*Journal of Computer Science*

Original Research Paper

# p ≠ np: The Set of Deterministic Problems that are Solvable in Polynomial Time is Unequal to the Set of Non-Deterministic Problems that are Solvable in Polynomial Time

**Andres Boldori**

*Department of Mathematics, University of Zurich, Zurich, Switzerland*

*Article history*
Received: 23-01-2024
Revised: 26-03-2024
Accepted: 06-04-2024

Email: boldori084@gmail.com

**Abstract:** This study constructs a solution to the "p vs. np" problem using complexity theory. We show through counterexamples that p ≠ np and formalize the two sets using stochastic, probabilistic and non-deterministic modeling. While the well-known sets "pspace" and "npspace", analyzing the storage of a device, can be claimed to be equal, p and np differ and are exclusively defined through the elapsed time of their algorithms. Indeed, calculations including the probabilistic family of discrete uniform distributions prove the well-known inequality p ≠ np. In this study, using complexity and probability theory, we give some examples that fit into the new theory: There are problems that can be solved by non-deterministic Turing machines and which are in np (they are non-deterministic and just of polynomial time growth), but they are not in p itself (since they are not, deterministic and just of polynomial time growth).

**Keywords:** p ≠ np, Complexity Theory, p and np Complexity, Exponential Runtime, Hamiltonian Paths

## Introduction

The amazing aim of this study is to provide a formalization and a probabilistic, even almost combinatorial proof of the inequality of the well-known theoretical sets p and np. This is a short contribution to science, as the inequality already follows mathematically from the literature and therefore, no further simulations or experiments are needed except the ones already at hand within the bibliography. This study generalizes the work of Glock *et al*. (2022) "the n-queens completion problem", (Gent *et al*., 2017) "complexity of n-queens completion", (Conrad *et al*., 1994) "solution of the knights Hamiltonian path problem on chessboards" and (Squirrel *et al*., 1996) "A Warnsdorff-Rule Algorithm for Knights Tours on Square Chessboards" to a proof of p ≠ np. This theoretical generalization ultimately and almost becomes independent of the original papers. In addition, we use the experience of the Monte Carlo simulations on the chessboard within Silver *et al*. (2018) "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". Notice that the literature and references mentioned within the referenced books and papers themselves also are indirectly relevant to this study. The solution of p ≠ np is quite difficult to find within the family of discrete uniform distributions, but easy to understand.

### *Research Question, Main Research Objecti-ves and Basic Definitions*

The clay mathematics institute of Oxford defined p vs. np as one of the most important problems in technical and mathematical science and we show in this study, that it can be solved as an application of complexity theory (Compare to Encyclopedia of Philosophy, 2015) "computational complexity theory"). For instance, testing for a given solution candidate of a problem to be indeed a solution is not resource-taking at all, whereas, solving an arbitrary problem that might be unresolved, is, a priori, arbitrarily expensive. From the probabilistic point of view, the right class of problems must be taken into account to prove and formalize p ≠ np. We start with the definition of the deterministic Turing machine and go ahead with the theoretic and probabilistic model of a non-deterministic Turing machine (Fig. 1).

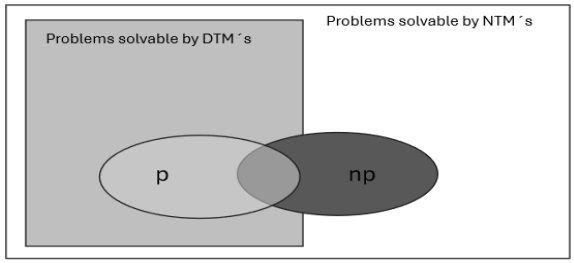


**Fig. 1:** Diagramme of the two theoretical sets p and np

*Def. Deterministic Turing Machine (DTM)*

A deterministic Turing machine is a 7-tuple:
$A = \{ Q, q_0, \Gamma, \nabla, \Sigma, F, \delta \}$ where:

| | |
|---|---|
| $Q$: | Is a finite non-empty set of states |
| $q_0 \in Q$: | The initial state |
| $\Gamma$: | Finite, non-empty set of tape alphabet symbols |
| $\nabla \in \Gamma$: | The blank symbol |
| $\Sigma \subseteq \Gamma \setminus \nabla$: | The set of input symbols |
| $F \subseteq Q$: | The set of final states or accepting states |
| $\delta$: | $(Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ the transition function |

*Def. Non-Deterministic Turing Machine (NTM)*

A non-deterministic Turing machine is a 7-tuple:
$A = \{Q, q_0, \Gamma, \nabla, \Sigma, F, \delta\}$ where:

| | |
|---|---|
| $Q$: | Is a finite non-empty set of states |
| $q_0 \in Q$: | The initial state |
| $\Gamma$: | Finite, non-empty set of tape alphabet symbols |
| $\nabla \in \Gamma$: | The blank symbol |
| $\Sigma \subseteq \Gamma \setminus \nabla$: | The set of input symbols |
| $F \subseteq Q$: | The set of final states or accepting states |
| $\delta \subseteq$ | $((Q \setminus F) \times \Gamma) \times (Q \times \Gamma \times \{L, N, R\})$ the transition relation |

The difference between a Deterministic (DTM) and a non-deterministic (NTM) Turing machine hence just lies in the different definition of $\delta$. A DTM and an NTM usually operate within an infinite tape $T$: $Z \rightarrow \Gamma$ (compare with the definition above) and have a read/write head that can read from and write to the tape (Fig. 2). The algorithm which is executed by the machine, as well as a large but finite memory, is specified by the transition function (DTM), or by the transition relation (NTM). A DTM is executed by applying its transition function delta to the current state and tape symbol repeatedly until termination (acceptance of the execution). As soon as the DTM enters one of the final states, it halts (terminates) and the calculation is over. Symbols written on the tape can be overwritten or erased by writing a blank symbol, which is equivalent to an empty field.

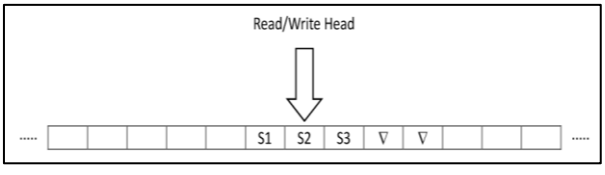


**Fig. 2:** Infinite tape of a deterministic turing machine already containing the three symbols S1, S2 and S3 and two blank symbols

The DTM transition function gives you the one and only function value (“new state”, “symbol to write") in return if the pair (“current state”, “symbol on tape") is given. The transition relation of the NTM, on the other hand, gives to any single possible (“state”, “symbol”) pair a whole set of, for the calculation possible, new (“state”, “symbol”) pairs (in the shape of an overall set). The transition relation therefore yields an entire tree of execution commands if applied many times. A non-deterministic Turing machine allocates all possible paths of a problem in parallel by giving a tree of possible execution paths as an outcome. Like a discrete stochastic process, the possibilities for execution at a given time form an entire set of objects. A ("stochastic") execution path like the one of the NTM terminates if and only if one path of the tree terminates. The tape of an NTM is not used but only to verify the paths of the NTM. Also, the NTM is executed by applying its transition relation delta repeatedly until termination. An NTM is said to be terminating, if one path of the repeated concatenations of the transition relation terminates. The definition of the deterministic Turing machine above could as well be changed and adapted easily by having an additional possibility of its "head" staying where it is (N), like within the one of a non-deterministic Turing machine. The head of the NTM and DTM will turn to the Left (L) and to the Right (R) on the infinite tape (or, which has no influence on the proof below, stay where it is (N), if an NTM). Neither a DTM nor an NTM needs to terminate necessarily, as they will remain within an infinite loop instead. Before starting the calculation and its algorithm, the DTM and the NTM do have the possibility of the tape to already contain a finite number of entries out of the set of input symbols $\Sigma$.

*Def. Problem*

A problem $X$ is an ordered set (3-tuple) containing a set $a$ of input values, a set $b$ of output variables and an unknown algorithm $w$. A solution to a problem $X$ is a set of values $y = b(w)$.

*Def. p, np*

$p$ is the set of problems, which possess a solution that can be calculated by a DTM in polynomial time. A problem $A$, which can be solved by a DTM within polynomial time lies in p: $A \in p$.

$np$ is the set of problems, which possess a solution that can be calculated by an NTM in polynomial time. A problem B, which can be solved by an NTM within polynomial time lies in np: $B \in np$.

The sets p and np are defined as the sets of problems which can, respectively, be solved by a DTM (deterministic Turing machine) and an NTM (non-deterministic Turing machine) in polynomial time as a function of the size of the problem parameters. The only

difference in their definition is that the set np allows the problems within to be solved by a tree but the set p just allows the problems within to be solved by concatenations of the same function, algorithm, calculation steps, evaluation steps (all within polynomial time, the number of transition relation/function concatenations). If the definitions were not restricted to polynomial growth in the time variable, questions about the two sets would certainly be less interesting. It is equivalent to ask if there is one realization of the theoretical concept of an NTM at all (which is not a DTM itself) for polynomial time algorithms, or is the theoretical definition of an NTM redundant and unnecessary even if restricted in time? The transition function must not necessarily be small, it can be an algorithm or just an entire "pc". Below, Problem $X$ is defined as one counterexample of the invalid equation p = np.

## Proof of n ≠ np

The reader will certainly have noticed that the definitions of the sets p and np belong to those of a DTM and an NTM, but restricted to polynomial time efficiency. The readers should also have noticed that the definition of an NTM contains all the models of all possible deterministic Turing machines and therefore is a generalization thereof. Are the sets of problems solved by NTM´s and DTM´s differing from each other under the polynomial time growth restriction? The problem of p vs. np can be translated into a logical one which is formally more convenient to solve: The DTM and the NTM are, up to their "execution" (transition relation or function), identical. Each transition function can be translated into a transition relation, but the converse does not hold. We need to prove that there is one problem ($X$), which can be calculated by an NTM within polynomial time, but whose result cannot longer be calculated by a DTM in polynomial time. The following problem $X$ can be solved by an NTM within polynomial time, which means that there is a tree of polynomial depth, containing the solution within one path, but there is no DTM which can solve (compute the output or outcome of) problem $X$ within polynomial time.

### *Problem X*

Is there a sequence q containing respectively one number per column of a random $m \times n$ matrix $X$ of "integers" (numbers in $\mathbb{Z}$ ), with entries drawn independently and all from the same discrete uniform distribution $U[-v, v] \subset \mathbb{Z}$, with $v$ arbitrary and fixed in $\mathbb{N}$ ($\mathbb{N}$ not containing 0), which has got sum zero, $\sum_{i=1}^{n} q_i = 0$ ? (Each of the possible outcomes $\{-v, -v+1, \ldots, v-1, v\}$ of one entry of $X$ occurs with probability $\frac{1}{2v+1}$ .)

**Matrix X**

| Columns | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 8. | ..... | n. |
|---|---|---|---|---|---|---|---|---|---|---|
| Object 1 | 2 | 5 | -7 | 8 | -3 | 15 | 1 | 0 | ..... | ..... |
| Object 2 | -5 | 8 | 10 | -20 | 13 | 5 | 8 | 4 | ..... | ..... |
| Object 3 | ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... |
| ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... |
| Object m | ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... | ..... |

| Solution q | 2 | 5 | -7 | -20 | 13 | 15 | 8 | 4 | ..... | ..... | Sum = 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|

**Fig. 3:** One possible realization of Matrix $X$ of problem $X$. The emphasized "squares" show a possible solution sequence candidate $q$ to problem $X$ which is "summed up" for verification within a separated additional row below

### *The Solution to Problem X*

The number of columns $n$ of matrix $X$ is the size parameter of problem $X$, it may be increased for comparison. The range v from which the discrete uniform distribution is drawn as the number of rows m of matrix $X$ remain fixed. Problem $X$ and the used discrete probabilistic uniform distribution contradicts the equality of the two sets p and np, in other words, we draw from the uniform distribution and want to see if the sum of the outcomes "cancels out" to 0 again. The table above shows a possible outcome situation of matrix $X$ as an example. Within a random matrix $X$ of integers as described and shown in the table above, the number of possible different sequences $\{q_i\}_{n \geq i \geq 1}$ of the random variables $\{X_{i,\ j}\}_{i,\ j}$ (entries of the Matrix $X$) containing exactly one number per column is $m^n$ (which is exponential in $n$ and not *polynomial*). It follows directly: Problem $X \notin p$, since the numbers of the matrix $X$ are drawn randomly and independently. The Table above (Fig. 3.) shows one sample of the matrix X of random numbers (integers) and a possible path q (compare to Glock *et al*. (2022) "the n-queens completion problem").

1. $q_i$ should appear within column $i$ in the table (Fig. 3.). There are therefore $m$ comparisons per element to verify $q$ as a solution as there are $m \cdot n$ comparisons in total
2. The sequence $q$ must be summed up and the result compared to 0 to verify it. To calculate this sum, $n$-1 additions are needed

The number of operations needed to confirm whether a new sequence is or is not a solution path of problem $X$ is polynomial in $n$: $m \cdot n + n\text{-}1 = n \cdot (m + 1)\text{-}1$. In the worst case, there are $m^n$ sums of all possible paths to be calculated. Since the numbers within matrix $X$ are independently drawn, each case is the worst case and each algorithm solving the problem must calculate the entire amount of sums to know if there is one sequence with a sum equal to zero, or not. So, there are $m^n$ sums to calculate anyway within problem $X$ and therefore, there is

no DTM that can solve it within polynomial time (problem $X$ lies outside $p$). But to confirm or to reject a solution of problem $X$ requires as well not more than polynomial time and therefore it must lie in np. More formally speaking, in our case, all possible “one number per column sums” of an in the variable $n$ increasing $m \times n$ matrix $X$ cannot be calculated fast enough (within polynomial time) by a DTM if $X$ is a realization of n·m independent and identically (*i. i. d.*), uniformly distributed random variables. The notion of a random variable is not necessarily primary in this article, since we only analyze its realization (the “outcome” of, see probability theory). Nonetheless, problem $X$ can be verified by an NTM just within polynomial time. As a matter of fact, problem $X \in np$ but problem $X \notin p$ and the two sets cannot be equal. There are problems solved by NTM’s which are in np, but not in $p$. The NTM within the proof below can solve problem $X$ within a polynomial amount of transition relation applications. A solution provided by an NTM is always of theoretical nature since it is a tree. An NTM never solves a problem directly but does test (allocate) if the numbers and objects within are ready for calculation and then decides if the problem is solvable/terminating, or not. If we wanted to formalize the “stochastic and probabilistic" definitions of an NTM into real-world calculation steps for problem $X$, the NTM would sum up at a given time $n_0$, all possible (and in total $m^{n0}$) partial sums, up to column $n_0$, which contain one number per column. The DTM instead calculates the output of the problem explicitly and path by path. So, classically it will not calculate all partial sums but an entire sum of one possible path and will alter its entries systematically until all sums are calculated, achieving the same result. A part of the algorithm and also of the transition function and transition relation the DTM and NTM would use for problem $X$ (in common) does look like the following:

1. Add all numbers of a sequence $q$ containing one number per column of Matrix $X$, $\rightarrow s$
2. If $s = 0$, return true (coding yes), otherwise return false (coding no)

We still did not specify “a sequence", hence the set definitions below hold to both, to a DTM and to an NTM solving problem $X$:

$Q$: = {“read”, “write”, “True”, “False”}
$q_0$: = “write”
$\Gamma$: = $\{z \in Z \mid -v \cdot n \leq z \leq v \cdot n\} \cup \{\nabla\}$ write a partial sum of problem $X$
$\Sigma$: = $\Gamma \backslash \nabla$
$F$: = {“*True*”, “*False*”}

Finally, the definitions of the DTM and NTM solving problem $X$ are, respectively, for a DTM $\delta_0$: =:

1. If the current state is “read", read the current symbol on the tape and store it in the memory. If it is a zero, change into state "True" (The DTM terminates), otherwise change state into "write" and turn the read/write head to the right
2. If $k = m^n + 1$, change state into "False" (the DTM terminates)
3. If $k = 1$, define the first row of entries of the matrix $X$ as the first “adequate” sequence r
4. If $k \geq 1$ and the current state is "write", add the adequate sequence r of entries of matrix $X_{i,j}$. Write its sum to the tape. Change current state into state “read”. Define the new adequate sequence of entries of the matrix $X$ as follows: Alter the last entry of the adequate sequence r according to the last column of matrix $X$, after that the last two entries according to the last two columns of matrix $X$, ... until as well altering the first entry of the adequate sequence according to the first column (once) of $X$. Repeat all past steps, respectively, once again for each of the entries of the first column of matrix $X$, until all and (in total) $m^n$ different possible sequences have been summed once $k$: = $k$+1

And for an NTM, $\delta_1$: =:

1. Store all possible partial sums of matrix $X$ including all columns up to $j$ as a new $m^j$- dimensional vector $h$. If $j = n$ and one entry of $h$ is zero, change into state “True”, else change into state “False” $j$: = $j$ +1

The algorithms on both machines repeat the execution of $\delta_i$ until termination (each repetition is one time unit) and the variables before starting the calculation are set to $i$: = 1, $j$: = 1, $k$: = 1. $\delta_0$ takes the input as possible sequence by possible sequence, where $\delta_1$ takes the input column by column. The DTM differs from the NTM in the calculation procedure, but both always terminate and yield the result, if there is a sequence within the outcome of the random discrete uniform matrix $X$ as we defined it above. The application of the transition function or transition relation requires one unit of time. The required space is not analyzed further (i.e., large enough). There is no transition function replacing the transition relation $\delta_1$ for problem $X$ within an algorithm that runs just a polynomial amount of time (number of transition function applications), since there are exponentially many possible sequences of matrix $X$, to be calculated. Since the NTM also writes an (in n) exponential amount of symbols within each application of its transition relation, the elapsed time (counting the number of transition relation applications), is again polynomial (the exponential growth cancels out). Still, each path of the execution tree (of polynomial depth) the NTM (above) generates can be verified (traversed) within a polynomial amount of time (since there is just a polynomial amount of transition relation applications, of the NTM within the set np). There is no transition function replacing the NTM in polynomial time, since the transition function of the DTM just writes a predefined constant, non-random (and therefore in time

just linearly increasing) amount of symbols at each application of the transition function. Hence np ≠ p.

### *The Theorem of Savitch as a Conclusion of the Proof p ≠ np*

### *Def. Pspace, Npspace*

The two sets of all problems, which can be solved by a DTM and an NTM within polynomial space, respectively.

The theorem of savitch (informal):

1. Each NTM with traversing time *n* can be replaced by a DTM with traversing time $m^n$, where m is the maximal amount of objects per node (calculation step of the NTM)
2. pspace = npspace

Complexity theory lies somewhere in the intersection of computer science and mathematics. We don't discuss applications of the theorem of savitch nor its proof, directly in this article but we try to explain its assumptions. A (finite) NTM's calculation possesses n nodes, one for each application of the transition relation, each containing a set of up to m different objects to possibly calculate. An NTM and a DTM calculate the solution of the given problem from node 1 to node n. The calculated "nodes" (transition function applications) of a DTM are sequentially and simply connected, whereas the objects to calculate within the nodes of an NTM are connected like within a tree to one or to many objects of the next node, respectively. Within the traveling salesman problem, for instance, each of the nodes of the NTM would not just contain one city, but all cities to travel to (and all nodes would be identical) as they would be connected through their objects, the next cities to travel to. Here, the connection of the nodes defines the travel (cities A, B, C, D, E) and not the (identical) nodes themselves.

Each planar graph can be embedded within a new graph by defining all nodes containing exactly all possible objects (Fig. 4.).

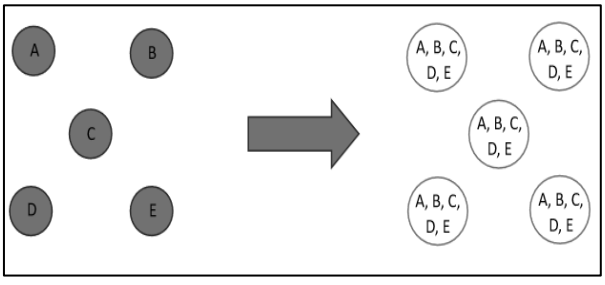


**Fig. 4:** From now on, each node contains all possible objects (and cities) and not just one"

We leave the above $K_7$ (Fig. 5.) aside and recall the proof of p ≠ np. If we use the notion of graphs, objects and nodes, a DTM possesses just one object per node (transition function application). The nodes of an NTM (transition relation application) contain each and in total more objects than a single one. However, not all algorithms share the same calculation efficiency. An algorithm needs one unit of time to traverse one node in addition (as such, time is defined here). The NTM now starts at node one and calculates all paths "in parallel", yielding a tree of calculation possibilities as a result and stops at node *n*. Defining $m \cdot n$ new nodes, one for each object of the *n* NTM's nodes, allows one single DTM to traverse the same calculation. Since the generated tree (of the NTM) possesses at least $m^n$ edges (connection of the objects with the ones of the next node), a DTM can in general not solve an NTM's problem within less time than $m^n$ (transition function applications). Indeed, we just take a look at algorithms which increase the units of time needed as a polynomial function of the problem size parameters. Each problem that can be solved by a non-deterministic Turing machine can also be solved explicitly by a theoretic probabilistic (and mathematical) model. However, the Theorem of Savitch also concerns data storage. The two sets p and np are defined just through their polynomial traversing time and therefore differ within calculations including the family of uniform distributions. The sets pspace and npspace could indeed be identical, since the space used by an algorithm within a device does not depend on the way the data is stored, deterministically or non-deterministically. The result npspace = pspace is therefore not an exclusively surprising one. All the conclusions in this particular section do not and cannot replace any additional proof of any additional theorem.

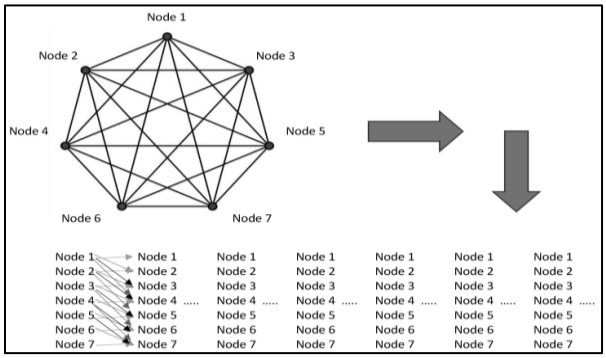


**Fig. 5:** Also the edges of a $K_n$ let themselves capture within a table: Two identical columns containing all possible nodes are aequivalent to the $K_n$, but in our terms still not convenient to represent it graphically. *N* identical columns containing all of n different nodes can instead be used by an NTM to calculate along the paths of a $K_n$.

*The Queens Completion Problem*

The queens completion problem fits into the framework of NTM's and DTM's. NTM's for instance are already used for time allocation, computational decision problems, verification of solutions of a problem, randomization, randomly accessed space, etc. The chessboard and chess (alphazero, chessprogramme from DeepMind) may not be completely independent of this discussion. "{read, write, True, False}" separates the DTM from the NTM and p from np. If we took a solving path of the well known travelling salesman problem, as another example of a Hamiltonian paths problem, it would not be difficult to recognize it as one within a reasonable amount of time, but to find or to discard that there is another one, is more time-expensive. Another issue is that the queens completion problem still cannot be classified as polynomial or exponential in time. Both, like many other problems of this class lead to the question, if $p \neq np$, or not. However, neither the travelling salesman nor the queens completion problem are convenient to show the inequality. Again, the aim of this section is not necessarily to classify the queens completion problem within the sets of problems p and np or even to solve it, but to show an example of our framework (within the proof of $p \neq np$).

*Remark*

An 8×8 chessboard is a black and white squared board. There are 32 figures in the game of chess, 16 white and 16 black. Each of the two "colours" contains 1 king, 1 queen, 2 rooks, 2 bishops, 2 knights and 8 pawns. So, up to how many queens (black or white) can one place on the chessboard, with there being no possibility of capturing themselves? The answer is maximally n, since each queen has to be placed in its own column. But, for instance, the queens set on the board must also necessarily be placed on different diagonals not to capture themselves. The black queens must be at least a knight move away from the white ones and two queens of the same colour are at least one move further ahead from a knight´s move (since a knight´s move always leads to the opposite colour). Also, the total number of possible knight moves as a further example of a Hamiltonian paths problem on the chessboard, is already known. To find a possible algorithm for the queens completion problem or to see there is none could look like the following.

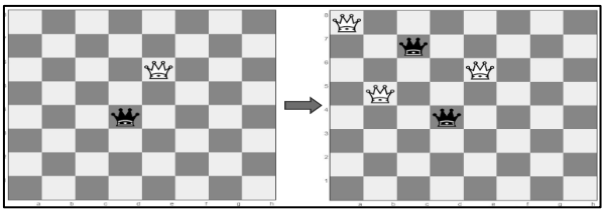

**Fig. 6:** The initial configuration on the left cannot be "completed" this way since the five queens on the right already cover the entire board

*Queens Completion Problem*

Can there n-m queens (black and white) be added to a position where m queens have already been placed within an n × n chess-like squared board? They should have no possibility of capturing each other. Is there one position of queens *y* which “completes” a given initial configuration *a* (of queens):

1. The NTM could do as follows: as soon as there are as many white as black queens in the initial configuration, go ahead with the colour: white. If there are more queens of one colour in the initial configuration, go ahead with this colour

2. See (a) and (b)

(a) Try to complete the board by adding just queens of the opposite color than 1.)
(b) If not possible, add one queen of color 1.) to the initial configuration and again try to complete it by adding queens of the opposite colour than 1.). Try all possibilities for that one queen

3. If not already terminated, try 2.) but by adding 2 queens of colour 1.), 3 queens, ..., up to n-m queens

Or maybe also the opposite strategy is leading to success within this problem: 1. Place up to 4 queens on the border of the board, the next up to 4 queens on the "second row border (new border of the board without its current border)" ...etc., until up to 1 queen within the 4 middle squares of the board. Below (Fig. 6.) is shown an initial configuration to complete and a first attempt to complete it (within the algorithm above, the time needed to find a solving configuration grows in general exponentially with the size of the board n):

## Materials and Methods

This study represents a rather theoretical and mathematical methodology. No materials are used except the ones already used in Silver *et al.* (2018) "Mastering chess and shogi by self-play with a general reinforcement learning algorithm".

## Results and Discussion

The major result p not equal np within this work is an outstanding discovery. Within the last decades, more and more "relevant information" has been released within this topic. For instance, Glock *et al.* (2022) “The n-queens completion problem", which at last made the whole proof

of p ≠ np possible was published just in 2022. The examples chosen within the proof are optimal and easiest possible: A DTM and an NTM which compute the sums of a random Matrix X and represent a counterexample to the equation at hand using the sets p and np, are not to observe any day. Within any "deterministic" problem which has no random components and which increases its running time cost exponentially in its parameters, one still could define a solving DTM´s transition function which calculates an exponential amount each application. But indeed, this setting would be inconvenient to prove p not np since, for all problems with random components (which are as well in these sets), the DTM would exactly be useless and the calculation of its transition function still just always predefined.

## Conclusion

We have shown successfully and mathematically that the sets p and np are distinct: P ≠ np. One of the so-called millennium problems claimed by the clay mathematics institute of Oxford has been solved with this study. We are very grateful to have been able to contribute and even to solve the entire problem.

## Acknowledgment

This work was funded by the University of Zurich, Switzerland. We want to thank to all who made this patiently possible: Prof. Dr. Reinhard Furrer, Prof. Dr. Regula Kyburz-Graber, Prof. Dr. Erwin Bolthausen and to the now emerited Leader of the Mathematics Institute of the University Prof. Dr. Andrew D. Barbour, whom also recommended the book: Everitt, B. S. (1999). “An R and S-Plus Companion to Multivariate Analysis, Springer Texts in Statistics” as introduction to simulation using R.

## Funding Information

The authors have not received any financial support or funding to report.

## Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

## References

Conrad, A., Hindrichs, T., Morsy, H., & Wegener, I. (1994). Solution of the knight’s Hamiltonian path problem on chessboards. *Discrete Applied Mathematics*, *50*(2), 125-134. https://doi.org/10.1016/0166-218x(92)00170-q

Everitt, B. S. (1999). *An R and S-Plus Companion to Multivariate Analysis, Springer Texts in Statistics* (1st Ed.), Springer London, XIII, 221. https://doi.org/10.1007/b138954

Gent, I. P., Jefferson, C., & Nightingale, P. (2017). Complexity of n-Queens Completion. *Journal of Artificial Intelligence Research*, *59*, 5512. https://doi.org/10.1613/jair.5512

Glock, S., Munhá Correia, D., & Sudakov, B. (2022). The n-queens completion problem. *Research in the Mathematical Sciences*, *9*, 41. https://doi.org/10.1007/s40687-022-00335-1

SEP. (2015). *Computational Complexity Theory*. Stanford Encyclopedia of Philosophy. https://plato.stanford.edu/entries/computational-complexity/?ref=https://githubhelp.com

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, ... (2018). A general reinforcement learning algorithm that masters chess, shogi and go through self-play. *Science*, *362*(6419), 1140-1144. https://doi.org/10.1126/science.aar6404

Squirrel, D., & Cull, P. (1996). *A Warnsdorff-Rule Algorithm for Knights Tours on Square Chessboards*. https://raw.githubusercontent.com/douglassquirrel/warnsdorff/master/5_Squirrel96.pdf