Grid Hamiltonicity

Project report

Lazaros Koromilas
koromil@csd.uoc.gr

Computer Science Department
University of Crete

May 27, 2011

1 Introduction

The infinite grid (G^∞) can be thought of as the graph whose vertex set consists of all points in the plane with integer coordinates; edges exist only between vertices whose Euclidean distance is equal to 1. A *grid graph* is a finite, node-induced subgraph of G^∞, thus is totally defined by its vertex set. A bounded face with exactly four edges on its boundary is called a *cell*; all other bounded faces are called *holes*. Other types of grids exist apart from the square grid described previously. For instance, triangular grids can be built by tiling equilateral triangles together; the same can be done with regular hexagons to form hexagonal grids and so on. For more information on general convex polygon grids see [2]. Throughout this document the terms "grid graph" and "grid" are used interchangeably, and refer to finite square grids.

Computations on grid graphs are very commonly used in the implementation of computer graphics engines, in computational geometry, robotics and more. In computer science, in general, the grid is used as a tool to approximate continuous domains with discrete point sets. Of course, the theoretical construct of grid graphs is also of great importance. In this work, the properties of grid graphs, as well as the Hamiltonian cycle and Hamiltonian path problems on grids are being explored.

Grid graphs have unique properties that probably make some of the hard problems easier to solve, and thus permit efficient algorithms. Firstly, they are *planar* and have *maximum degree 4* by design. Another property is that

they are *bipartite*. The latter can be seen by coloring vertices according to their coordinates: each vertex v has coordinates v_x, v_y whose sum is either even or odd, and all neighbors of v have different parity because they differ by 1 in only one of the coordinates.

A tour that visits each vertex of a graph only once is called a *Hamiltonian cycle*. Moreover, a graph that contains at least one such tour is called Hamiltonian. If two endpoints s, t are specified, then the problem of finding a path, beginning at s and finishing at t, that visits all vertices only once is called the *Hamiltonian path* problem. Despite the distinct nature of grids, the Hamiltonian cycle problem on general grid graphs has been proven to be NP-complete [4]. The Hamiltonian path, which is a very similar problem, is also NP-complete. Consider choosing a corner (degree 2) of the grid G as s and any of its neighbors as t. Then the problem of finding a Hamiltonian path from s to t in G becomes a matter of finding a Hamiltonian cycle in G.

A Hamiltonian cycle can also be seen as a special case of a *2-factor*. A 2-factor of a graph G is a subgraph of G with the same vertex set and a smaller edge set, such that all vertices have degree 2. Visually, this is a set of disjoint cycles of G, and every such cycle is a component of the 2-factor. Consequently, a 2-factor with only one component is by definition also a Hamiltonian cycle.

The study of grid hamiltonicity requires more restrictions to be put on the form of grid graphs in order for the problems to become easy. To this end, a classification of grids is necessary. The class of grid graphs that will be discussed here is known as *solid grid graphs* or *grid graphs without holes*. More formally, this means that given a grid G, both graphs G and $G^{\infty} - G$ are connected. Trivial cases of graphs containing a vertex of degree 1 are excluded, because it cannot possibly be part of a tour. Examples of a grid graph with and without holes are given in Figures 1 and 2 respectively.

Furthermore, the Hamiltonian cycle problem is closely related to the *Traveling Salesman* problem. The Traveling Salesman problem is the problem of identifying a Hamiltonian cycle of minimum total length, and it generally concerns graphs with weighted edges. The complexity of the Traveling Salesman problem in a solid grid graph is still an open problem, although the Hamiltonian cycle problem for this subclass of grids has been proven to be P-hard [6].

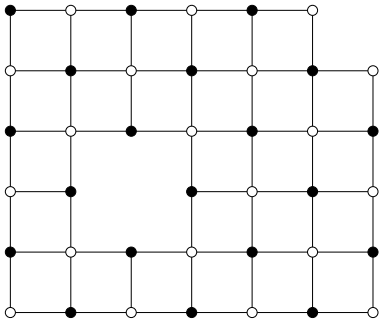

Figure 1: A grid graph with a hole.

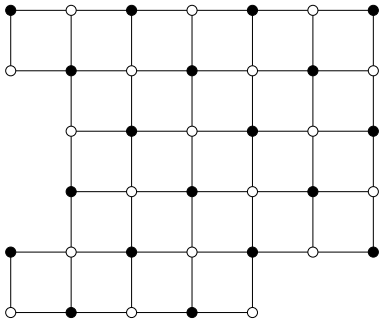

Figure 2: A solid grid graph.

2 Related work

In [4] a polynomial time algorithm for discovery of Hamiltonian paths in *rectangular* (see Figure 3) solid grid graphs is provided. They employ a *striping and splitting* technique which is essentially a divide and conquer solution that terminates when prime problems (those which cannot be stripped or split) are encountered. Prime problems either have known solutions or no solution at all. Rectangular solid grid graphs is of course a subclass of grid graphs without holes.

Another grid class is defined in [1] and can be described as solid grids with a vertical boundary, a horizontal boundary and a distorted *ladder* boundary. An example of this type of grids can be seen in Figure 4. A linear algorithm is given for computing Hamiltonian cycles in this class of grids, that obviously includes rectangular grids (a rectangular grid has a ladder in which all "steps" are of equal height). The main idea of the algorithm is to locate suitable horizontal and vertical *cuts*, in order to obtain solvable subproblems and glue their solutions together. The choice of cuts uses properties of the ladder and of graph balance, that is also defined for these grids.

A polynomial solution for hamiltonicity for a subclass of 3-dimensional grid graphs can be found in [3]. The class of graphs they studied is called *polycubes* and are constructed as layers of *simple polyminos*. A simple polymino is a bi-connected solid grid graph; an example of this type of grids is included in Figure 5. The authors even conjectured that deciding Hamiltonicity for solid grids is NP-complete.

The computation of Hamiltonian cycles in solid grids, is proven in [6] to have a polynomial solution. The algorithm first finds an initial 2-factor F of the grid G and works on the borders of the 2-factor's components. A border cell is a face that shares edges with many components. It is then possible

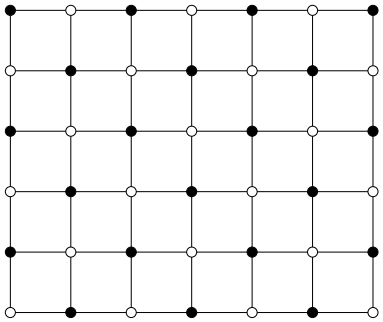

Figure 3: A rectangular grid.

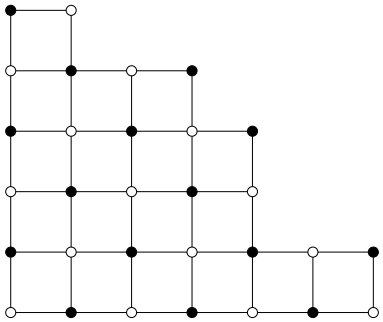

Figure 4: A grid graph with a ladder.

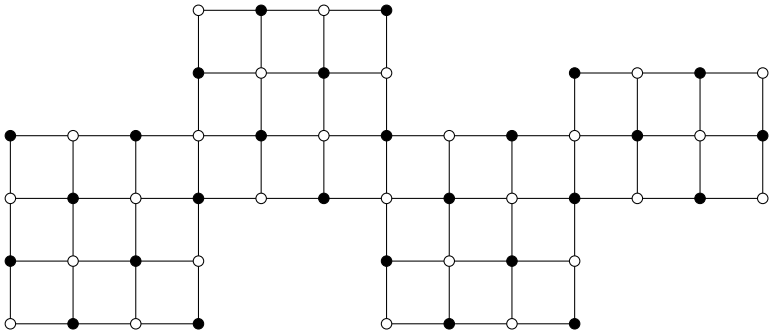

Figure 5: A simple polymino.

to employ *cycle merging* by identifying cell configurations and *flipping* their adjacent edges. As an example consider a border cell with exactly two non-adjacent edges be part of F. Then, by inverting the face's edges, two components of the 2-factor are merged (Figure 6).

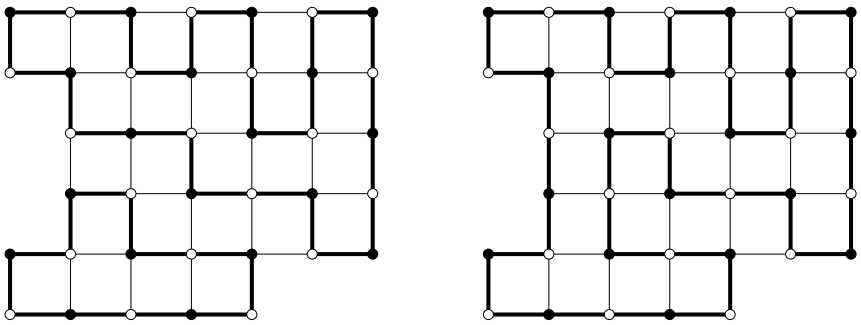

Figure 6: Merging of two cycles.

3 Algorithm implementation

In this project, the first step to finding Hamiltonian cycles in solid grid graphs, as well as a *visualizer* for grid graphs, have been implemented. More specifically, the algorithm studied finds a 2-factor in a grid, if one exists, using a reduction to matching. A *matching* is a vertex-disjoint subset of the edges chosen in such a way that no edges share a vertex. If all vertices are matched with another one, then the matching is called *perfect*.

An auxiliary graph is constructed from the grid graph using the following rules/steps: *(i)* two dummy vertices are added on every edge and *(ii)* a clone vertex is created for every original vertex, and this clone vertex is connected to the dummy vertices adjacent to the original. A *maximum matching* in the auxiliary graph gives a 2-factor in the original graph. If a maximum matching cannot be obtained, then there is no 2-factor in the grid and thus it cannot contain a Hamiltonian cycle. In other words, a grid graph has a 2-factor if and only if the auxiliary graph has a perfect matching. A proof of this can be found in [5]. The constructed graph for the grid of Figure 2 and a perfect matching (shown with bold lines) are displayed in Figure 7. Finally, to obtain the 2-factor (see Figure 6) only the edges between dummy nodes are considered and map to the original edges. If such an edge is included in the matching then it is excluded from the 2-factor. The algorithm runs in time $O(n \cdot m)$ which is equivalent to $O(n^2)$ because the degree of a grid is bounded.

The key function of the matching algorithm is based upon the notion of *augmenting paths*. An augmenting path is a path starting from an unmatched vertex, continuing with edges that alternate between being out and in the matching, and ending to an unmatched vertex again. While such paths are identified, their edges are inverted so that those already inside are

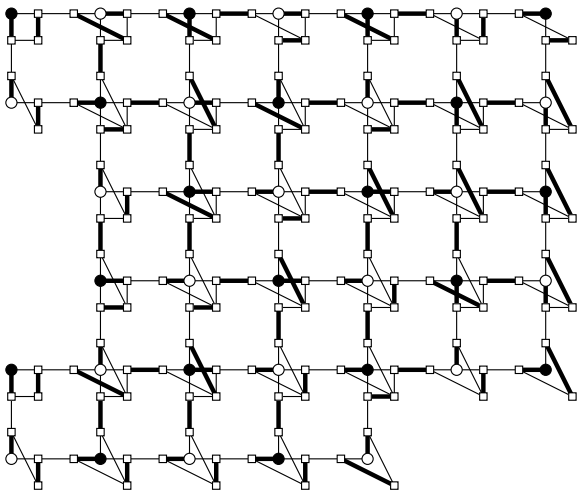

Figure 7: A perfect matching on the auxiliary graph.

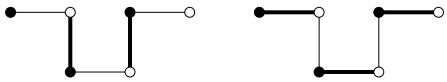

Figure 8: Augmenting of a path.

removed from —while the rest are added to— the matching (see Figure 8). Observe that the path after the augmentation contains one more edge that contributes to the matching. This is an approach of iterative improvement: the solution is improved step-by-step until there is no room for improvement. The resulting matching is guaranteed to be maximum.

The main idea of the algorithm described above is shown in Figure 9 (TwoFactor). In line 1 the auxiliary graph T is constructed and a maximum matching M is computed on this graph (line 2). Finally, at line 3 the 2-factor F is obtained by the set difference between the edges E of the original graph and the edges M that comprise the matching in the auxiliary graph. As stated previously, only the edges between the dummy nodes added at step *(i)* are comparable to the original edges of the graph.

The other algorithm of Figure 9 (HamiltonCycle) is the main procedure for discovering Hamiltonian cycles. If a 2-factor F cannot be obtained at line 1, the algorithm exits (line 2) as there is no possible Hamiltonian cycle in the specific graph G. The loop at lines 3–8 iterates over the components of F and tries to merge the respective cycles until only one component remains. This component will be a Hamiltonian cycle. At each iteration, all

HamiltonCycle(G)
Input: G is a grid without holes

```
1 F ← TwoFactor(G)
2 if not F then exit
3 while components > 1 do
4     LabelComponents(F)
5     S ← FindStripSequence(G, F)
6     if not S then exit
7     else ApplySequence(F, S)
8 end
```

TwoFactor(G)
Input: G is a grid graph

```
1 T ← ConstructAux(G)
2 M ← MaxMatching(T)
3 F ← E − M
```

Figure 9: The high-level algorithm for finding Hamiltonian cycles.

current components are identified/labeled (line 4) and the procedure FindStripSequence searches for a sequence S of *strips* that when flipped, the number of components in F will be reduced by one. These strips are series of cells with very specific configurations; for a detailed description of these configurations refer to [6]. If no such sequence exists the algorithm terminates resulting in a negative result for the existence of Hamiltonian cycle in G (line 6). If a sequence S is found, it is applied at line 7 and the loop continues. The time complexity of this algorithm is dominated by the running time of all-pairs-shortest-paths needed in FindStripSequence that is $O(n^3)$ with the generic algorithm and is repeated at most n times, so it has a total time complexity $O(n^4)$. A smarter procedure for finding such sequences would lower the time complexity and the total complexity could be affected only by the initial 2-factorization.

The development of the visualizer required some understanding of the *FIG format*[1] (Facility for Interactive Generation of figures). Consider an object-oriented programming environment, where the `Graph` class contains a `to_fig` method that reflects the state of the graph to a `.fig` file. This can also work as a snapshoting mechanism in order to record an algorithm's progress by hooking figure generation calls in its code. The produced figures can be exported in numerous vector and raster graphics formats using the `fig2dev` utility. The whole implementation consists of some hundred lines of Ruby code and several shell scripts for image exporting and video/animations generation. To better demonstrate the results of the TwoFactor algorithm and the visualizer, another example of a 2-factor for a larger solid grid graph is displayed at Figure 10.

[1] `http://www.xfig.org/userman/fig-format.html`

4 Conclusion

Several subclasses of grid graphs have been studied in this project, all with very specific morphology that enable efficient algorithms for the Hamiltonian cycle problem. For general grid graphs (grids *with* arbitrary holes) it is NP-hard to identify a Hamiltonian cycle. The inherent difficulty of grids containing holes, when it comes to algorithms' correctness, is a topological issue. The 2-factorization, for instance, does not require a solid grid, but the algorithm that performs cycle merging cannot work without the assumption of a solid grid. Intuitively, this is because, when applying a transformation, an otherwise successful cycle merging may now fail because of the borders of a hole. Similarly, for the divide and conquer algorithms proposed for rectangular grids and grids with a ladder, the solutions of the subproblems will not always have a proper join if holes are in-between. However, it is still an open problem if *some holes* can be allowed in specific regions of a grid for the Hamiltonian cycle problem to have an efficient polynomial solution.

References

[1] F.N. Afrati. The Hamilton Circuit problem on Grids. *Informatique Théorique et Applications (ITA)*, 28(6):567–582, 1994.

[2] E.M. Arkin, S.P. Fekete, K. Islam, H. Meijer, J.S.B. Mitchell, Y. Núņez-Rodríguez, V. Polishchuk, D. Rappaport, and H. Xiao. Not Being (Super) Thin or Solid is Hard: A Study of Grid Hamiltonicity. *Computational Geometry*, 42(6-7):582–605, 2009.

[3] C. Hwan-Gue and A. Zelikovsky. Spanning Closed Trail and Hamiltonian Cycle in Grid Graphs. In *Proceedings of the 6th international symposium on Algorithms and Computations (ISAAC), Cairns, Australia*, page 342. Springer Verlag, 1995.

[4] A. Itai, C.H. Papadimitriou, and J.L. Szwarcfiter. Hamilton Paths in Grid Graphs. *Society for Industrial and Applied Mathematics (SIAM) Journal on Computing*, 11:676, 1982.

[5] C. Umans. An Algorithm for Finding Hamiltonian Cycles in Grid Graphs Without Holes. Bachelor's Thesis, Williams College, 1996.

[6] C. Umans and W. Lenhart. Hamiltonian Cycles in Solid Grid Graphs. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 496–505. IEEE, 1997.

Appendix

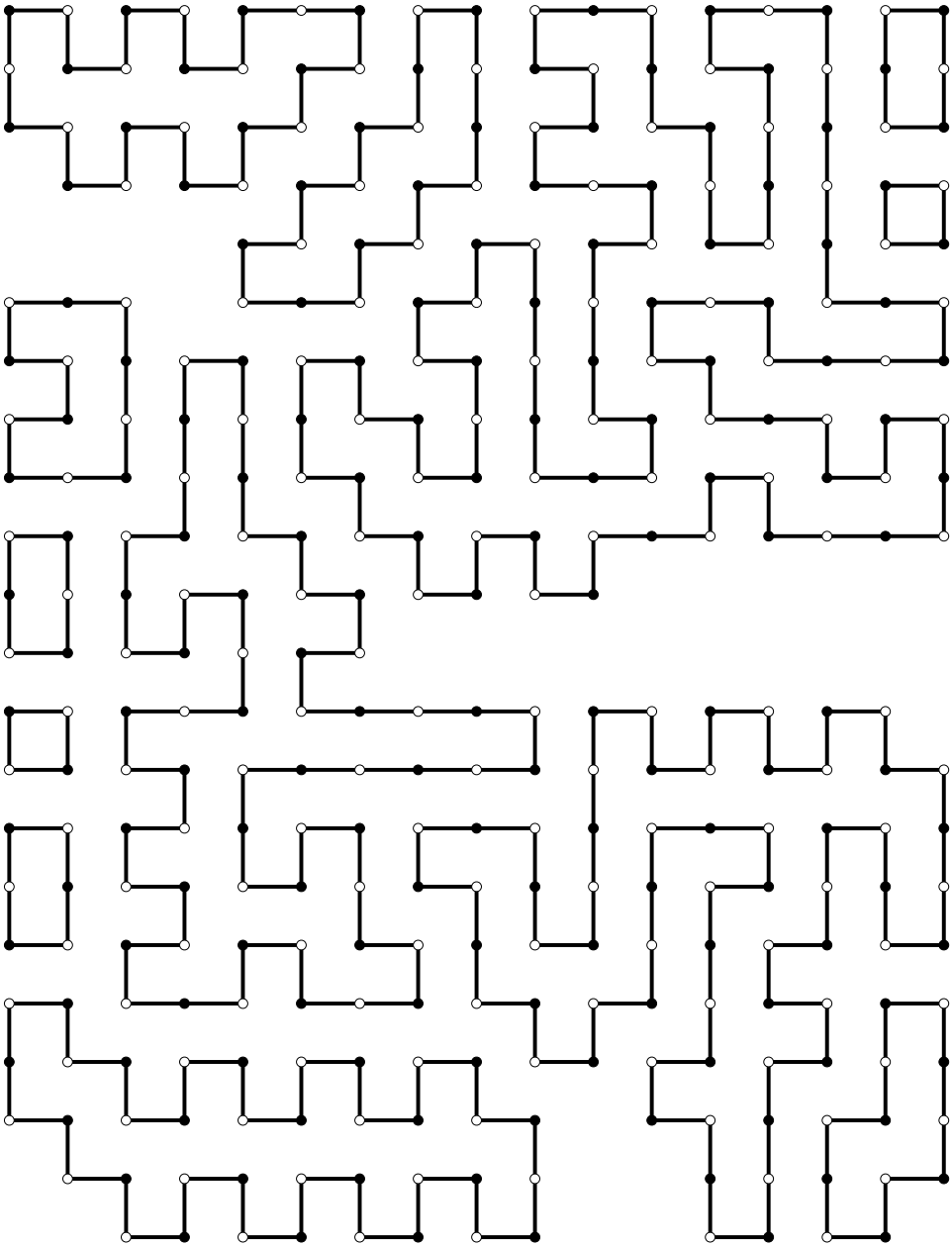

Figure 10: The 2-factor obtained for a larger grid graph.