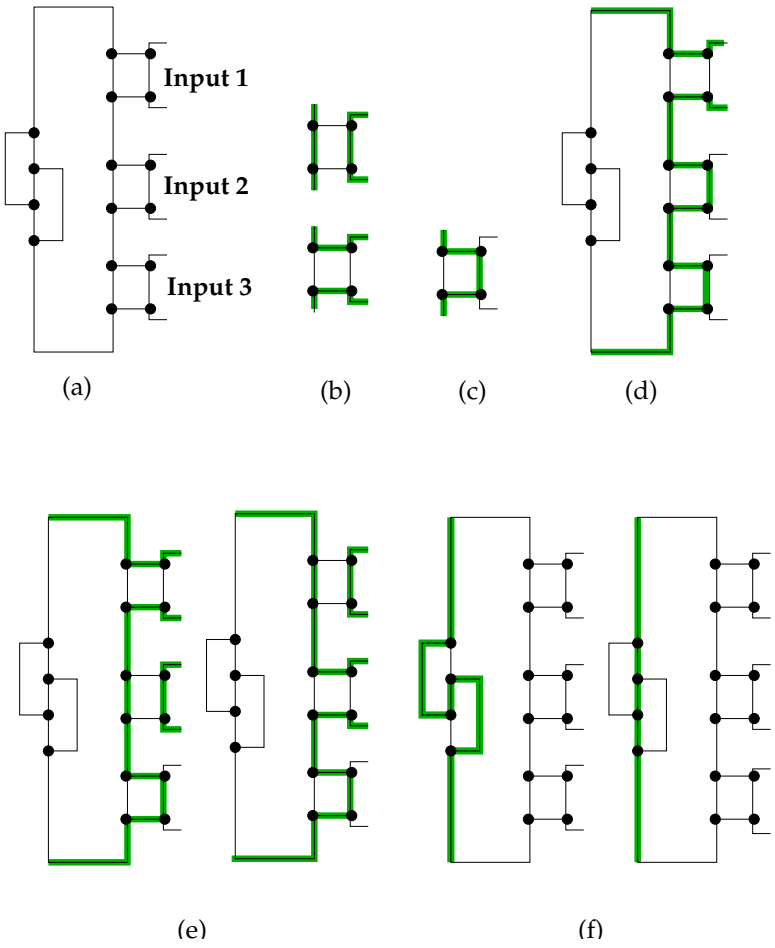

Figure 3: The OR Gadget. (a) A three-input OR gadget. (b) Two ways to traverse an input line whose value is true. (c) The way to traverse an input line whose value is false. (d) The traversal of input lines when only input 1 is true. (e) Two possible traversals of input lines when exactly two inputs are true. (f) Two ways to create a Hamiltonian traversal of the four nodes on the left side of the gadget.

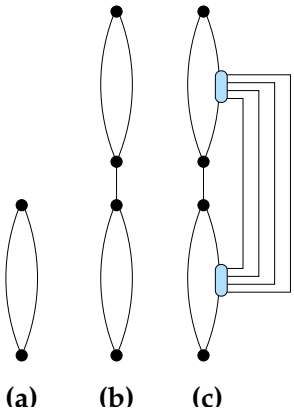

Figure 4: (a) A two-node cycle. (b) A pair of two-node cycles joined by an edge. (c) A pair of two-node cycles joined by an edge and by an XOR-gadget.

In an OR-gadget, each four-node rectangle on the right-hand side is considered to be an input. The line at the right end of an input part provides connection to the outside world. The one shown in Figure 3 is a three-input OR-gadget. Each input line will be either entirely available or entirely unavailable. If an input line is available we think of the situation as the input being assigned true as a value. Otherwise, we think of the situation as the input being assigned false. In the former case, there are two ways to traverse the line (see Figure 3 (b)). In the latter there is only one way to touch the two nodes on the line (see Figure 3 (c)). If the number of inputs that are assigned true is one, there is only one way to traverse the nodes on the input lines (see Figure 3 (d)). If the number is two, there are two possibilities (see Figure 3 (e)). Finally, there are two ways to traverse all the four nodes on the left-hand side of an OR-gadget (see Figure 3 (f)).

Another important components of the graph G are two-node cycles (see Figure 4). A node-node cycle is represented as a pair of nodes that are vertically lined up and two arcs connecting the two nodes. We refer to the two nodes by the *top node* and the *bottom node* and refer to the two arcs by the *right edge* and the *left edge*. The paths consisting solely of one of the two arcs are the Hamiltonian paths of a two-node cycle.

The graph G is essentially two vertical sequences of two-node cycles that are side-by-side. The right sequence is called Π and the left sequence is called Σ here. In each of the two sequences each neighboring cycle-pair is joined by an edge. The cycles in Σ are correspond to the literals of ψ, so there are exactly $24m + 1$ cycles in it. On the other hand, the cycles in Π correspond to truth assignments, so there are $2(n+m+1)$ cycles in it. The very bottom node of Σ and the very bottom node of Π are respectively called s and t. The first $24m$ cycles of Σ are divided into $8m$ three-cycle blocks, where for each i, $1 \leq i \leq 8m$, the ith block corresponds to the ith three-literal clause of ψ, i.e., the first of the three cycles corresponds to the first literal of the clause, the second cycle to the second literal, and the third cycle to the third literal (see Figure 5). The three cycles in each three-cycle block are connected to each other by an OR-gadget. The last cycle corresponds to the unique single-literal clause of ψ. To the left edge of the two-node cycle we attach a one-input OR-gadget. The sequence Π is divided into $n + m + 1$ blocks of cycle-pairs, where for each i, $1 \leq i \leq n + m + 1$, the ith pair corresponds to x_i, the ith variable. For each pair, the top cycle corresponds to $\overline{x_i}$ and the bottom to x_i. We connect the right edges of each pair by an XOR-gadget as shown in Figure 4. This has the effect of each Hamiltonian path of Π to select for each pair of two-node cycles exactly one cycle whose left edge is traversed, where the other cycle will be

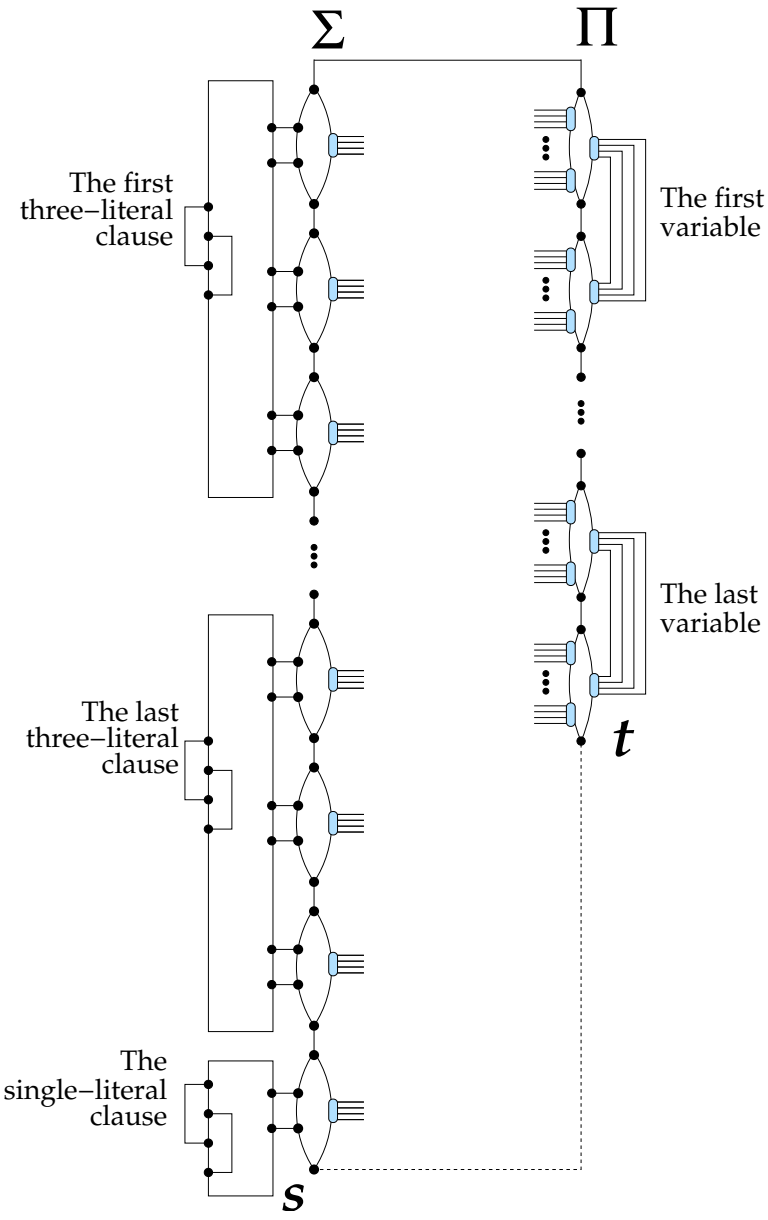

Figure 5: The Two Sequences, Σ and Π

traversed on the right edge. Now, for each i, $1 \leq i \leq 2(n+m+1)$, and each j, $1 \leq j \leq 24m+1$, if the literal represented by the ith cycle of Π is the literal at the jth position in Σ, join the left edge of the ith cycle in Π the right edge of the jth cycle in Σ by an XOR-gadget. Here, in the case in which two XOR-gadgets connecting Σ and Π need to be crossed, we resolve the issue by the crossing of XOR-gadgets described earlier.

Note that for every i, $1 \leq i \leq n+m+1$, traversing the XOR-gadget connecting the two right edges in the ith cycle-pair in Π corresponds to selection of a value for the ith variable. Here if the right edge of the top (respectively, bottom) cycle is used to traverse the XOR-gadget, then it frees up the left edge of the bottom (respectively, top) cycle, enforcing that all the XOR-gadgets attached to the left edge of the bottom (respectively, top) cycle are traversed from the Π side and that all the XOR-gadgets attached to the left edge of the top (respectively, bottom) cycle are traversed from the Σ side. We view this situation as the ith variable assigned the value true (respectively, false). So, each Hamiltonian path of the Π side corresponds to a truth assignment of ψ. Let a Hamiltonian path of the Π side be fixed and let α be the corresponding truth-assignment of ψ. Let j, $1 \leq j \leq 8m$, be an integer. If α does not satisfy the jth clause, then in each of the three cycles in the jth block in Σ, the right edge needs to be taken so as traverse the XOR gadgets attached to it, which means that the OR-gadget attached to the block cannot be traversed. If α satisfies exactly one literal, exactly one of the three left edges of the three cycles is available, so there are two ways to traverse all the nodes in the block. If α satisfies exactly two literals, exactly two of the three left edges of the three cycles are available, so there are four ways to traverse all the nodes in the block. As to the last single-cycle with one single-input OR-gadget, if the literal is not satisfied, then there is no way to traverse the OR-gadget, and if the literal is satisfied, then there are two ways to construct a Hamiltonian path in it. The formula ψ is designed so that every satisfying assignment of it satisfies exactly $4m$ three-literal clauses by satisfying exactly two literals and exactly $4m$ three-literal clauses by satisfying exactly one literal. So, each Hamiltonian path of G corresponds to a satisfying assignment of ψ. Furthermore, for each satisfying assignment α of G, the number of Hamiltonian paths of G that represent α is

$$2\left(4^{4m}2^{4m}2^{12(n+m+1)}2^{12(24m+1)}2^{48r}\right) = 2^{48r+12n+312m+25},$$

where r is the number of crossings of XOR-gadgets. Note that the degree of the nodes in G is all three except s and t, for both of which the degree is two.

By combining the above observations and the results in Proposition 2.1 and Lemma 2.3, we obtain the following lemma.

Lemma 3.1 *The problem of counting Hamiltonian paths in planar graphs of maximum degree three is* #P*-complete under* $\leq^p_{r\text{-}shift}$*-reductions.*

In the above construction, if we join s and t by an edge, then the resulting graph becomes 3-regular and the edge (s,t) is traversed by every Hamiltonian cycle of G (see the dashed line in Figure 5). Thus, we have strengthened the NP-completeness result by Garey, Johnson, and Tarjan as follows:

Lemma 3.2 *The Hamiltonian Cycle Problem of planar 3-regular graphs is* NP*-complete in the sense that there exists a polynomial-time many-one reduction f from 3SAT to the problem such that, together with another polynomial time computable function, f acts as a polynomial-time* $\leq^p_{r\text{-}shift}$*-reduction from* #3SAT *to* #HamCycle *of planar 3-regular graphs.*

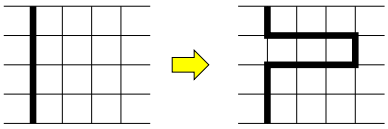

Figure 6: Bending a vertical line.

4 Self-Avoiding Walks in Two-Dimensional Grid Graphs

This section proves #P-completeness of the problem of counting the number of SAW's in the two-dimensional grid graph. As mentioned in the introduction, there are following six versions of the problem we are concerned with depending what type of SAW's are counted:

1. the SAW's from the origin to a specific point having a specific length,
2. the SAW's from the origin to any point having a specific length,
3. the SAW's between any two points having a specific length,
4. the SAW's from the origin to a specific point having any length,
5. the SAW's from the origin to any point having any length,
6. the SAW's between any two points having any length.

Theorem 4.1 *Each of the six types of the problem of counting the number of SAW's in two-dimensional grid graphs is complete for* #P *under* $\leq^p_{r\text{-}shift}$*-reductions.*

Proof. We first establish that the first three versions are each complete for #P under $\leq^p_{r\text{-}shift}$-reductions. Let f be an arbitrary #P function. Let g be the reduction from f to #HamPath stated in Lemma 3.1. It is known that planar graphs can be embedded in the two-dimensional grid in polynomial time (for example, see [CP95]). In the case when the maximum degree is 3, the embedding can be made so that there is no edge contention. We pick one such method. Let x be a string for which $f(x)$ needs to be evaluated. Let (G, s, t) be the value of g on input x. Let N be the number of nodes in G. Construct G' from G by adding two nodes s' and t' and two edges (s, s') and (t, t'). We apply our embedding algorithm to G' and obtain a two-dimensional grid graph. Let U be the set of all images of the nodes in G with respect to this embedding. Since s' and t' are both exterior nodes, we may assume that s' is mapped to be the node with the smallest x-coordinate among those in U having the smallest y-coordinate and t' is mapped to the node with the largest x-coordinate among those in U having the largest y-coordinate. We may also assume that s' is $(0, 0)$. If not, we will shift the embedding accordingly so that s' is located on $(0, 0)$. This is E_0. Let Q be the integer such that $f(x)$ is equal to the number of Hamiltonian paths in G divided by 2^Q.

We construct a new two-dimensional grid graph E_1 as follows: We enlarge E_0 by a factor of four. In other words, we move each point (a, b) of E_0 to $(4a, 4b)$ and replace each edge $((a, b), (a', b'))$ of E_0 by the four-unit-length straight-line path from between $(4a, 4b)$ and $(4a', 4b')$. For each edge $e = (u, v)$ of G', if the edge e is realized by a straight vertical line, then we bend the straight line as follows: Suppose that the straight line runs from (a, b) to (a, b'), where $b' \leq b - 4$. We replace the edge $((a, b-1), (a, b-2))$ by the path:

$$(a, b-1), (a+1, b-1), (a+2, b-1), (a+3, b-1),$$
$$(a+3, b-2), (a+2, b-2), (a+1, b-2), (a, b-2).$$

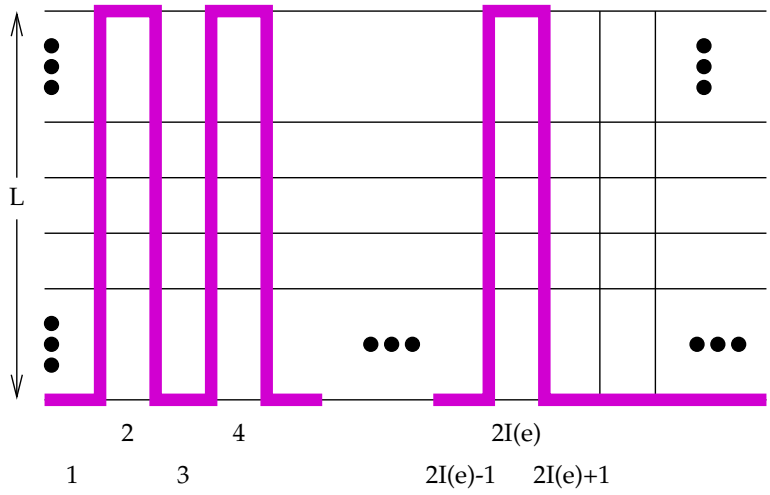

Figure 7: Construction of towers.

(see Figure 6). Since we have already enlarged E_0 by a factor of four, bending vertically straight line paths does not interfere with the other paths. On the other hand, if the edge e is realized by a path that includes a horizontal edge of the two-dimensional grid, since we have enlarged the embedding by a factor of four, the path realizing e contains a horizontal line of length four. This is the graph E_1. For every edge e of G', the path realizing in E_1 contains a horizontal line of length at least three.

Furthermore, we construct a grid graph E_2 from E_1. Let L be the smallest integer l such that l is a power of 2, $l \geq N^2$, and for all edges e of G' the path realizing e in E_1 has length at most l. We enlarge E_1 by a factor of L. For each edge e of G', let $\delta(e)$ denote the length of the path that realizes e in the enlarged E_1. Then, for all edges e of G, $\delta(e)$ is a multiple of $2L$ (since the lengths of the paths realizing edges of G' in E_1 are even), is at most L^2, and is at least $2L$. Also, for all edges e of G', the path in the enlarged E_1 that realizes e contains a horizontal line having length at least $3L$.

Now for each edge e of G' we do the following. We pick a horizontal line of length $2L+1$ that is a part of the path that realizes e, call the $2L+1$ grid edges $a(1), \ldots, a(2L+1)$ from left to right. Let $I(e) = \frac{L^2-\delta(e)}{2L}$. Note that $I(e)$ is an integer. We replace for each i, $1 \leq i \leq I(e)$, the edge $a(2i)$ by the tower of width 1 and of height L going from $a(2i)$ (see Figure 7). This is E_2. Each tower increases the length of the path by $2L$. So, for all edges e of G', e is realized by a path of length L^2.

Let $h = L^2(N+1)$. Since G' has $N+2$ nodes hence every Hamiltonian path of G' is realized by a path in E_2 of length h. Furthermore, every path in E_2 having length h corresponds to a Hamiltonian path in G'. So, regardless of whether the end points of paths are specified or not, the number of SAW's in E_2 having length h is exactly the number of Hamiltonian paths in G'. Let τ' in E_2 be the image of t'. Define $R_1(x) = (E_2, \tau', h)$ and let $R_3(x)$ be the same function as R_3 which witnesses the $\leq^p_{r\text{-}shift}$-reduction from f to counting Hamiltonian paths in planar graphs of maximum degree three (see Lemma 3.1). Then the pair (R_1, R_3) witnesses that the types 1, 2, and 3 of the counting problem of SAW's in two dimensional grid graphs are each #P-complete under $\leq^p_{r\text{-}shift}$-reductions.

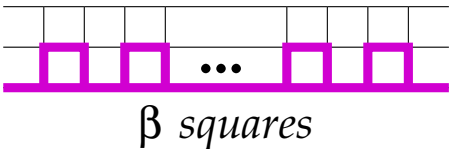

Figure 8: Squares attached.

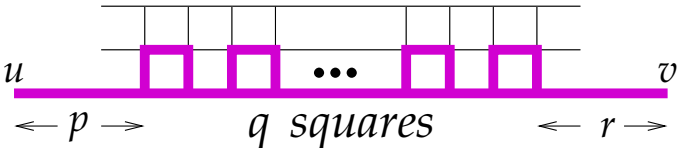

Figure 9: $S(p, q, r)$.

Let $\alpha = 4L^2$ and $\beta = L^2$. We make further modifications to E_2 and construct E_3. We enlarge E_2 by a factor of α. This is carried out by transforming each edge $((a, b), (a', b'))$ of E_2 to the straight line of length α between $(\alpha a, \alpha b)$ and $(\alpha a', \alpha b')$. Then for each edge e of G' identify a horizontal line of length α and attach above the line β unit-size squares with a gap of unit length in between (see Figure 8). This is E_3.

Note that for each edge e of G', now e is realized by a set of 2^β paths. So, the number of SAW's between the origin and the image τ of t is at most

$$2^{\beta(N-1)}\#\mathrm{HamPath}(G, s, t) + 2^{\beta(N-2)}\text{the number of } s\text{-}t \text{ non-Hamiltonian simple paths.}$$

Since the degree of every node in G other than s and t is exactly three, so the number of s-t non-Hamiltonian simple paths in G is at most

$$1 + 2 + 2^2 + \cdots + 2^{N-2} < 2^{N-1}.$$

Since $\beta \geq N$, the number of SAW's between the origin and the image of t in E_3 is

$$2^{\beta(N-1)}\#\mathrm{HamPath}(G, s, t) + \rho,$$

where $0 \leq \rho < 2^{\beta(N-2)+(N-1)} < 2^{\beta(N-1)}$. So, by letting $R_1(x) = (E_3, \tau)$ and $R_3(x) = R_3^f(x) + \beta(N-1)$, where R_3^f denotes the R_3 function which witnesses the $\leq^p_{r\text{-}shift}$-reduction from f to counting Hamiltonian paths in planar graphs, we have that the fourth type is complete for #P under $\leq^p_{r\text{-}shift}$-reductions.

To show #P-completeness of the last two problems, let for each triple of positive integers p, q, r, $S(p, q, r)$ be a structure shown in Figure 9, which is a sequence of q unit-size squares such that each pair of neighboring squares are connected by a unit-length edge at the bottom and a line having length p and a line having length r are respectively attached to the left and and the right end of the sequence. Since each edge of e is realized by a path having length L^2 in E_2 and we have enlarged E_2 by a factor of α, the attachment of squares described in the above is equivalent to replacing each edge of e by an $S(p, \beta, r)$ for some positive integers p and r such that $p + r + 2\beta - 1 = \alpha L^2$.

Let p, q, r, and ℓ be positive integers such that $p + r + 2q - 1 \leq \ell$. Let u and v denote the left and the right ends of the structure, respectively. The simple paths within $S(p, q, r)$ have the following properties:

1. The number of those that connect u and v is 2^q.
2. The number of those that touch u but not v is
$$p + 7(1 + 2 + 2^2 + \cdots + 2^{q-1}) + 2^q(r-1) = 2^q(r+6) + p - 7.$$
This is less than $2^q\ell$. Similarly, the number of those that touch v but not u is less than $2^q\ell$.
3. The number of those that touch neither u nor v is less than $2^{q+1}\ell^2$. This can be shown as follows:
   - The number of length-zero paths touching neither u nor v is at most $2q + \ell - 1$.
   - The number of length-one paths touching neither u nor v is at most $3q + \ell - 2$.
   - The number of length-two paths within single squares is $4q$.
   - The number of length-three paths within single squares is $4q$.
   - The number of paths π such that either the left end of π is a point between u and the leftmost square or the right end of π is a point between v and the rightmost square is less than $(2^q\ell^2)/2$.
   - All the paths that are not consider in the previous cases are those that have length more than one and connect between two distinct squares. For every i, $2 \leq i \leq q$, the number of such paths that touch precisely i squares is equal to $7^2 \cdot 2^{i-2}$ (since there are seven possibilities for one end point and seven for the other end point). The sum of this for all i, $2 \leq i \leq q$, is $49(2^{q-1} - 1)$.

   By summing up all the upper bounds calculated in the above, the number of simple paths that touch neither u nor v is bounded from above by
$$2\ell + 13q + (\ell^2 + 49)2^{q-1}.$$

Let σ, σ', τ, and τ' be respectively the images of s, s', t, and t' in E_3. Here σ' has the smallest y-coordinate and τ' has the large y-coordinate. Then we modify E_3 attaching to σ' a vertical $S(6, 2\beta, 1)$ downwards, where the other end of the S-structure is called σ'', and attaching to τ' a vertical $S(6, 2\beta, 1)$ downwards, where the other end of the S-structure is called τ'' Let this two-dimensional grid graph be E_4 (see Figure 10 for the whole construction).

Let Y be the number of s-t Hamiltonian paths in G. Then Y is equal to the number of s'-t' paths having length $N + 1$ in G'. For each simple path π in G', let $M(\pi)$ denote the set of all SAW's $\gamma = [v_1, \ldots, v_k]$ in E_4 such that elimination from γ of all the nodes not corresponding to the nodes of G' produces π. Also, for each simple path π in G', let $\mu(\pi)$ denote the cardinality of $M(\pi)$. We first establish #P-completeness of the fifth type.

Let Z_1 be the number of SAW's in E_4 from the origin (recall that the origin of E_4 is σ') to any node in E_4. We evaluate Z_1 as follows:

- Suppose π is a length $N + 1$ path from s' to t'. Each SAW corresponding to π begins in σ' or in σ'' or somewhere between these nodes and ends in τ' or in τ'' or somewhere between them. By (1) and (2) in the above, the number of paths that touch σ' and any point between σ' and σ'' (including σ'') is
$$2^{2\beta} + 7(2^{2\beta}) = 2^{2\beta+3}.$$
The same holds for the number of paths that touch τ' and any point between τ' and τ''. So,
$$\mu(\pi) = 2^{2\beta+3}2^{\beta(N+1)}2^{2\beta+3} = 2^{\beta(N+5)+6}.$$

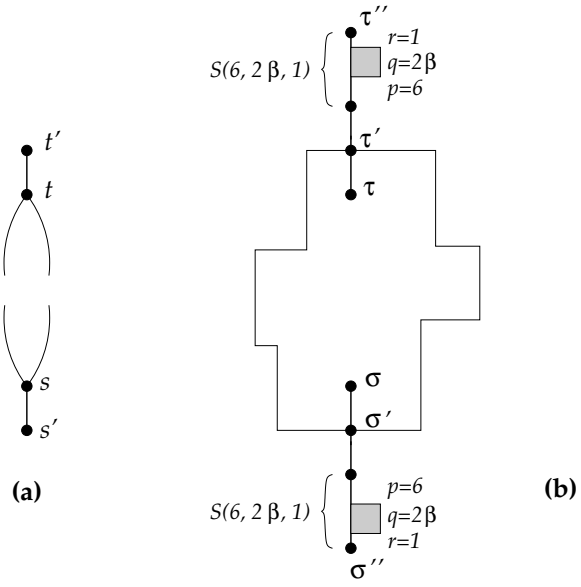

Figure 10: (a) Graph G' and (b) the corresponding modifications to E_3.

- Suppose π is an s'-t' path having length at most N. Then π does not include a Hamiltonian path of G. So, by an analysis similar to the previous one,

$$\mu(\pi) \leq 2^{2\beta+3} 2^{\beta N} 2^{2\beta+3} = 2^{\beta(N+4)+6}.$$

- Suppose that π is a path from s' having length at least one to a node other than t'. Then π traverses at most N edges of G'. Let u be the node in E_4 corresponding to the end point of π. There are two directions in E_4 to extend beyond u. By property (2) in the above, there are at most $2(2^\beta \alpha L^2)$ possibilities for the selection. Thus,

$$\mu(\pi) \leq 2^{2\beta+3} 2^{\beta N} 2(2^\beta \alpha L^2) = 2^{\beta(N+3)+4} \alpha L^2.$$

- Suppose that π is the single node path s'. Then each SAW corresponding to π begins in σ' and either ends somewhere between σ' and σ'' (including σ'') or somewhere between σ' and σ (excluding σ). Hence by property (2) in the above

$$\mu(\pi) < 2^{2\beta+3} + 2^\beta \alpha L^2.$$

Let Y be the number of s-t Hamilton paths in G. Then there are exactly Y paths of the first kind. The number of paths π of the second kind is at most

$$1 + 2 + 2^2 + \cdots + 2^{N-2} < 2^{N-1}.$$

The number of paths π of the third kind is at most

$$1 + 2 + 2^2 + \cdots + 2^{N-2} + 2^{N-1} < 2^N$$

The number of paths π of the last kind is one. Thus, the sum

$$(2^{\beta(N+4)+6}) 2^{N-1} + (2^{\beta(N+3)+4} \alpha L^2) 2^N + (2^{2\beta+3} + 2^\beta \alpha L^2)$$

is an upper bound for the number of SAW's in E_4 which correspond to the paths of the last three kinds. Since $\alpha = 4\beta$, $\beta = L^2$, and $L \geq N^2$, for all but finitely many N, this sum is less than $2^{\beta(N+5)+6}$. So,

$$Z_1 = (2^{\beta(N+5)+6})Y + \rho_1,$$

where $0 \leq \rho_1 \leq (2^{\beta(N+5)+6}) - 1$. Thus, Y can be recovered from Z_2 by right-bit-shifting. This proves that the fifth type is #P-complete.

Finally, let Z_2 be the number of any SAW's in E_4. As in the previous cases, we evaluate Z_2 as follows:

- Suppose π is a length $N+1$ path from s' to t'. Then

  $$\mu(\pi) = 2^{\beta(N+5)+6}.$$

  There are Y paths of this kind.

- Suppose π is an s'-t' path having length at most N. Then

  $$\mu(\pi) = 2^{\beta(N+4)+6}.$$

  The number of path of this kind is at most 2^N

- Suppose that π is a path from s' having length at least one to a node other than t' or a path from t' having length at least one to a node other than s'. By the previous analysis for the third type it holds

  $$\mu(\pi) \leq 2^{\beta(N+3)+4}\alpha L^2.$$

  and there are at most $2(2^N) = 2^{N+1}$ paths of this kind.

- Suppose that π is the single node path s' or the single node path t'. Then

  $$\mu(\pi) < 2^{2\beta+3} + 2^{\beta}\alpha L^2.$$

- Suppose that π is a path in G. Then

  $$\mu(\pi) \leq (2(2^{\beta}\alpha L^2))^2 2^{\beta(N-1)} = 2^{\beta(N+1)+2}\alpha^2 L^4$$

  and there are at most $N(2^N)$ paths of this kind.

- Finally suppose that π is the empty path (i.e. it has no node at all). Then since both s' and t' have degree 1 and the remaining nodes of G' have degree 3 hence G' has exactly $3N/2+1$ edges and therefore in E_4 we have $3N/2+1$ structures $S(p,\beta,r)$, with $p+r+2\beta-1 = \alpha L^2$, corresponding to the edges. Recall that in E_4 in we have additionally 2 structures $S(6, 2\beta, 1)$. Then we conclude

  $$\mu(\pi) \leq (3N/2+1)(2^{\beta}\alpha L^2) + 2(2^{2\beta+3}) \leq (3N/2+3)(2^{2\beta+3}\alpha L^2).$$

It is not hard to see that

$$Z_2 = 2^{\beta(N+5)+6}Y + \rho_2,$$

where $0 \leq \rho_2 \leq 2^{\beta(N+5)+6} - 1$. So, Y can be recovered from Z_2 by right-bit-shifting. Thus, the sixth type is #P-complete. □

5 Self-Avoiding Walks in Hypercube Graphs

5.1 #P-Completeness

Let f be an arbitrary #P function. Let g be a reduction defined in the proof of Theorem 4.1. Let x be an input to f. Let E_4 be the two-dimensional grid graph that g on x produces. We show that the graph E_4 can be embedded in an $\mathcal{O}(\log n)$-dimensional hypercube so that there is an integer d such that every edge of E_4 is realized by a path having length d. Then, by simply replacing the parameter λ by λd, all the analysis in the proof of Theorem 4.1 hold. So, it remains to show that such embedding is indeed possible.

Lemma 5.1 *Let $d \geq 1$ be an integer. There exists a polynomial-time computable embedding $\mathcal{E}$ and a pair of polynomial-time computable functions $s, \ell : \mathbf{N} \to \mathbf{N}$ such that the following properties hold:*

1. *$s(n) = \mathcal{O}(\log n)$ and $\ell(n) = \mathcal{O}(\log \log n)$.*
2. *For every $n \geq 1$, and every undirected graph $G = (V, E)$ of n nodes having maximum degree d, $\mathcal{E}(G)$ is an embedding of G in the $s(n)$-dimensional hypercube $HC^{(s(n))}$ such that*
   - *for all $u \neq v$ in G, $\mu(u) \neq \mu(v)$, and*
   - *for every edge $e = (u, v)$ in G, $\nu(e)$ has dilation $2^{\ell(n)}$ and visits no $\mu(w)$ between $\mu(u)$ and $\mu(v)$.*

   Here μ and ν are respectively the embedding of nodes and the embedding of edges specified by $\mathcal{E}(G)$.

Proof. Let $d, n, m \geq 1$ be integers. Let G be a degree d graph with n nodes and m edges. Note that $m < n^2/2$. Identify the nodes in G with the set of integers from 0 to $n-1$ and identify the edges in G with the set of integers from 0 to $m-1$. For each node u of G, fix an enumeration of its neighbors. For every edge (u, v) of G, let $I_u(v)$ denote the order of v in the enumeration of the neighbors of u. Let q be the smallest integer such that $3q + 4$ is a power of 2 and such that $q \geq \lceil \log n \rceil$. Let $s = 6q + d + 2$. Let H denote the s-dimensional hypercube. Each node of G will be viewed as a q-bit binary number and each edge of G as a $2q$-bit binary number. For a binary string u, let $\overline{u}$ denote the string constructed from u by flipping all of its bits.

The s-bit representation of a node in H is divided into the following five components.

- The Node Part: This part has length $2q$. Here for each node $u = u_1 \cdots u_q$ of G, $\overline{u}u$ encodes u. Note that this encoding has exactly q many 1s in it.
- The Edge Part: This part has length $4q$ and is used to encode the edge to be traversed. Here each edge $e = e_1 \cdots e_{2q}$ is encoded as $\overline{e}e$.
- The Neighbor Part: This part has length d and is used to encode the value of $I_u(v)$ or the value of $I_v(u)$ when the edge (u, v) is traversed.
- The Switch Part: This part has length 2.

Let u, v, w, y be binary strings of length $2q$, $4q$, d, and 2, respectively. Then, $\langle u, v, w, y \rangle$ denotes the node $uvwy$ in $HC^{(s)}$.

The embedding $\mathcal{E}(G) = (\mu, \nu)$ is defined as follows: For each node $u = u_1 \cdots u_q$,

$$\mu(u) = \langle\, \overline{u}u, 0^{2q}, 0^d, 000 \,\rangle.$$

As for ν, let $e = (u, v)$ be an edge in G such that $u < v$. Let $A = a_1 \cdots a_{2q} = \overline{u}u$, $B = b_1 \cdots b_{2q} = \overline{v}v$, $C = c_1 \cdots c_{4q} = \overline{e}e$, $W_1 = 0^{I_u(v)-1}10^{d-I_u(v)}$, and $W_2 = 0^{I_v(u)-1}10^{d-I_v(u)}$. Let $i_1, \ldots, i_q$ be the enumeration of all the positions at which A has a bit 1 in increasing order. Let $j_1, \ldots, j_q$ be the enumeration of all the positions at which B has a bit 1 in increasing order. Let $k_1, \ldots, k_{2q}$ be the enumeration of all the positions at which C has a bit 1 in increasing order. For each t, $1 \leq t \leq q$, let $A_t = 0^{i_t} a_{i_t+1} \cdots a_{2q}$ and $B_t = 0^{j_t} b_{j_t+1} \cdots b_{2q}$. Also, for each t, $1 \leq t \leq 2q$, let $C_t = c_1 \cdots c_{k_t} 0^{4q-k_t}$. Note that $A_q = B_q = 0^q$ and $C_{2q} = C$. The edge e is represented by a path from $\langle\, A, 0^{4q}, 0^d, 00 \,\rangle$ to $\langle\, 0^q, C, 0^d, 11 \,\rangle$ and a path $\langle\, B, 0^{4q}, 0^d, 00 \,\rangle$ to $\langle\, 0^q, C, 0^d, 11 \,\rangle$, each of length $3q + 4$. The first path is defined by:

$$\langle\, A, 0^{4q}, 0^d, 00 \,\rangle, \langle\, A, 0^{4q}, W_1, 00 \,\rangle, \langle\, A, 0^{4q}, W_1, 10 \,\rangle, \langle\, A, C_1, W_1, 10 \,\rangle, \cdots, \langle\, A, C_{2q}, W_1, 10 \,\rangle,$$
$$\langle\, A_1, C, W_1, 10 \,\rangle, \cdots, \langle\, A_q, C, W_1, 10 \,\rangle, \langle\, 0^q, C, 0^d, 10 \,\rangle, \langle\, 0^q, C, 0^d, 11 \,\rangle.$$

The second path is defined with B in place of A. The total length of the join of the two paths is $6q + 8$ and this is a power of 2 by assumption. Note that every node in the entire path contains one of the following: (i) C in the edge part, (ii) A in the node part and W_1 in the neighbor part, or (iii) B in the node part and W_2 in the neighbor part. So, no two edges share internal nodes in their path representation. It is not hard to see that the embedding can be computed in logarithmic space. This proves the lemma. □

5.2 Scaling up to exponential time

Let f be a function belonging to #EXP. Then there is a one-tape nondeterministic exponential-time machine M such that $f = \#acc_M$. It can be assumed that there is a polynomial p such that for all strings x, M on input x halts at step $2^{p(|x|)}$. By applying the tableau method to the computation of M and then by reducing the number of occurrences of each variable by introducing its copies, it can be shown that the computation of f can be transformed to #SAT by a polynomial-time local computation.

Lemma 5.2 *For each $f \in$ #EXP, there is a function R that satisfies following conditions.*

1. *For every string x, $R(x)$ is a 3CNF formula.*
2. *For every string x, $f(x) = \#\mathrm{SAT}(R(x))$.*
3. *For every string x, $R(x)$ can be locally computed in polynomial time in the following sense:*
   (a) *For each variable y of $R(x)$, y or $\overline{y}$ appears in exactly three clauses.*
   (b) *For each string x, let $\nu(x)$ denote the number of variables in $R(x)$. Then ν is polynomial-time computable.*
   (c) *For each string x, let $\mu(x)$ denote the number of variables in $R(x)$. Then μ is polynomial-time computable.*
   (d) *For each string x and each i, $1 \leq i \leq \mu(x)$, let $C(x, i)$ be the ith clause in $R(x)$. Then C is polynomial-time computable.*
   (e) *For each string x and each i, $1 \leq i \leq \nu(x)$, let $S(x, i)$ be the set of all indices j such that the ith variable appears in $C(x, j)$. Then S is polynomial-time computable.*

Proof. A proof of this result can be found in [Pap94, Chapter 20]. □

Now apply conversion of Lemma 2.3 to each formula generated by R. Denote the resulting transformation from the set of strings to NAE3SAT by R'. Since R is locally polynomial-time computable, so is R'. Modify the construction for Lemma 3.1 so that crossing of XOR-gadgets is allowed. Then the maximum degree remains three and the scaling factor becomes $2^{12n+352m}$. The connectivity of the gadgets in the graph essentially depends only on the occurrences of corresponding variables, so the mapping can be computed locally in polynomial time. Now apply the embedding described above. Denote the resulting transformation from the set of strings to the set of hypercube graphs by H. The length of the path is proportional to the logarithm of the number of nodes, so they are polynomially bounded. For each string x, let $d(x)$ be the dimension of $H(x)$, $N(x)$ denote the set of all strings of length $d(x)$ that encode a node in $H(x)$, and $E(x)$ denote the set of all strings ww' having length $2d(x)$ such that (w, w') is an edge of $H(x)$.

Lemma 5.3 *d is polynomial-time computable. There are polynomial-time computable functions C_N and C_E such that $C_N(x)$ and $C_E(x)$ are both polynomial-size boolean circuits and accept $N(x)$ and $E(x)$, respectively.*

This lemma gives the following theorem.

Theorem 5.4 *The problem of counting the number of simple paths in hypercube graphs is* #EXP*-complete under $\leq^p_{r\text{-}shift}$-reductions if the graphs are specified by circuits.*

6 Complexity of Checking the Self-Avoiding Property

This section studies the problem of testing whether a path on the complete two-dimensional grid is a SAW. We encode SAW's using strings over alphabet $\{U, D, L, R\}$ in a natural way: symbols of such strings will describe in an obvious way (i.e. U stays for up-move, D for down, and so on) successive unit-length parts of a given SAW. Let L_{SAW} denote the set of all strings over $\{U, D, L, R\}$ which encode correct SAW's. Let 1-NL denote the set of all languages decided by logarithmic space-bounded nondeterministic TMs with one-way input-heads. Let co-1-NL denote the set of all languages whose complements belong to 1-NL. It is easy to see that L_{SAW} belongs to L and to co-1-NL.

Proposition 6.1 $L_{\mathrm{SAW}} \in \mathrm{L} \bigcap \text{co-1-NL}$.

Proof. Let $n \geq 1$ be an integer and $w = w_1 \cdots w_n$ be a word of length n over the alphabet $\Gamma = \{U, D, L, R\}$. Then $w \notin L_{\mathrm{SAW}}$ if and only if there exist some i and j, $1 \leq i < j \leq n$ such that the path $w_i \cdots w_j$ is a cycle. Let $y = y_1 \cdots y_m$ be a nonempty path in Γ^*. Then y is a cycle if and only if

$$\begin{aligned} &\|\{i \mid 1 \leq i \leq m \wedge y_i = U\} - \|\{i \mid 1 \leq i \leq m \wedge y_i = D\} \\ &= \|\{i \mid 1 \leq i \leq m \wedge y_i = L\} - \|\{i \mid 1 \leq i \leq m \wedge y_i = R\} \\ &= 0. \end{aligned}$$

It is easy to see that this equality can be tested using a deterministic logspace with one-way input head. To test non-membership in L_{SAW}, a one-way nondeterministic logspace machine has only to select nondeterministically i and j while scanning the input and test whether the portion of the

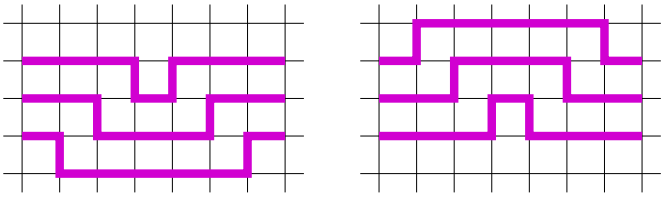

Figure 11: The SAW's of the Gadget G_0 (left) and G_1 (right).

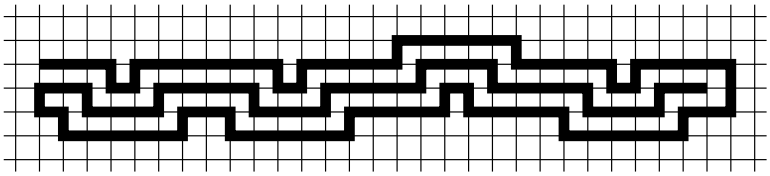

Figure 12: SAW encoded by string $W(X)Z(X)$, with $X = 0010$.

input selected is a cycle. Thus, $\overline{L_{\mathrm{SAW}}} \in$ 1-NL, so $L_{\mathrm{SAW}} \in$ co-1-NL. This test can be carried out deterministically in logspace if the input head is two-way. Thus, $L_{\mathrm{SAW}} \in$ L. □

Proposition 6.1 immediately raises the question of whether $L_{\mathrm{SAW}} \in$ 1-NL. How likely is it that this containment holds? In the theorem below we prove that the containment does not hold.

Theorem 6.2 *To recognize L_{SAW} each 1-way nondeterministic Turing machine needs at least linear space.*

Proof. Let us assume that a nondeterministic Turing machine M with a 1-way input head recognizes the language L_{SAW} in a sublinear space. Below we construct for a given integer $n > 0$ a walk of length $3(9n) + 5$ that is not a self-avoiding one but nevertheless M accepts a string encoding the walk. We start with a construction of a family of 2^n SAW's each of the length $27n + 5$. To this aim we use the following Gadgets G_0 and G_1 each of them consisting of three SAW's of length 9. We call the SAW's respectively the upper-border, lower-border, and the backbone and we illustrate them in Figure 11.

Now let A_0 be a string which encodes the upper border of G_0, let B_0 be a string encoding its lower border and finally let C_0 be a code of the backbone of G_0. We will traverse the upper borders and the backbones from left to right and the lower borders from right to left. Hence we have: $A_0 = RRRDRURRR$, $B_0 = LDLLLLLUL$, and $C_0 = RRDRRRURR$. In the similar way we encode the SAW's of G_1: $A_1 = RURRRRRDR$, $B_1 = LLLULDLLL$, and $C_1 = RRURRRDRR$. Next for a binary string $X = b_1 b_2 \dots b_n$ of the length n we define

$$W(X) := A_{b_1} A_{b_2} \dots A_{b_n}\ RDDL\ B_{b_n} \dots B_{b_2} B_{b_1}\ U \quad \text{and}$$
$$Z(X) := C_{b_1} C_{b_2} \dots C_{b_n}.$$

Figure 12 shows an example of a SAW encoded by string $W(X)Z(X)$.

Clearly $W(X)Z(X)$, with $X \in \{0,1\}^n$, represent 2^n different SAW's. Now using a counting argument one can show that if M recognizes L_{SAW} in a sublinear space then there exist two different binary strings $X, X' \in \{0,1\}^n$ such that M can not distinguish between prefixes $W(X)$ and $W(X')$. Speaking more formally for the input strings $W(X)Z(X)$ and $W(X')Z(X')$ there exist

two accepting computations of M such that M reading the last symbol of $W(X)$ is in the same configuration as reading the last symbol of $W(X')$. Hence there exists an accepting computation of M on the input string $W(X)Z(X')$, too. But this leads to a contradiction because this string encodes a walk that is not a self-avoiding one. □

The situation changes drastically if one restricts the walks on the complete two-dimensional grid to up-side SAW's, i.e. to the self avoiding walks that move up and sideways but not down. Let us denote by $L_{\text{up-side}}$ the set of all strings over $\{U, L, R\}$ which encode such SAW's. Then obviously this language can be recognized even by a deterministic 1-way finite automaton. Below we show how using this fact one can deduce a formula for the number of up-side SAW's of a given length n. Let us fix the following automaton M for $L_{\text{up-side}}$, with the states $Q = \{q_U, q_L, q_R\}$, where all of the states are accepting and q_U is the initial state. The transition function $\delta : Q \times \{U, L, R\} \rightarrow 2^Q$ is defined as follows: for any $x, y \in \{U, L, R\}$ let

$$\delta(q_x, y) := \begin{cases} \emptyset & \text{if } x = L, y = R \text{ or } x = R, y = L \\ \{q_y\} & \text{otherwise.} \end{cases}$$

Now consider the following transition matrix

$$A := \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

that can be interpreted as follows: if the states q_U, q_L, q_R correspond respectively to integers $1, 2, 3$ then $A(i, j) = 1$ if and only if M being in state corresponding to i can reach in one step the state corresponding to j. If one defines

$$\begin{bmatrix} f_n \\ g_n \\ h_n \end{bmatrix} := A^n \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

then obviously for any integer $n \geq 1$, f_n (resp. g_n, h_n) is equal to the number of words of the length n that are accepted by M when it starts in state q_U (resp. q_L, q_R). Hence f_n describes at the same time the number of up-side SAW's of the length n. To find a formula for this number observe that

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 2 \end{bmatrix}$$

and in general if for $n \geq 1$

$$A^n \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{2n} \\ a_{2n-1} \\ a_{2n-1} \end{bmatrix}$$

then

$$\begin{bmatrix} a_{2(n+1)} \\ a_{2(n+1)-1} \\ a_{2(n+1)-1} \end{bmatrix} = A^{n+1} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{2n} \\ a_{2n-1} \\ a_{2n-1} \end{bmatrix} = \begin{bmatrix} a_{2n} + 2a_{2n-1} \\ a_{2n-1} + a_{2n-1} \\ a_{2n-1} + a_{2n-1} \end{bmatrix}.$$

It is easy to check that for the integer sequence $a_1, a_2, a_3, \ldots$ defined as above, it holds for any integer $n \geq 1$ that:

$$\begin{aligned} a_{2n-1} &= \tfrac{1}{2\sqrt{2}}\left[(1+\sqrt{2})^{n+1} - (1-\sqrt{2})^{n+1}\right] \quad \text{and} \\ a_{2n} &= \tfrac{1}{2}\left[(1+\sqrt{2})^{n+1} + (1-\sqrt{2})^{n+1}\right] \end{aligned}$$

(for a generating function of this sequence see e.g. [SP95]). Hence for the number of up-side SAW's $f_n = a_{2n}$ we become the formula obtained by previously by Williams [Wil96].

Acknowledgment

The authors are grateful to Joerg Rothe for his helpful comments and his careful reading of a draft to detect errors in an early version of the paper and to Andreas Jakoby and Dick Lipton for stimulating discussions.

References

[CG96] A. R. Conway and A. J. Guttmann. Square lattice self-avoiding walks and corrections-to-scaling. *Physical Review Letters*, 77(26):5284–5287, 1996.

[CP95] M. Chrobak and T. H. Payne. A linear time algorithm for drawing a planar graph on a grid. *Information Processing Letters*, 54(4):241–246, 1995.

[GJT76] M. Garey, D. Johnson, and E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.

[MS93] N. Madras and G. Slade. *The Self-Avoiding Walk.* Birkhäuser, Boston, MA, 1993.

[Pap94] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[PY86] C. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.

[RS93] D. Randall and A. Sinclair. Testable algorithms for self-avoiding walks. In *Proceedings of 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 593–602. ACM/SIAM, 1993.

[Sch78] T. Schaefer. The complexity of satisfiability problem. In *Proceedings of 10th Symposium on Theory of Computing*, pages 216–226. ACM Press, 1978.

[SP95] N. J. A. Sloane and S. Plouffe. *The Encyclopedia of Integer Sequencfes.* Academic Press, San Diego, CA, 1995.

[Val79a] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[Val79b] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[Wel93] D. Welsh. *Complexity: Knots, Colourings and Counting.* Cambridge University Press, 1993.

[Wil96] L. Williams. Enumerating up-side self-avoiding walks on integer lattices. *Electronic Jounral on Combinatorics*, 3(1):Research Paper 31, 1996.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'