Enumerating Hamiltonian Cycles

Ville H. Pettersson[*]

Department of Communications and Networking
Aalto University School of Electrical Engineering
P.O. Box 13000, 00076 Aalto, Finland

ville.pettersson@aalto.fi

Submitted: Jul 4, 2014; Accepted: Sep 26, 2014; Published: Oct 9, 2014
Mathematics Subject Classifications: 05C30, 05C38, 05C45, 05C85

Abstract

A dynamic programming method for enumerating hamiltonian cycles in arbitrary graphs is presented. The method is applied to grid graphs, king's graphs, triangular grids, and three-dimensional grid graphs, and results are obtained for larger cases than previously published. The approach can easily be modified to enumerate hamiltonian paths and other similar structures.

Keywords: hamiltonian cycles; enumeration; dynamic programming

1 Introduction

A *hamiltonian cycle* in a graph G is a cycle that visits each vertex exactly once. Hamiltonian cycles have been studied extensively in graph theory [21, Section 7]; popular research topics include existence and enumeration. The problem of determining whether a given graph contains a hamiltonian cycle and the problem of enumerating hamiltonian cycles are known to be NP-complete [3, p. 199] and #P-complete [20] respectively for general graphs.

Given a graph, it can be difficult to obtain a closed form solution to the problem of enumerating hamiltonian cycles. Many of the recent results on enumerating hamiltonian cycles rely on algorithmic approaches instead. Examples of such approaches include [5] and [7] which enumerate hamiltonian cycles in the n-cube (for $n \leqslant 6$) and in the $n \times n$ grid graph (for $n \leqslant 20$) respectively.

The algorithms used in [5, 7] are based on dynamic programming, where large computational problems are solved by breaking them down to small reoccurring subproblems.

[*]The work was supported in part by the Academy of Finland under Grant No. 132122, the GETA Graduate School, the Nokia Foundation, and the Finnish Foundation for Technology Promotion.

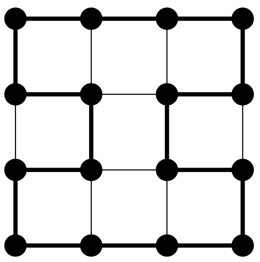

Figure 1: A hamiltonian cycle in the 4×4 grid graph

Dynamic programming has recently been used to enumerate other combinatorial structures as well, such as paths, latin squares, and perfect matchings [6, 10, 13]. Examples of other methods for enumerating hamiltonian cycles include backtrack search and ZDD trees by Knuth [8, pp. 254–255].

We present a method based on dynamic programming to enumerate hamiltonian cycles in arbitrary graphs. We apply the method to grid graphs, king's graphs, triangular grids, and three-dimensional grid graphs, and obtain results for larger cases than previously published. Notable sources of ideas for our method include [5], which considers enumerating hamiltonian cycles in bipartite graphs, and [6], which considers enumerating certain paths in the grid graph.

Some of the progress in computational methods for enumerating hamiltonian cycles can be illustrated conveniently using the the $n \times n$ grid graph as an example. Hamiltonian cycles in the $n \times n$ grid graph have received a lot of research attention over the years, partly due to applications in physics. The *grid graph* is the cartesian product of paths

$$P_{n_1} \times P_{n_2} \times \cdots \times P_{n_k},$$

and the $n \times n$ grid graph is the special case $P_n \times P_n$. Figure 1 shows a hamiltonian cycle in the 4×4 grid graph.

One early paper on enumerating hamiltonian cycles in the $n \times n$ grid graph is [16] from 1984, where the authors use a transfer matrix method to obtain the values for $n \leqslant 10$. Several articles were published in the following years, in particular on determining exact formulas for $n \times m$ grids with fixed small n [19]. However, for square grids the next case $n = 12$ was resolved as late as 2006 [11] along with $n = 14$. Soon after, in 2007 the cases $n \leqslant 20$ were resolved [7]. This large jump was made possible not only by improvements in computing hardware, but by using a more advanced dynamic programming method. In 2010 the case $n = 22$ was resolved [15]. Compared with [7], the method used in [15] contains a few modifications that trade running time for memory requirements. Unfortunately, this tradeoff makes the method too slow to be practical for larger cases. In the current work we compute the cases up to $n = 26$ by introducing several improvements to the method, all of which are generalized to arbitrary graphs.

Our method can be easily modified to solve the existence problem of hamiltonian cycles. It may be particularly useful for graphs which are almost nonhamiltonian in the sense that normal constructive methods do not find a cycle in reasonable time.

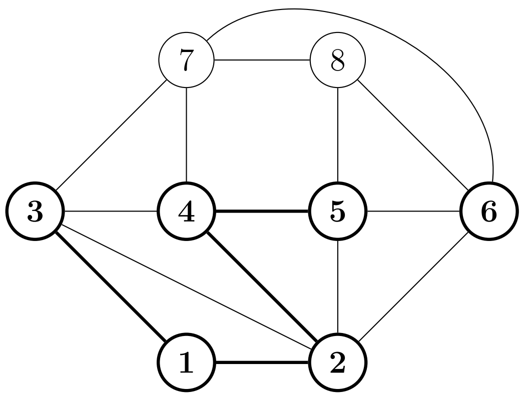

Figure 2: A partial hamiltonian cycle

In the current work we focus on hamiltonian cycles. It should be noted that there are many structures that are closely related to hamiltonian cycles, and some of the computational methods that work for hamiltonian cycles can be applied to these closely related structures with few modifications. A *hamiltonian path* in a graph G is a path that visits each vertex exactly once. A *bent hamiltonian path* is a hamiltonian path in the grid graph that changes direction at each step [12].

The article is organized as follows. In Section 2 we present our method for enumerating hamiltonian cycles. In Section 3 we describe the graphs to which we apply the method, and finally, in Section 4 we show the results for each graph and discuss how to gain confidence in the the correctness of the results.

2 Enumerating Hamiltonian Cycles

The method presented in the current work builds hamiltonian cycles by extending incomplete cycles one vertex at a time. With this in mind, we begin with some definitions. Let G be a graph and H a hamiltonian cycle in G.

Definition 1. For any $U \subset V(G)$, the *border* of U, denoted by $B_G(U)$, is the set of vertices in U that are adjacent to a vertex in $V(G) \setminus U$.

The subgraph of H induced by U, denoted by $H(U)$, is a collection of (possibly trivial) paths, the endpoints of which are in $B_G(U)$.

Definition 2. For $U \subset V(G)$, a *partial hamiltonian cycle on* U is a collection of (possibly trivial) paths in the subgraph of G induced by U such that every vertex in U is contained in exactly one path and every endpoint of a path is in $B_G(U)$.

Figure 2 illustrates these definitions. Vertices belonging to U are in bold, and edges belonging to the partial hamiltonian cycle C are thick. Here

$$U = \{1, 2, 3, 4, 5, 6\},$$

and the border of U is

$$B_G(U) = \{3, 4, 5, 6\}.$$

One way to complete the partial hamiltonian cycle in Figure 2 is to add the edges between vertices 3 and 7, vertices 7 and 6, vertices 6 and 8, and finally vertices 8 and 5. Note that a partial hamiltonian cycle as defined in Definition 2 is not necessarily a subgraph of any complete hamiltonian cycle.

Definition 3. For $U \subset V(G)$, a *border structure on* U is a partition of a subset of $B_G(U)$ such that each part contains one or two vertices.

Definition 4. For $U \subset V(G)$, $\mathcal{S}_G(U)$ is the set of all possible border structures on U.

We denote the set of endpoints of a path p by $e(p)$. If p is a trivial path, $e(p)$ contains just one vertex.

Definition 5. For $U \subset V(G)$ and a partial hamiltonian cycle C on U, $S(C)$ is the border structure $\{e(p) : p \text{ is a maximal path in } C\}$.

Note that $S(C) \in \mathcal{S}_G(U)$. Intuitively, $S(C)$ is the border structure that corresponds to C. In Figure 2, we have

$$S(C) = \{\{3, 5\}, \{6\}\}.$$

Similar definitions appear under many different names in the literature concerning dynamic programming algorithms for graphs. For example, [6] uses the terms frontier and frontier state that correspond to border and border structure respectively.

Definition 6. For $s \in \mathcal{S}_G(U)$, the *function representation of* s is the function $f : B_G(U) \to B \cup \{\infty\}$ such that $f(b) = b$ if $\{b\} \in s$, $f(b) = a$ if $\{a, b\} \in s$, and $f(b) = \infty$ otherwise.

Definition 7. For $s \in \mathcal{S}_G(U)$, and $\pi = (\pi_0, \pi_1, \dots, \pi_{m-1})$ the vertices in $B_G(U)$ in some order, and f the function representation of s, the *string representation of* s *ordered by* π is the string $(\pi^{-1}f(\pi_0), \pi^{-1}f(\pi_1), \dots, \pi^{-1}f(\pi_{m-1}))$, where $\pi^{-1}v$ is the index of v in π if $v \neq \infty$, and ∞ otherwise.

In Figure 2 the function representation of $S(C)$ has values

$$f(3) = 5, f(4) = \infty, f(5) = 3, f(6) = 6,$$

and the string representation of $S(C)$ ordered by, say, $(3, 4, 5, 6)$, is

$$(2, \infty, 0, 3).$$

Definition 8. For $U \subset V(G)$, a partial hamiltonian cycle C on U, and $v \in V(G) \setminus U$, the *extension of* C *with* v, $N'(C, v)$ is the set of all a partial hamiltonian cycles on $U \cup v$ that can be obtained by adding such edges to C that are incident to v and a vertex in U.

Definition 9. For $U \subset V(G)$, a partial hamiltonian cycle C on U, $s = S(C)$, and $v \in V(G) \setminus U$, the *extension of s with v*, $N(s,v)$, is the multiset $\{S(C') : C' \in N'(C,v)\}$, where the multiplicity of $s' \in N(s,v)$ is the number of different $C' \in N'(C,v)$ for which $s' = S(C')$.

Intuitively, $N(s,v)$ is the set of border structures that can be obtained from s by adding edges between v and U. For brevity of definition, in Definition 9 we define the extension only for border structures that have a corresponding partial hamiltonian cycle. It would be possible to define the extension analogously for all border structures, but this is not necessary for our purposes.

In Figure 2 the extension of $S(C)$ with, say, vertex 7, is the multiset

$$N(S(C),7) = \{\{\{7,5\},\{6\}\},\{\{5,6\}\}\},$$

where the multiplicity is 1 for both border structures.

Definition 10. For $U \subset V(G)$, the *set of all hamiltonian border structures on U*, $\mathcal{H}_G(U)$, is $\{S(U(\Gamma)) : \Gamma \text{ is a hamiltonian cycle on } G\}$.

Note that usually $\mathcal{H}_G(U) \neq \mathcal{S}_G(U)$.

Definition 11. For $\mathcal{S} \subset \mathcal{S}_G(U)$, $s \in \mathcal{S}$, and some ordering $(\mathcal{S}, <)$, the *rank of s in $\mathcal{S}$*, $\operatorname{rank}(s, \mathcal{S}, <)$, is $|\{s' \in \mathcal{S} : s' < s\}|$.

The method presented in the current work relies on the following observations. Given a graph G and a vertex $v \in V(G)$, the set of all hamiltonian cycles in G can be generated from the set of partial hamiltonian cycles on $V(G) \setminus v$ by adding edges between v and $V(G) \setminus v$ in all possible ways. These partial hamiltonian cycles can in turn be generated by adding edges to smaller partial hamiltonian cycles. Furthermore, when determining how edges can be added between a vertex and a partial hamiltonian cycle, the internal structure of the cycle is irrelevant and only the border structure matters. These observations lead to the approach shown in Algorithm 1.

In Algorithm 1, we iterate over the vertices of G one by one, and at every point we keep track of all the partial hamiltonian cycles on the iterated vertices. These partial hamiltonian cycles are grouped based on their border structure, in the sense that we do not store the individual partial cycles in memory, we only store their total number for each border structure in the counters C_k.

The efficiency of the method depends on how each part is implemented. There are several alternatives for implementing different parts of the method, which are discussed in the following subsections.

2.1 Ordering vertices

The order of vertices v_i is crucial for the performance of the method. One possible criterion for choosing the order is that the maximum border size

$$\max_k |B_G(\bigcup_{i=1}^{k} v_i)|$$

Algorithm 1 Enumerate hamiltonian cycles in G

```
Input: Graph $G$, where $V(G) = \{v_1, v_2, \ldots, v_n\}$
Output: Returns the number of hamiltonian cycles in $G$
  $\mathcal{S}_0 \leftarrow \{\emptyset\}$
  $C_0(\emptyset) \leftarrow 1$
  for $k = 1, 2, \ldots, n-1$ do
    $U \leftarrow \bigcup_{i=1}^{k} v_i$
    $\mathcal{S}_k \leftarrow$ any subset of $\mathcal{S}_G(U)$ such that $\mathcal{H}_G(U) \subset S_k$
    Initialize counter $C_k : \mathcal{S}_k \to \mathbb{N}$ to zero
    for $s \in \mathcal{S}_{k-1}$ do
      for $s' \in N(s, v_k) \cap \mathcal{S}_k$ do
        $C_k(s') \leftarrow C_k(s') + C_{k-1}(s)$
      end for
    end for
  end for
  return $\sum_{s \in \mathcal{S}_{n-1}} C_{n-1}(s) \cdot$ (Number of ways to complete $s$)
```

should be as small as possible. The motivation behind this criterion is that the running time and memory requirements for one step of the method depend strongly on the size of the corresponding border. Finding such an order of vertices that minimizes the maximum border size is known as determining the vertex separation number, or equivalently pathwidth. Determining the pathwidth is NP-complete for general graphs, but many results exist for specific graph families [2, 4, 14].

Instead of extending partial hamiltonian cycles one vertex at a time, one could extend several vertices at a time. Some examples of this approach include [5, 15]. Extending several vertices at a time can help reduce the maximum border size. Additionally, it allows the use of symmetries in some cases. The downside is that the running time for the extension step depends strongly on the number of vertices being added at one time.

2.2 Storing counters

2.2.1 Data structure

In Algorithm 1, the values of the counters C_k need to be stored in memory. As done in [6], we store the values in a vector u, which has the advantage that we need not store representations of border structures. Vector u has size $|\mathcal{S}_k|$ and the index corresponding to a border structure s is

$$\mathrm{rank}(s, \mathcal{S}_k, <),$$

for some (arbitrary) ordering $<$. Furthermore, various compression techniques can be used for storing u.

The choice of $\mathcal{S}_k$ is important from the point of view of running time and memory efficiency. We will discuss how $\mathcal{S}_k$ can be chosen for various classes of graphs in Section 2.3.

Here we described how the value of counters C_k can be stored in a vector. Another alternative would be storing the counters in a hash table that has representations of border structures as keys and the corresponding counters as values. This works reasonably well in many cases. One example of this approach is [7].

2.2.2 Chinese remainder theorem

The memory efficiency of our method can be improved by computing the number of hamiltonian cycles modulo a prime power for several small prime powers, i.e., prime numbers raised to a positive integer power. The algorithm needs to be run several times, once for each prime power. The total number of hamiltonian cycles can then be reconstructed with the Chinese remainder theorem. This results in a tradeoff, as memory requirements will be roughly inversely proportional to running time. A similar approach was used in [7].

Note that we can store the values of counters C_k modulo a prime power throughout the algorithm, because the only operations performed on the counters are multiplication and addition.

The least common multiple of prime powers of size at most 256 is approximately $1.7 \cdot 10^{109}$, which is larger than the number of hamiltonian cycles in the graphs considered in this article, so here only one byte is needed to store the value of a counter. We do not use all prime powers up to 256 for each graph, we only use as many as necessary to accurately reconstruct the total number of hamiltonian cycles. To determine how many prime powers we need for a graph, it is sufficient that the least common multiple of the prime powers is larger than an upper bound for the number of hamiltonian cycles in the graph.

2.3 Choosing $\mathcal{S}_k$

In Algorithm 1, we use $\mathcal{S}_k$ as the set of border structures for which we store the values of counters. Recall that in the algorithm the value of $\mathcal{S}_k$ is any set of border structures such that

$$\mathcal{H}_G(U) \subset \mathcal{S}_k \subset \mathcal{S}_G(U).$$

The choice of $\mathcal{S}_k$ is important from the point of view of performance and memory efficiency. Since the running time and memory requirements of our method are proportional to $|\mathcal{S}_k|$, we would like $|\mathcal{S}_k|$ to be as small as possible. The only limitations on the choice of $\mathcal{S}_k$ are that $\mathcal{S}_k$ should contain $\mathcal{H}_G(U)$ and we should be able to compute

$$\operatorname{rank}(s, \mathcal{S}_k, <)$$

for any $s \in \mathcal{S}_k$, for some ordering $<$.

Those border structures in $\mathcal{S}_k$ that are not in $\mathcal{H}_G(U)$ are unnecessary for enumerating hamiltonian cycles, since they cannot be extended to any complete hamiltonian cycle. Ideally, we would want $\mathcal{S}_k = \mathcal{H}_G(U)$, but $\mathcal{H}_G(U)$ is difficult to determine. Instead, we approximate $\mathcal{H}_G(U)$ as accurately as we can for each graph family, so that $\mathcal{S}_k$ is as small as possible while still containing $\mathcal{H}_G(U)$.

For general graphs, we can simply choose $\mathcal{S}_k = \mathcal{S}_G(U)$. In the following subsections we show possible choices of $\mathcal{S}_k$ for planar graphs and grid graphs.

2.3.1 Planar graphs

Consider a planar graph G, with $U \subset V(G)$, and C a partial hamiltonian cycle on U. Assume there is a planar embedding of G such that the vertices in $B_G(U)$ are on a line, the vertices in $V(G) \setminus U$ are above the line, and all other vertices are below the line. This is the case in Figure 2. Now the paths in C cannot cross in this embedding. Let the order of vertices along the line be

$$\pi = (\pi_1, \pi_2, \dots, \pi_m).$$

The condition of paths not crossing is equivalent to the property that the endpoints of paths in the string representation of s ordered by π be properly nested. Thus, we can choose $\mathcal{S}_k = \mathcal{S}_G^p(U)$, where $\mathcal{S}_G^p(U)$ is defined as follows. Here v' is the index of vertex v in π.

$$\mathcal{S}_G^p(U) = \{s \in \mathcal{S}_G(U) : a_1' < b_1' < a_2' \Rightarrow a_1' < b_2' < a_2', \forall \{a_1, a_2\}, \{b_1, b_2\} \in s\}$$

The string representations of the border structures in $\mathcal{S}_G^p(U)$ ordered by π are equivalent to 2-colored Motzkin words [17], which are strings with the character set

$$\{\alpha_1, \alpha_2, \text{opening bracket `('}, \text{closing bracket `)'}\},$$

and the property that the brackets be properly nested. The string representation of a border structure can be mapped to the corresponding 2-colored Motzkin word by replacing endpoints of paths with opening and closing brackets, vertices forming a trivial path with α_1, and vertices with degree two with α_2.

2.3.2 Grid graphs

The $n \times n$ grid graph is planar, but it has some additional structure that allows us to choose an $\mathcal{S}_k$ smaller than $\mathcal{S}_G^p(U)$. Given a border structure s, the following theorem gives a sufficient condition for $s \notin \mathcal{H}_G(U)$.

Theorem 12. *A border structure s on U, where a vertex v with $|\{\{v, a\} \in E(G) : a \in V(G) \setminus U\}| \leqslant 1$ forms a trivial path, cannot be extended to a complete cycle.*

Proof. Assume that C is a partial hamiltonian cycle such that $S(C) = s$, and assume that C can be extended to a complete hamiltonian cycle Γ.

The degree of v in Γ is $|\{\{v, a\} \in E(\Gamma)\}|$, which can be divided into two parts $|\{\{v, a\} \in E(\Gamma) : a \in U\}| + |\{\{v, a\} \in E(\Gamma) : a \in V(G) \setminus U\}|$. Since v forms a trivial path in s, we have $|\{\{v, a\} \in E(\Gamma) : a \in U\}| = |\{\{v, a\} \in E(C) : a \in U\}| = 0$. Furthermore, $|\{\{v, a\} \in E(\Gamma) : a \in V(G) \setminus U\}| \leqslant |\{\{v, a\} \in E(G) : a \in V(G) \setminus U\}| \leqslant 1$. By combining these two results we obtain that the degree of v in Γ is at most 1, a contradiction. □

Some of the borders $B_G(U)$ we use in the computations for grid graphs have the property that $|\{\{v,a\} \in E(G) : a \in V(G) \setminus U\}| \leqslant 1$ holds for every vertex $v \in B_G(U)$. For such borders, we can choose $\mathcal{S}_k = \mathcal{S}^g_G(U)$, where $\mathcal{S}^g_G(U)$ is defined as follows.

$$\mathcal{S}^g_G(U) = \{s \in \mathcal{S}^p_G(U) : s \text{ contains no trivial paths}\}$$

Other borders $B_G(U)$ we use in the computations for grid graphs have the property that $|\{\{v,a\} \in E(G) : a \in V(G) \setminus U\}| \leqslant 1$ holds for every vertex $v \in B_G(U)$ except one, v_1. For these borders, we can choose $\mathcal{S}_k = \mathcal{S}^{g,v_1}_G(U)$, where $\mathcal{S}^{g,v_1}_G(U)$ is defined as follows.

$$\mathcal{S}^{g,v_1}_G(U) = \{s \in \mathcal{S}^p_G(U) : s \text{ contains no trivial paths other than } \{v_1\}\}$$

Let π be defined as in Section 2.3.1. The string representation of $\mathcal{S}^g_G(U)$ ordered by π corresponds to Motzkin words [1], which are strings with the character set

$$\{\alpha_1, \text{opening bracket `('}, \text{closing bracket `)'}\},$$

and the property that the brackets be properly nested.

2.3.3 The size of $\mathcal{S}_k$

To understand the memory requirements of our method, it is useful to know $|\mathcal{S}_k|$ for each of the four choices of $\mathcal{S}_k$ discussed in the previous subsections, namely, $\mathcal{S}_G(U), \mathcal{S}^p_G(U), \mathcal{S}^g_G(U)$, and $\mathcal{S}^{g,v_1}_G(U)$. The following theorem gives $|\mathcal{S}_G(U)|$.

Theorem 13. *For a graph G and $U \subset V(G)$, we have $|\mathcal{S}_G(U)| = T_{|B_G(U)|}$, where $T_0 = 1, T_1 = 2$, and $T_n = 2T_{n-1} + (n-1)T_{n-2}$ for $n \geqslant 2$.*

Proof. For any border structure in $\mathcal{S}_G(U)$, the corresponding function representation f fulfills the following property: (i) $f(f(v)) = v$ for every such $v \in B_G(U)$ where $f(v) \neq \infty$. Conversely, every function with property (i) corresponds to a unique border structure on $B_G(U)$. Thus, $|\mathcal{S}_G(U)|$ is the same as the number of functions $f : B_G(U) \to B_G(U) \cup \{\infty\}$ with property (i). This number depends only on the size of $B_G(U)$, and is denoted by $T_{|B_G(U)|}$.

Clearly, $T_0 = 1$ and $T_1 = 2$. Assume $|B_G(U)| = n \geqslant 2$, and choose some $v \in B_G(U)$. If $f(v) \in \{\infty, v\}$, then f restricted on $B_G(U) \setminus v$ is a function that fulfills (i), so the number of functions where $f(v) \in \{\infty, v\}$ is $2T_{n-1}$. If $f(v) \notin \{\infty, v\}$, then f restricted on $B_G(U) \setminus \{v, f(v)\}$ is again a function that fulfills (i). The number of possible choices of $f(v) \notin \{\infty, v\}$ is $n-1$, so the number of functions where $f(v) \notin \{\infty, v\}$ is $(n-1)T_{n-2}$. The result follows by taking the sum of both cases. $\square$

Denote $n = |B_G(U)|$. Now $|\mathcal{S}^p_G(U)|$ is the number of 2-colored Motzkin words of length n, which is known to be the $(n+1)$th Catalan number C_{n+1} [17]. Similarly, $|\mathcal{S}^g_G(U)|$ is the number of Motzkin words of length n, i.e., the nth Motzkin number M_n [1]. Lastly, $|\mathcal{S}^{g,v_1}_G(U)| = M_{n-1} + M_n$, which can be easily seen by dividing $\mathcal{S}^{g,v_1}_G(U)$ in two parts

$$\mathcal{S}^{g,v_1}_G(U) = \{s \in \mathcal{S}^{g,v_1}_G(U) : \{v_1\} \in s\} \cup \{s \in \mathcal{S}^{g,v_1}_G(U) : \{v_1\} \notin s\}.$$

The size of the first part is M_{n-1} and the size of the second part is M_n.

In the Open Encyclopedia of Integer Sequences [18], T_n, C_n, and M_n are the sequences A005425, A000108, and A001006 respectively. Table 1 shows the approximate sizes of $\mathcal{S}_G(U), \mathcal{S}_G^p(U), \mathcal{S}_G^g(U)$, and $\mathcal{S}_G^{g,v_1}(U)$ for small n.

Table 1: Approximate size of $\mathcal{S}_k$

n	$\lvert\mathcal{S}_G(U)\rvert$	$\lvert\mathcal{S}_G^p(U)\rvert$	$\lvert\mathcal{S}_G^g(U)\rvert$	$\lvert\mathcal{S}_G^{g,v_1}(U)\rvert$
10	$1.2 \cdot 10^5$	$5.9 \cdot 10^4$	$2.2 \cdot 10^3$	$3.0 \cdot 10^3$
15	$2.7 \cdot 10^8$	$3.5 \cdot 10^7$	$3.1 \cdot 10^5$	$4.2 \cdot 10^5$
16	$1.3 \cdot 10^9$	$1.3 \cdot 10^8$	$8.5 \cdot 10^5$	$1.2 \cdot 10^6$
17	$7.0 \cdot 10^9$	$4.8 \cdot 10^8$	$2.4 \cdot 10^6$	$3.2 \cdot 10^6$
18	$3.7 \cdot 10^{10}$	$1.8 \cdot 10^9$	$6.5 \cdot 10^6$	$8.9 \cdot 10^6$
19	$2.0 \cdot 10^{11}$	$6.6 \cdot 10^9$	$1.8 \cdot 10^7$	$2.5 \cdot 10^7$
20	$1.1 \cdot 10^{12}$	$2.4 \cdot 10^{10}$	$5.1 \cdot 10^7$	$6.9 \cdot 10^7$
21	$6.2 \cdot 10^{12}$	$9.1 \cdot 10^{10}$	$1.4 \cdot 10^8$	$1.9 \cdot 10^8$
22	$3.5 \cdot 10^{13}$	$3.4 \cdot 10^{11}$	$4.0 \cdot 10^8$	$5.4 \cdot 10^8$
23	$2.1 \cdot 10^{14}$	$1.3 \cdot 10^{12}$	$1.1 \cdot 10^9$	$1.5 \cdot 10^9$
24	$1.2 \cdot 10^{15}$	$4.9 \cdot 10^{12}$	$3.2 \cdot 10^9$	$4.3 \cdot 10^9$
25	$7.4 \cdot 10^{15}$	$1.8 \cdot 10^{13}$	$9.0 \cdot 10^9$	$1.2 \cdot 10^{10}$
26	$4.6 \cdot 10^{16}$	$7.0 \cdot 10^{13}$	$2.6 \cdot 10^{10}$	$3.5 \cdot 10^{10}$
27	$2.8 \cdot 10^{17}$	$2.6 \cdot 10^{14}$	$7.3 \cdot 10^{10}$	$9.9 \cdot 10^{10}$

It turns out that in the computations for the king's graph almost all border structures in $\mathcal{S}_G(U)$ are encountered at any given step of the algorithm. In the computations for the grid graph, approximately half of the border structures in $\mathcal{S}_G^{g,v_1}(U)$ are encountered.

2.4 Computing rank

2.4.1 General graphs

We can compute the rank of border structures in $\mathcal{S}_G(U)$ with standard methods as follows.

We order the border structures lexicographically based on their string representation ordered by π, for some arbitrary π. Denote the size of the border $B_G(U)$ by n. Consequently, the length of each string representation is n. Consider some $s \in S_G(U)$, and let s' be the string representation of s ordered by π. It is easy to see that the lexicographic rank of s is $\text{rank}(s, S_G(U), <) = \sum_{i=0}^{n-1} F(i, s')$, where $F(i, s')$ is the number of border structures in $S_G(U)$ whose string agrees with s' on indices smaller than i and is strictly smaller on index i.

The value of $F(i, s')$ can in turn be computed as follows. We call the indices from 0 to $i-1$ the left part of the string, and from $i+1$ to $n-1$ the right part of the string. Let N be the number of indices in the right part of the string for which the value is already fixed in the left part, i.e., the number of indices k such that $k > i > s'(k)$. Let L be the number of values that could be at index i in a lexicographically smaller string without conflicting with the left part of s', i.e., the number of indices k such that $i \leqslant k < s'(i)$ and $s'(k) > i$. Note that if the values at m different indices are fixed in a string representation, then there are T_{n-m} ways to fill the remaining values. We now have

$$F(i, s') = \begin{cases} 0 & \text{if } s'(i) < i+1 \\ T_{n-i-N-1} & \text{if } s'(i) = i+1 \\ T_{n-i-N-1} + T_{n-i-N-2}(L-1) & \text{if } s'(i) > i+1. \end{cases}$$

Unranking can be done by performing the above operations in reverse one index at a time.

2.4.2 Planar graphs and grid graphs

Computing the rank of border structures in $\mathcal{S}_G^g(U)$ corresponds to computing the rank of Motzkin words. One method for this is shown in [9]. The same method can be used with minor modifications for border structures in $\mathcal{S}_G^p(U)$.

The rank of border structures in $\mathcal{S}_G^{g,v_1}(U)$ can be computed as follows. We divide $\mathcal{S}_G^{g,v_1}(U)$ in two parts,

$$\mathcal{S}_G^{g,v_1}(U) = \{s \in \mathcal{S}_G^{g,v_1}(U) : \{v_1\} \in s\} \cup \{s \in \mathcal{S}_G^{g,v_1}(U) : \{v_1\} \notin s\}.$$

Note that $\{s \in \mathcal{S}_G^{g,v_1}(U) : \{v_1\} \notin s\} = \mathcal{S}_G^g(U)$. The rank is now defined as

$$\operatorname{rank}(s, \mathcal{S}_G^{g,v_1}(U), <'') = \begin{cases} \operatorname{rank}(s, \mathcal{S}_G^g(U), <') & \text{if } \{v_1\} \notin s \\ \operatorname{rank}(s \setminus \{v_1\}, \mathcal{S}_G^g(U \setminus v_1), <') + M_n & \text{otherwise,} \end{cases}$$

where $<'$ is the ordering used in the method in [9].

3 Graphs

We apply our enumeration method to the following graph families. The $n \times m$ *grid graph* is the cartesian product of two paths, $P_n \times P_m$, and the *three-dimensional grid graph* is the cartesian product of three paths $P_n \times P_m \times P_k$. The $n \times n$ *king's graph* is the strong product of two paths of length n; its vertices correspond to squares on an $n \times n$ chessboard and the edges correspond to to king's moves on the chessboard. In the *triangular grid* the set of vertices is $\{(i, j) : 1 \leqslant j \leqslant i \leqslant n\}$, and the set of edges is

$$\{\{(i, j), (i', j')\} : (i - i', j - j') \in \{(0, 1), (1, 0), (1, 1)\}\}.$$

Figure 3 shows the grid graph, the king's graph and the triangular grid for $n = 4$.

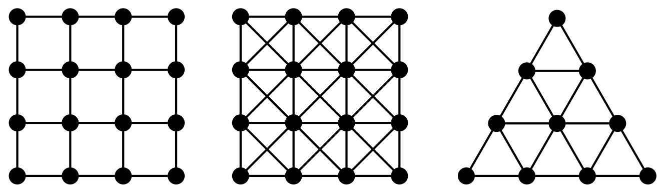

Figure 3: The grid graph, the king's graph, and the triangular grid

The number of hamiltonian cycles in an $n \times n$ grid is sequence A003763 in the Open Encyclopedia of Integer Sequences [18]. Note that this sequence only shows the values for even n, since the number of hamiltonian cycles in the $n \times n$ grid graph is zero for odd n. For $n \times n$ king's graphs and triangular grids with n vertices on one side the numbers of hamiltonian cycles are in sequences A140519 and A112676 respectively.

In the dynamic programming method, we order the vertices of these graphs as follows. For the grid graph and the king's graph we order the vertices lexicographically, i.e., from left to right, and from top to bottom. For the triangular grid we use reverse lexicographic ordering. For the three-dimensional grid graph we order the vertices based on the distance to one of the corners, with ties broken lexicographically.

4 Results

Results for all the graph families described in Section 3 are shown in Tables 2 to 5, and new results are denoted with a star. In all of the cases studied here the computations were bounded more by memory than CPU-time. Enumerating hamiltonian cycles in the grid graph for $n = 26$ required the most resources, approximately 3 core-years of CPU-time and 160 gigabytes of memory using four bytes per counter.

The following two observations give some confidence in the correctness of the results and the implementation. First, the results agree with previously published work for small parameter values. Second, the total number of hamiltonian cycles given by the Chinese

Table 2: Hamiltonian cycles in $n \times n$ grid graphs

n	N
2	1
4	6
6	1 072
8	4 638 576
10	467 260 456 608
12	1 076 226 888 605 605 706
14	56 126 499 620 491 437 281 263 608
16	65 882 516 522 625 836 326 159 786 165 530 572
18	1 733 926 377 888 966 183 927 790 794 055 670 829 347 983 946
20	1 020 460 427 390 768 793 543 026 965 678 152 831 571 073 052 662 428 097 106
22	13 404 209 505 893 761 748 339 786 653 564 937 498 745 897 123 531 041 248 680 272 448 954 956
24*	3 924 231 694 060 647 894 532 092 926 553 286 517 550 515 989 932 148 978 428 996 623 179 638 255 782 936 503 022
26*	25 578 285 385 897 276 060 130 031 526 614 700 187 075 412 685 764 186 583 833 403 069 393 167 252 132 218 312 152 073 569 856 334 502

remainder theorem always converges, in the sense that the final value remains the same after computing the number of cycles for a few more prime powers than necessary.

Table 3: Hamiltonian cycles in $n \times n$ king's graphs

n	N
1	1
2	3
3	16
4	2 830
5	2 462 064
6	22 853 860 116
7	1 622 043 117 414 624
8	961 742 089 476 282 321 684
9$*$	4 601 667 243 759 511 495 116 347 104
10$*$	179 517 749 570 891 592 016 479 828 267 003 018
11$*$	56 735 527 086 758 553 613 684 823 040 730 404 215 973 136
12$*$	145 328 824 470 156 271 670 635 015 466 987 199 469 360 063 082 789 418
13$*$	3 013 072 757 042 748 407 212 267 203 778 429 049 866 618 090 427 057 156 382 635 712
14$*$	505 396 541 863 296 313 964 793 910 305 382 425 060 965 154 779 449 831 170 884 147 484 924 489 066
15$*$	685 457 393 589 353 762 730 302 985 699 040 971 223 260 321 251 614 789 007 892 889 891 954 959 485 049 448 085 648
16$*$	7 514 427 561 614 895 453 501 809 269 193 245 238 210 545 618 759 874 171 463 008 264 116 894 117 263 163 499 356 026 497 440 057 866

References

[1] R. Donaghey, L.W. Shapiro: Motzkin numbers. J. of Combinatorial Theory, Series A, 23, 291–301 (1977)

[2] J. Ellis, R. Warren: Lower bounds on the pathwidth of some grid-like graphs. Discrete Applied Math., 156, 545–555 (2008)

[3] M. R. Garey, D. S. Johnson: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)

[4] R. Hochberg, C. McDiarmid, M. Saks: On the bandwidth of triangulated triangles. Discrete Math., 138, 261–265 (1995)

[5] H. Haanpää, P.R.J. Östergård: Counting Hamiltonian cycles in bipartite graphs. Math. Comp., 83, 979–995 (2014)

Table 4: Hamiltonian cycles in triangular grids with n vertices on one side

n	N
1	1
2	1
3	1
4	3
5	26
6	474
7	17 214
8	1 371 454
9$*$	231 924 780
10$*$	82 367 152 914
11$*$	61 718 801 166 402
12$*$	97 482 824 713 311 442
13$*$	323 896 536 556 067 453 466
14$*$	2 262 929 852 279 448 821 099 932
15$*$	33 231 590 982 432 936 619 392 054 662
16$*$	1 025 257 090 790 362 187 626 154 669 771 934
17$*$	66 429 726 878 393 651 076 826 663 971 376 589 034
18$*$	9 036 923 456 137 297 343 631 952 691 847 844 984 938 922
19$*$	2 580 487 118 457 573 944 114 883 930 049 209 668 222 079 445 692
20$*$	1 546 380 273 056 984 325 468 288 805 366 239 469 621 495 241 688 789 648

Table 5: Hamiltonian cycles in three-dimensional $n \times m \times k$ grid graphs

$n \times m \times k$	N	$n \times m \times k$	N
$2 \times 2 \times 2$	6	$3 \times 3 \times 3$	0
$2 \times 2 \times 3$	22	$3 \times 3 \times 4$	3 918 744
$2 \times 2 \times 4$	82	$3 \times 3 \times 5$	0
$2 \times 2 \times 5$	306	$3 \times 4 \times 4$	3 777 388 236
$2 \times 3 \times 3$	324	$3 \times 4 \times 5*$	3 180 229 736 508
$2 \times 3 \times 4^*$	4 580	$3 \times 5 \times 5$	0
$2 \times 3 \times 5^*$	64 558	$4 \times 4 \times 4*$	57 958 048 716 672
$2 \times 4 \times 4^*$	232 908	$4 \times 4 \times 5*$	857 931 886 492 226 164
$2 \times 4 \times 5^*$	11 636 834	$4 \times 5 \times 5*$	211 005 720 188 979 351 400 104
$2 \times 5 \times 5^*$	2 040 327 632	$5 \times 5 \times 5$	0

[6] H. Iwashita, Y. Nakazawa, J. Kawahara, T. Uno, S. Minato: Efficient Computation of the Number of Paths in a Grid Graph with Minimal Perfect Hash Functions. Tech. Report. TCS-TR-A-13-64, Division of Computer Science, Graduate School of Information Science and Technology, Hokkaido University (2013)

[7] J.L. Jacobsen: Exact enumeration of Hamiltonian circuits, walks and chains in two and three dimensions. J. Phys. A: Math. Theor. 40, 14667–14678 (2007)

[8] D. E. Knuth: The Art of Computer Programming, Volume 4A, Combinatorial Algorithms, Part 1. Addison-Wesley, Upper Saddle River (2011)

[9] J. Liebehenschel: Ranking and unranking of lexicographically ordered words: an average-case analysis. J. Autom. Lang. Comb., 2, 227–268 (1997)

[10] B.D. McKay, I.M. Wanless: On the number of Latin squares. Ann. Comb., 9, 335–344 (2005)

[11] R. Oberdorf, A. Ferguson, J.L. Jacobsen, J. Kondev: Secondary structures in long compact polymers. Physical Review E, 74, 051801 (2006)

[12] P.R.J. Östergård: On the nonexistence of bent Hamiltonian paths in the grid graph $P_3 \times P_5 \times P_5$. Bull. Inst. Combin. Appl., 42, 87–88 (2004)

[13] P.R.J. Östergård, V. Pettersson: Enumerating perfect matchings in n-Cubes. Order, 30, 821–835 (2013)

[14] Y. Otachi, R. Suda: Bandwidth and pathwidth of three-dimensional grids. Discrete Math., 311, 881–887 (2011)

[15] A.M. Karavaev: A dynamic programming method for counting the number of cycles in a rectangular lattice (in Russian), 2010. `http://habrahabr.ru/post/105705/`

[16] T.G. Schmalz, G.E. Hite, D.J. Klein: Compact self-avoiding circuits on two-dimensional lattices. J. Phys. A: Math. Gen., 17, 445–453 (1984)

[17] A. Sapounakis, P. Tsikouras: Counting peaks and valleys in k-colored Motzkin paths. Electron. J. Combin., 12, R16 (2005)

[18] N.J.A. Sloane: The On-Line Encyclopedia of Integer Sequences, 2012. `http://oeis.org`

[19] R. Stoyan, V. Strehl: Enumeration of hamiltonian circuits in rectangular grids. J. of Combinatorial Mathematics and Combinatorial Computing, 21, 109–128 (1996)

[20] L.G. Valiant: The complexity of enumeration and reliability problems. SIAM J. Comput., 8, 410–421 (1979)

[21] D.B. West: Introduction to Graph Theory, 2nd edition. Prentice Hall, Upper Saddle River (2001)