Diplomatico – A Graph-based Game Analysis

Filippo Garagnani
199670

October 24, 2025

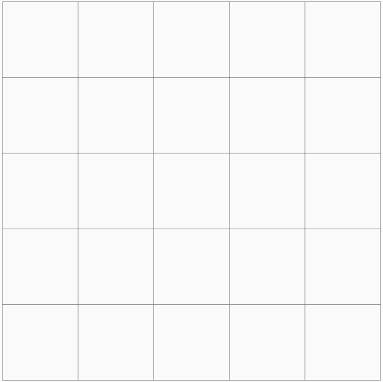
Tesina del corso Graph Analytics, Prof.ssa Laura Po

Outline

Introduction

The Game

The game is played on a $n \times m$ **Board**:



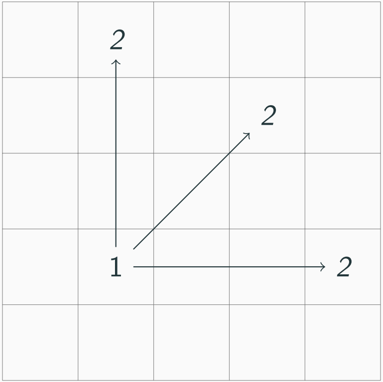
The Game

The player chooses a square (i, j) in which to start – writing a $\mathbf{1}$:

	$\mathbf{1}$			

The Game

Each turn, the player can write the next integer, either moving horizontally/vertically by three squares or moving diagonally by two squares:

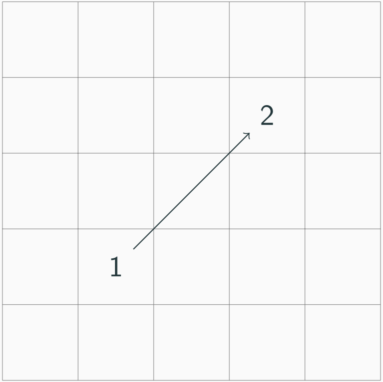


Goal of the game is to fill the whole grid:

	1			

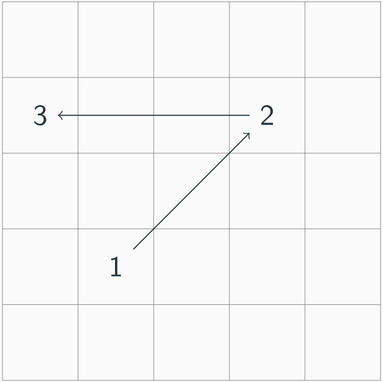
The Game

Goal of the game is to fill the whole grid:



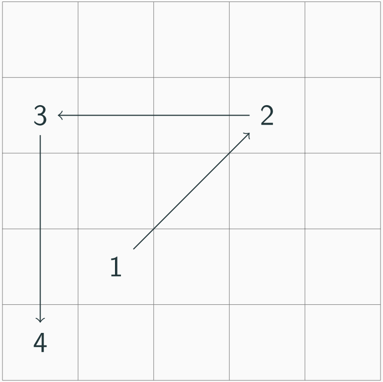
The Game

Goal of the game is to fill the whole grid:



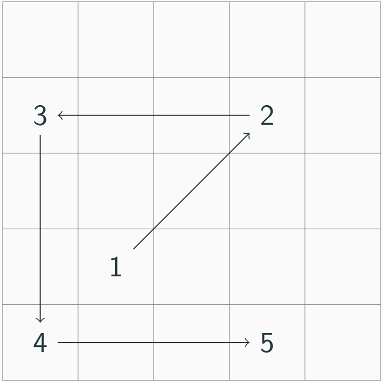
The Game

Goal of the game is to fill the whole grid:



The Game

Goal of the game is to fill the whole grid:



The Game as a Graph

The game can be represented by an undirected graph. Assuming the Board to be of size $n \times m$:

- **Nodes** – Represent the *cells* of the grid:

$$V = \{v_{(i,j)},\, i \in \mathbb{N}_n,\, j \in \mathbb{N}_m\}$$

The Game as a Graph

The game can be represented by an undirected graph. Assuming the Board to be of size $n \times m$:

- **Nodes** – Represent the *cells* of the grid:

$$V = \{v_{(i,j)},\ i \in \mathbb{N}_n,\ j \in \mathbb{N}_m\}$$

- **Edges** – Represent possible moves between cells:

$$E \subseteq \binom{V}{2} \quad \text{s.t.}$$

$$\begin{aligned}\forall e = \{v_{(i,j)},\ v_{(k,\ell)}\} \in E, \quad & (|i-k| = 3 \land j = \ell) && (\textit{vertical move})\\ & \lor (|j-\ell| = 3 \land i = k) && (\textit{horizontal move})\\ & \lor (|i-k| = |j-\ell| = 2) && (\textit{diagonal move})\end{aligned}$$

The Game as a Graph

A **solution** to the game is an **Hamiltonian Path** of the graph – a sequence of nodes that contains each vertex once and exactly once:

$$\mathfrak{h} = (v^1,\, v^2,\, \ldots,\, v^{n\times m})$$

$$\forall \ell \in \mathbb{N}_{n\times m},\ \left\{v^{(\ell)},\, v^{(\ell+1)}\right\} \in E \quad \text{(path)}$$

$$\forall v_{(i,j)} \in V,\ \exists! \ell \in \mathbb{N}_{n\times m} \text{ s.t. } v^{\ell} \equiv v_{(i,j)} \quad \text{(hamiltonian)}$$

Theoretical Results

Theoretical Results

Theorem (Solvability)

Each Graph representing a board of size $n \times m$ is solvable (i.e., the graph admits at least one hamiltonian path) iff:

$$m \geq n \geq 4 \land (n,\, m) \neq (4,\, 4)$$

Theoretical Results

Theorem (Solvability)

Each Graph representing a board of size $n \times m$ is solvable (i.e., the graph admits at least one hamiltonian path) iff:

$$m \geq n \geq 4 \wedge (n, m) \neq (4, 4)$$

Theorem (Number of Solutions)

Given a solvable graph $\mathfrak{G}_{n\times m}$ representing a grid of size $n \times m$, the number of solutions $\mathbf{N}(\mathfrak{G}_{n\times m})$ (i.e., how many distinct hamiltonian paths it has) is:

$$2^{(n-4)(m-4)/25} \leq \mathbf{N}(\mathfrak{G}_{n\times m}) < 7^{nm-1}$$

Empirical Results

Finding a Solution

Four different approaches have been tried to find hamiltonian paths:

- **Neo4J**:

Finding a Solution

Four different approaches have been tried to find hamiltonian paths:

- ◇ **Neo4J**:
  - **Raw** — Expand all possible paths of length $n \times m$ and check which are hamiltonian;

Finding a Solution

Four different approaches have been tried to find hamiltonian paths:

- **Neo4J**:
  - **Raw** — Expand all possible paths of length $n \times m$ and check which are hamiltonian;
  - **Constructive** — The query is built dynamically, at each step pruning choices that would not yield an hamiltonian path;

Finding a Solution

Four different approaches have been tried to find hamiltonian paths:

◇ **Neo4J**:

- **Raw** — Expand all possible paths of length $n \times m$ and check which are hamiltonian;
- **Constructive** — The query is built dynamically, at each step pruning choices that would not yield an hamiltonian path;
- **APOC** — Calling the apposite APOC function to return an hamiltonian path;

Finding a Solution

Four different approaches have been tried to find hamiltonian paths:

- **Neo4J**:
  - **Raw** — Expand all possible paths of length $n \times m$ and check which are hamiltonian;
  - **Constructive** — The query is built dynamically, at each step pruning choices that would not yield an hamiltonian path;
  - **APOC** — Calling the apposite APOC function to return an hamiltonian path;
- **Python**: Building the graph and searching a path using a backtracking algorithm.

Finding a Solution — Raw Implementation

```
MATCH p = (start:Node)-[:MOVE*{len}]->(end:Node)
WHERE ALL(
        n IN nodes(p)
        WHERE single(m IN nodes(p) WHERE m = n)
        )
RETURN p
```

Finding a Solution — Constructive Implementation

```
MATCH (n0:Node)-[:MOVE]->(n1:Node)
WHERE id(n1) NOT IN {id(n0)}
MATCH (n1)-[:MOVE]->(n2:Node)
WHERE id(n2) NOT IN {id(n0), id(n1)}
...
```

Finding a Solution — APOC Implementation

```
MATCH (start:Node)
CALL apoc.path.expandConfig(
startNode, {
        relationshipFilter: "MOVE>",
        minLevel: {parameters["pathLength"]},
        maxLevel: {parameters["pathLength"]},
        uniqueness: "NODE_GLOBAL",
        labelFilter: 'Node'
}
) YIELD path
```

Time Results

Board Size	Raw	Constructive	APOC	Python
4×5	1.1492s	0.0331s	0.0143s	**0.0034s**
4×6	$>$30s	0.6907s	**0.0307s**	0.0503s
4×7	$>$30s	16.6208s	**0.6338s**	0.6547s
5×5	$>$30s	3.1501s	2.9022s	**0.4530s**

Table 1: Time results in finding every solution, with constraints on the starting and ending nodes.

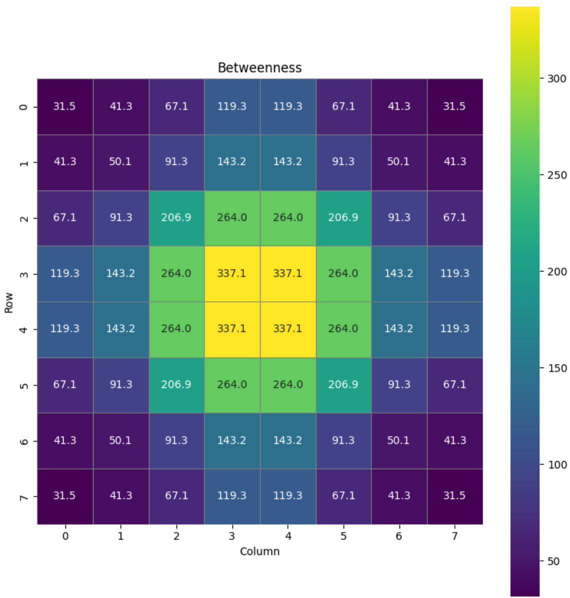
Time Results

Board Size	APOC	Python
5×6	**0.0578s**	0.4754s
4×8	**3.9755s**	6.3072s
5×7	0.1442s	**0.0007s**
6×6	**0.2700s**	0.5473s

Table 2: Time results in finding only one solution for larger boards.

So: as boards get larger, the usage of **GraphDBs seems more effective** than a more traditional approach. The APOC library also provides several critical optimization to reduce computing time.

Centrality in a Board



Centrality and Number of Solutions

Board	Betweenness	Closeness	Degree	Eigenvector
4×5	**-0.8228*****	-0.5899**	-0.7845***	-0.1728
4×6	**-0.7744*****	-0.6316***	-0.7197***	-0.5179**
5×5	**-0.6820*****	-0.2543	-0.4037*	-0.1460
4×7	**-0.9098*****	-0.8204***	-0.8596***	-0.5842**
5×6	-0.7353*	-0.4310	**-0.7729***	-0.5072

Table 3: Correlation coefficients (r) between Hamiltonian paths and centrality measures across different board sizes.

So: the more central a node is, **the less solutions** can be found starting from that node. It's empirically better to start the game from nodes **with low centrality values**.

References

References i

- R. C. contributors, “Solve a hopido puzzle,” https://rosettacode.org/wiki/Solve_a_Hopido_puzzle, 2025, accessed: 2025-09-17.
- Neo4j Labs, “Apoc: Awesome procedures on cypher,” 2025, version 2025.07.1core. [Online]. Available: https://neo4j.com/labs/apoc
- Neo4j, Inc., *Neo4j Graph Data Science Library Manual*, 2025, online documentation and software plugin. [Online]. Available: https://neo4j.com/docs/graph-data-science/current/
- M. Sipser, *Introduction to the Theory of Computation*, 3rd ed. Cengage Learning, 2013, pp. 292–314.

H. C. von Warnsdorff, "Des rösselsprunges einfachste und allgemeinste lösung." [Online]. Available: https://zs.thulb.uni-jena.de/receive/jportal_jparticle_00189099

Thank you!

Questions?