Graph-Based Analysis of Solution Spaces in the Game *Diplomatico*

Filippo Garagnani
Dipartimento di Ingegneria ”Enzo Ferrari”
Università degli Studi di Modena e Reggio Emilia
298707@studenti.unimore.it

***Abstract*—The game of *Diplomatico* (also known as ’The 100 Squares Puzzle’ or ’Hopido’ [1]) has been studied during the last decades. The few results found are limited to determining whether the game is solvable or not for certain given constraints. However, to the best of our knowledge, no one has ever tried to analyze different approaches to explore the solution space in its entirety. In this work, we model the game as a graph where nodes represent squares and edges represent valid moves between these cells. We apply various graph analytics techniques to uncover structural properties of the solution space, identify key states, and explore potential strategies for solving the game.**

I. Introduction

The game of *Diplomatico* is a combinatorial puzzle game, usually played on a square grid. The goal of the game is to fill the grid with numbers, by placing them according to the following rules:

1) The number 1 can be placed on any available square.
2) The number $i > 1$ must be placed either at a distance of 3 horizontally or vertically, or at a distance of 2 diagonally from the square containing the number $i - 1$.
3) If the number i cannot be placed, the game ends.

An example of valid moves is shown in Figure 1.

3			2
	6		
	1		
4			5

Fig. 1. A 4×4 grid following the rules of *Diplomatico*. The number 2 is placed diagonally from 1; while the number 3 is placed horizontally from 2.

As it can be seen, the requirement of a square grid is completely arbitrary. The game could be also played on rectangular grids with or without ’blocked’ cells (i.e., squares which are not available).

Some very simple results are known about the game. For example, it has been shown that the game is unsolvable for square grids of size 4×4 or smaller - except for the trivial case in which the grid is of size 1×1 (see Figure 2).

1

Fig. 2. The trivial case of a 1×1 grid.

The game can be easily modeled as a graph. Each square of the grid is represented as a node; each available ’move’ (i.e., a transition from a square to another according to the game rules) is represented as a directed edge between two nodes. Hence, as it will be shown, finding a solution means finding an Hamiltonian path inside this particular type of graph. It is known that determining if a graph allows an Hamiltonian path is an NP-Complete problem [2]. It will be shown, moreover, the number of solutions for the game of *Diplomatico* grows exponentially over the size of the grid.

II. Preliminaries

The following definitions are used throughout this work. The main case being considered is the one in which the game grid is rectangular and doesn’t have any cell blocked.

Definition II.1. *Board Graph*
Given a board of size $n \times m$ (with $n, m \in \mathbb{N}$), the ***Board Graph*** *of size $n \times m$ is an undirected graph $\mathfrak{G}_{n\times m} = (V, E)$, where:*

$$V = \{v_{i,j},\ i = 1, \dots, n \wedge j = 1, \dots, m\}$$

and where the following condition holds:

$$\forall e = \{v_{i,j},\ v_{k,\ell}\} \in E, \quad (|i-k| = 3 \wedge j = \ell) \\ \vee (|j-\ell| = 3 \wedge i = k) \\ \vee (|i-k| = |j-\ell| = 2)$$

Of course, the node v_{ij} will represent the square located at row i and column j of the original grid of the game. The condition required for the edge set E ensures that only legal moves are present as undirected arcs in a Board Graph. The edges are undirected because, of course, the movement rules are perfectly simmetrical (i.e., being able to move from cell A to cell B means that it’s also possible to move from cell B to cell A).

Definition II.2. *Winning Path Given a Board Graph $\mathfrak{G}_{n\times m} = (V, E)$, a* ***Winning Path*** *is an Hamiltonian Path in $\mathfrak{G}_{n\times m}$. Specifically, an Hamiltonian Path is a sequence of vertices $\mathfrak{h} = (v^1, v^2, \dots, v^{n\times m})$ such that:*

$$\forall \ell = 1, \dots, k-1, \ \{v^\ell, v^{\ell+1}\} \in E$$

and such that:

$$\forall v \in V, \ \exists!\, v^\ell \in \mathfrak{h} : v^\ell = v$$

An Hamiltonian Path is simply, as the Definition II.2 gives, a path in the graph that visits every node exactly once.

Definition II.3. *Solvability*
Given a Board Graph $\mathfrak{G}_{n\times m}$, it is said to be ***solvable*** *if:*

$$\exists \mathfrak{h} = (v^1, v^2, \dots, v^{n\times m}) :$$
$$\mathfrak{h} \textit{ is a Winning Path in } \mathfrak{G}_{n\times m}.$$

Definition II.3 simply states that a Board Graph is solvable if it contains at least one Hamiltonian Path; that is, if there is a way to start from a node (i.e., a cell in the game of *Diplomatico*), and to be able to reach every other node - exactly once - by following the game rules.

Simply from the Definitions II.1-II.3, it is possible to prove the following Proposition.

Proposition II.1. *Unsolvability of $\mathfrak{G}_{4\times 4}$*
The Board Graph $\mathfrak{G}_{4\times 4}$ is unsolvable.

Proof. We can prove the unsolvability of $\mathfrak{G}_{4\times 4}$ by contradiction. Assume that there exists a Winning Path $\mathfrak{h} = (v^1, v^2, \dots, v^{16})$ in $\mathfrak{G}_{4\times 4}$.
Since $\mathfrak{h}$ is a Hamiltonian Path, it must visit every node exactly once. Consider the nodes in the center of the board - specifically, vertices $v_{2,2}$, $v_{2,3}$, $v_{3,2}$, $v_{3,3}$ (see Figure 3). Being $\mathfrak{G}_{4\times 4}$ a Board Graph, the conditions for edges linking those vertices can be explicitly determined (focusing, for the time being, on $v_{2,2}$):

$$\begin{aligned}
&\forall e = \{v_{2,2},\ v_{i,j}\} \in E,\\
&(|2-i| = 3 \wedge 2 = j) \implies v_{i,j} \in \{v_{5,2},\ v_{-1,2}\}\\
&(|2-j| = 3 \wedge 2 = i) \implies v_{i,j} \in \{v_{2,5},\ v_{2,-1}\}\\
&(|2-i| = |2-j| = 2) \implies v_{i,j} \in \{v_{0,0},\ v_{0,4},\ v_{4,0},\ v_{4,4}\}
\end{aligned}$$

However, of the 'suggested' nodes above, only $v_{4,4}$ is actually present in $\mathfrak{G}_{4\times 4}$ (the others are to consider 'out of bounds'). This means that the node $v_{2,2}$ can only be connected to $v_{4,4}$, implying that it must be either the first or the last node in the Winning Path $\mathfrak{h}$ (since it has degree 1). The same reasoning can be applied to the other three 'central' nodes; this means that four nodes must either be first or last in the path. This is not possible, hence the contradiction. $\square$

The following Definition and Proposition will give a sense to the complexity of the game, and to the main focus of the work.

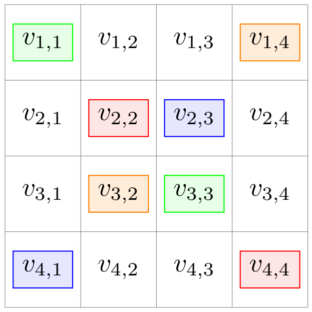

Fig. 3. The board representing the graph $\mathfrak{G}_{4\times 4}$. In corresponding colors, the only possible edges from each central node.

Definition II.4. *Numbers of Solutions*
Given a Board Graph $\mathfrak{G}_{n\times m}$, the ***Number of Solutions*** *$\mathbf{N}(\mathfrak{G}_{n\times m})$ is defined as:*

$$\mathbf{N}(\mathfrak{G}_{n\times m}) = |\{\mathfrak{h} : \mathfrak{h} \textit{ is a Winning Path in } \mathfrak{G}_{n\times m}\}|$$

Proposition II.2. *Bounds on $\mathbf{N}(\mathfrak{G}_{n\times m})$*
Given a Board Graph $\mathfrak{G}_{n\times m}$, with $n, m \geq 5$:

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

Proof. **Upper Bound.** Any winning path $\mathfrak{h}$ must pass from a 'corner' node (i.e., $v_{1,1}$, $v_{1,m}$, $v_{n,1}$, $v_{n,m}$). By evaluating the possibilities, it can be easily determined that only three edges are connected to any corner node.

$$\begin{aligned}
&\forall e = \{v_{1,1},\ v_{i,j}\} \in E,\\
&(|1-i| = 3 \wedge 1 = j) \implies v_{i,j} \in \{v_{4,1},\ v_{-2,1}\}\\
&(|1-j| = 3 \wedge 1 = i) \implies v_{i,j} \in \{v_{1,4},\ v_{1,-2}\}\\
&(|1-i| = |1-j| = 2) \implies\\
&\qquad v_{i,j} \in \{v_{3,3},\ v_{3,-1},\ v_{-1,3},\ v_{-1,-1}\}
\end{aligned}$$

In this specific case, the only possible adjacent nodes are $v_{4,1}$, $v_{1,4}$, $v_{3,3}$. The same reasoning applies to the other corner nodes.
Moreover, the start node - in the general scenario - will have at most 8 edges connected to it (assuming, from the previous verification, that each node is valid). Every other node, however, will have at most 7: surely, the edge chosen to reach it should not be considered, otherwise the Winning Path wouldn't be Hamiltonian. This means that, combinatorially speaking:

$$\mathbf{N}(\mathfrak{G}_{n\times m}) \leq 8 \cdot 3^4 \cdot 7^{nm-5} < 7^{nm-1}$$

$\square$

The more complex proof of the lower bound is given in Appendix A.

The derived bounds demonstrate that the number of solutions increases exponentially with the board's size. This exponential growth renders exhaustive search methods infeasible for larger boards, emphasizing the necessity for more efficient algorithms and heuristics to navigate the solution space effectively. Furthermore, identifying even a single solution is a challenging task. The subsequent sections will delve into different search strategies and evaluate their performance.

III. Implementation

In order to analyze the solution space in the game of *Diplomatico*, a few implementations have been developed. Three of them were done in Neo4J [3], leveraging its graph database capabilities to model the Board Graphs and perform various queries. To make the comparison valuable, another implementation in Python has been tested along the other. Before any query is run, the Board Graph is generated and stored in the database offered by Neo4J through a simple script that simply requires the number of rows and of columns of the board. Moreover, each query can be edited by command line arguments, by specifying - if needed - the start node of the path, the end node of the path and the total number of paths required.

The four main implementations of solution searching are briefly described below.

a) The `Raw` Implementation: The `Raw` implementation is a straightforward query in Neo4J, which tries to expand from a starting node to find one or more Winning Paths.

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

The query above simply matches a path p starting from any node labeled as `Node`, and tries to expand it by the number of edges specified in the parameter `pathLength` (which should be equal to $n \times m - 1$ for a board of size $n \times m$). The `WHERE` clause ensures that the path is Hamiltonian, by checking that every node in the path appears once and exactly once.

The starting node can be specified by preappending a `MATCH (start:Node {row: r, col: c})` clause before the main `MATCH` clause, where `r` and `c` are the row and column of the desired starting node. The same applies to the end node. Moreover, the query could be modified to return only a few solutions (by adding a `LIMIT` clause at the end of the query). Thanks to the Python API of Neo4J, the element inside the `parameters` dictionary is modified at runtime.

b) The `Constructive` Implementation: The `Raw` query, albeit simple, shows a few issues. The main problem being that it first finds all the possible paths of length $n \times m - 1$, and then filters out the non-Hamiltonian ones. This means that, for larger boards, the query will take a very long time to complete - if it completes at all. In order to address this issue, a more 'constructive' approach has been implemented. The idea is to start from a node, and to try to expand the path step by step, ensuring each time that the Hamiltonian property is preserved. The query is built in Python in the following way:

```
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)}
...
```

The query then contains every full path at the end, and these are guaranteed by construction to be Hamiltonian. The starting and ending nodes can be specified in the same way as in the `Raw` implementation, as well as limiting to only a few solutions.

c) The `APOC` Implementation: The `APOC` library [4] for Neo4J contains the `apoc.path.expandConfig` procedure. By leveraging its parameters, it's possible to convert it into an Hamiltonian-finding algorithm.

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

If needed, the starting node can be specified in the same way as in the previous implementations; the ending node can be passed as an additional parameter to the APOC function itself, just like the number of paths to return.

d) The `Python` Implementation: The last implementation is a pure Python one. It makes use of a backtracking algorithm to find one or more Winning Paths in a Board Graph. The algorithm is implemented in a recursive way, and it can be summarized as follows:

```
find_path(current_path, board):
    if current_path covers all nodes:
        return current_path;
    for each move:
        add move to current_path;
        find_path(current_path, board);
        remove move from current_path;
```

Another known problem, similar to *Diplomatico*, is the knight's tour problem. A well known heuristics for that problem is the Warnsdorff's rule [5]: this says to first visit nodes with fewest moves available. This heuristics has been implemented in the Python solution as well.

IV. Results and Discussion

In this section, the results of the different approaches in finding Hamiltonian paths for the *Diplomatico* graphs are analyzed. The experiments were conducted on Neo4J version 2025.07.01, and the APOC library version 2025.07.1core. The softwares were given maximum 16GB of RAM to execute the code. The following results have been obtained by running each query - or algorithm - 5 times, and by averaging the time taken to complete the task. The starting node and the ending node were not specified, and only one solution was required.

Board Size	**Raw**	**Constructive**	**APOC**	**Python**
4×5	0.4797s	1.3337s	0.1615s	**0.0085s**
4×6	>30s	2.3749s	0.2981s	**0.0180s**
4×7	>30s	4.4709s	0.6444s	**0.1897s**
5×5	>30s	5.7675s	0.7468s	**0.0040s**

While, instead, the following results have been obtained by running each query at the same conditions as above, but by also specifying the starting node as $v_{1,1}$ and the ending node as $v_{n,m}$ - the top-left and bottom-right corners of the board, respectively.

Board Size	**Raw**	**Constructive**	**APOC**	**Python**
4×5	0.7142s	0.0417s	0.0256s	**0.0032s**
4×6	>30s	0.0659s	**0.0250s**	0.0514s
4×7	>30s	3.2629s	**0.5479s**	0.5982s
5×5	>30s	0.6837s	0.4557s	**0.0147s**

In the following table, instead, not only the starting and ending nodes were specified, but also every solution was required.

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**

As it can be easily seen, the two most competitive implementations are the `APOC` one and the pure `Python` one. In order to compare them, a few more tests have been conducted on larger boards - excluding the too slow `Raw` and `Constructive` implementations. More specifically, each of the following time results has been obtained as a mean over 20 runs; only one solution was required and no starting and ending node were specified.

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

As expected, as the board size increases, the time taken to find a solution has an exponential growth. Curiously enough, it seems that - in general - solutions are easier to be find for more 'regular' board sizes, rather than very rectangular ones. This could be due to the fact that, in more rectangular boards, there are less possible moves available from each node - hence, the search space is smaller. Overall, the `APOC` implementation seems to be more stable in terms of time taken for runs, while the `Python` one starts going slower as the complexity rises. The board 5×7 is solved very fast - probably due Warnsdorff's heuristics working accurately well in this particular case.

Hereafter follow an overview of the number of solutions.

Board Size	**N° of Solutions**
4×5	144
4×6	128
4×7	72
5×5	12400

V. Conclusion and Future Work

This work has presented a comprehensive graph-based analysis of the solution space in the game of *Diplomatico*. By modeling the puzzle as a Board Graph and leveraging both Neo4J and Python implementations, we have demonstrated the exponential complexity inherent in enumerating Hamiltonian paths and provided empirical benchmarks for several search strategies. The results highlight the effectiveness of advanced graph database procedures (such as APOC) and heuristic-driven algorithms (notably Warnsdorff's rule) in efficiently navigating large solution spaces.

Key findings include the derivation of theoretical bounds on the number of solutions, the identification of structural properties that impact solvability, and the comparative performance of different computational approaches. The experiments confirm that exhaustive search quickly becomes infeasible as board size increases, underscoring the need for scalable heuristics and optimized queries.

Future work may focus on extending these methods to boards with blocked cells, exploring parallelization strategies, and integrating machine learning techniques to predict solvability or guide search. Additionally, further theoretical investigation into tighter bounds and the combinatorial structure of solution spaces could yield deeper insights. The approaches and tools developed here provide just a solid foundation for still unexplored game of Diplomatico.

References

[1] R. C. contributors, "Solve a hopido puzzle," https://rosettacode.org/wiki/Solve_a_Hopido_puzzle, 2025, accessed: 2025-09-17.

[2] M. Sipser, *Introduction to the Theory of Computation*, 3rd ed. Cengage Learning, 2013, pp. 292–314.

[3] Neo4j, Inc., *Neo4j Graph Database*, 2025, version 2025.07.01. [Online]. Available: https://neo4j.com

[4] Neo4j Labs, "Apoc: Awesome procedures on cypher," 2025, version 2025.07.1core. [Online]. Available: https://neo4j.com/labs/apoc

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

Appendix A
Over the Lower Bound of the Number of Solutions

Here is the detailed proof of the lower bound referenced in the main text, specifically for Proposition II.2. First, some handful lemmas are presented.

Lemma A.1. *For any Board Graph $\mathfrak{G}_{n\times m}$ with $9 \geq n, m \geq 5$, there exist at least 2 distinct Winning Path $\mathfrak{h}$ such that:*

$$\mathfrak{h}_1 = v_{3,\,1} \land \mathfrak{h}_{nm} = v_{3,\,m-2}$$

The Lemma A.1 can be proved, to the best of my knowledge, only by enumerating all the possible couples $(n,\, m)$ - with, without loss of generality, $n \geq m$, otherwise the indeces of the nodes could be easily swapped - and by explicitly finding the 2 distinct Winning Paths for each of them. What the Lemma states is that there is a guarantee of 2 different solutions which 'start' from the border node $v_{n-2,\,1}$ and that end with the node displaced by 2 both horizontally and vertically from the outer edges. This constraint guarantees the possibility to create a tilization of the board for every $n, m \geq 5$ - hence, guaranteeing a lower bound.

Lemma A.2. *Given the Board Graph $\mathfrak{G}_{(n+k)\times m}$ with $9 \geq n, m, k \geq 5$, then $\mathfrak{G}_{(n+k)\times m}$ is solvable.*

Proof. Consider the two Board Graphs $\mathfrak{G}_{n\times m}$ and $\mathfrak{G}_{k\times m}$. By Lemma A.1, both of them are solvable - since $9 \geq n, m, k \geq 5$. Let $\mathfrak{h}^1$ be one of the Winning Paths in $\mathfrak{G}_{n\times m}$ guaranteed to exists; and let $\mathfrak{h}^2$ be one of the Winning Paths in $\mathfrak{G}_{n\times k}$ guaranteed to exists. Defining $\hat{\mathfrak{h}}^2$ in the following way:

$$\hat{\mathfrak{h}}^2 = (v_{i,j+m} : v_{i,j} \in \mathfrak{h}^2)$$

So: the new defined path $\hat{\mathfrak{h}}^2$ is simply the path $\mathfrak{h}^2$ 'shifted' horizontally by m columns. It can be proven that the following $\mathfrak{h}^3$ is indeed a Winning Path:

$$\mathfrak{h}^3 = (\mathfrak{h}^1,\, \hat{\mathfrak{h}}^2)$$

First of all, the two paths $\mathfrak{h}^1$ and $\hat{\mathfrak{h}}^2$ are disjoint, since they refer to different nodes of the graph; moreover, they completely 'touch' each node of the partitions, being Winning Paths. The only thing left to prove is that the last node of $\mathfrak{h}^1$ is connected to the first node of $\hat{\mathfrak{h}}^2$. By Lemma A.1, the last node of $\mathfrak{h}^1$ is $v_{3,\,m-2}$; while the first node of $\hat{\mathfrak{h}}^2$ is $v_{3,\,m+1}$. It can be easily verified that:

$$(|m + 1 - m - 2| = 3 \land 3 = 3)$$

hence, the edge $\{v_{3,\,m-2},\, v_{3,\,m+1}\}$ exists, giving a complete solution for the Board Graph $\mathfrak{G}_{(n+k)\times m}$. □

The result exposed in the Lemma A.2 is graphically represented in Figure 4. It's to note that the above Lemma A.2 can be easily adapted to increasing values of columns for Board Graphs.

17	9	2	16	8	42	34	27	41	33
4	14	19	5	13	29	39	44	30	38
1	22	25	10	21	26	47	50	35	46
18	6	3	15	7	43	31	28	40	32
24	11	20	23	12	49	36	45	48	37

Fig. 4. A 5 × 10 grid solved by tiling. The two halves have the same path connecting $v_{1,3}$ to $v_{3,3}$. In blue, the 'link' between two otherwise disjoint paths.

The following Lemmas, instead, focus on stacking Board Graphs vertically.

Lemma A.3. *For any Board Graph $\mathfrak{G}_{n\times m}$ with $9 \geq n, m \geq 5$, there exist at least 2 distinct Winning Path $\mathfrak{h}$ such that:*

$$\mathfrak{h}_1 = v_{3,\,1} \land \mathfrak{h}_{nm} = v_{n,\,m}$$

Once again, a proof of the Lemma would require an enumeration of all the possible couples $(n,\, m)$, and their respective solved distinct Hamiltonian.

Lemma A.4. *Given the Board Graph $\mathfrak{G}_{n\times(m+k)}$ with $9 \geq n, m, k \geq 5$, then $\mathfrak{G}_{n\times(m+k)}$ is solvable.*

Proof. Consider the two Board Graphs $\mathfrak{G}_{n\times m}$ and $\mathfrak{G}_{n\times k}$. Without loss of generality, assume $m \geq k$. Then, by knowing Lemma A.3, both of them are solvable - since $9 \geq n, m, k \geq 5$. Let $\mathfrak{h}^1$ be one of the Winning Paths in $\mathfrak{G}_{n\times m}$ guaranteed to exist; and let $\mathfrak{h}^2$ be one of the Winning Paths in $\mathfrak{G}_{n\times k}$ guaranteed to exist, either by Lemma A.1 or by Lemma A.3. Then, define $\hat{\mathfrak{h}}^2$ in the following way:

$$\hat{\mathfrak{h}}^2 = (v_{i+n,m-j+1} : v_{i,j} \in \mathfrak{h}^2)$$

This path $\hat{\mathfrak{h}}^2$ is simply the path $\mathfrak{h}^2$ swapped over the columns and 'shifted' vertically by n rows. It can be proven that the following $\mathfrak{h}^3$ is indeed a Winning Path:

$$\mathfrak{h}^3 = (\mathfrak{h}^1,\, \hat{\mathfrak{h}}^2)$$

First of all, the two paths $\mathfrak{h}^1$ and $\hat{\mathfrak{h}}^2$ are disjoint, since they refer to different nodes of the graph; moreover, they completely 'touch' each node of the partitions, being Winning Paths. The only thing left to prove is that the last node of $\mathfrak{h}^1$ is connected to the first node of $\hat{\mathfrak{h}}^2$. By Lemma A.3, the last node of $\mathfrak{h}^1$ is $v_{n,m}$ while the first node of $\hat{\mathfrak{h}}^2$ is $v_{n+3,\,m}$. It can be easily verified that:

$$(|n + 3 - n| = 3 \land m = m)$$

Hence, the edge $\{v_{n,m},\, v_{n+3,\,m}\}$ exists, giving a complete solution for the Board Graph $\mathfrak{G}_{n\times(m+k)}$. □

10	5	2	26	8	3
22	15	12	21	18	13
1	27	9	4	28	25
11	6	19	14	7	20
23	16	29	24	17	30
44	49	56	43	48	55
52	35	46	51	34	41
38	58	32	37	57	31
45	50	53	42	47	54
60	36	39	59	33	40

Fig. 5. A 5×6 grid solved by tiling. The two halves have the same path, albeit rotated over the columns, connecting $v_{3,1}$ to $v_{5,6}$ and then $v_{8,6}$ and $v_{10,1}$. In blue, the 'link' between two otherwise disjoint paths.

Proof. **Lower Bound**. We will show that for any $n, m \geq 5$:

$$\mathbf{N}(\mathfrak{G}_{n\times m}) \geq 2^{(n-4)(m-4)/25}$$

To establish this, we can use a combinatorial argument based on the structure of the board graph. More specifically, given a generic Board Graph $\mathfrak{G}_{n\times m}$, we can partition it into smaller sub-graphs of size 5×5 plus - in case of n, m not divisible by 5 - some additional sub-graphs of size $k_i \times \ell_i$, with $9 \geq k_i, \ell_i \geq 5$. Thanks to Lemma A.2 and Lemma A.4, we know that all these sub-graphs are solvable, and particularly that the whole Board Graph $\mathfrak{G}_{n\times m}$ is also solvable, by tiling the sub-graphs accordingly.

The total number of sub-graphs in which to partition the original Board Graph is $\lfloor \frac{n}{5} \rfloor \cdot \lfloor \frac{m}{5} \rfloor$. This can be proven by simply considering that each sub-graph has a size of 5×5; hence, the number of sub-graphs that can fit in the original Board Graph is simply the integer division of the dimensions of the original graph by 5. Each of these sub-graphs has at least 2 distinct Winning Paths, as guaranteed by Lemma A.1 and Lemma A.3. This means that, combinatorially speaking, the total number of distinct Winning Paths in the original Board Graph is at least:

$$\mathbf{N}(\mathfrak{G}_{n\times m}) \geq 2^{\lfloor n/5 \rfloor \cdot \lfloor m/5 \rfloor}$$

In order to show the original, exponential in n, m, bound, it's only necessary to consider that:

$$\lfloor n/5 \rfloor \geq (n-4)/5$$

This is because, in the worst case scenario, $n = 5 \times k + 4$; hence, $\lfloor n/5 \rfloor = k$ and $(n-4)/5 = k$. The same reasoning applies to m; hence, the original bound is proven. □

Appendix B
Over the Creation of a Board Graph in Neo4J

In order to provide a common ground for retrieving a solution in Neo4J [3] (see Section III), a variable-dependent query was designed to create a Board Graph of desired size. The structure of the graph is organized in the following way:

- Each node is created with the label `Node`, and has - as properties - *row* and *col*, respectively the indeces of row and of column in the original board-like structure.
- Each link between two nodes (present only if a legal move allows the movement between those two nodes) is created, with label `MOVE`.

The Python function implementing the creation of this query is described in the current Appendix in pseudocode.

```
create_graph_query(rows, cols):
 for i = 1, ..., rows:
  for j = 1, ..., cols:
   query += create_query(i, j)
```

This snippet regulates the creation of all the necessary nodes, each with its own specific properties. The code loops through all the possible row indices (from 1 to the input parameter `rows`) and through all the possible column indices. For every possible pair of row and column indices, then, it stores a line created by the method `create_query(i, j)`. This returns the following query:

```
"CREATE (n_i_j:Node {row: i, col: j})\n"
```

of course, substituting the two parameters i, j with the passed values. For instance, for the node at position $(2, 3)$, the line will ask to create a node with label `Node`, called `n_2_3`, and with properties `row`=2 and `col`=3.