Notes van tevoren:

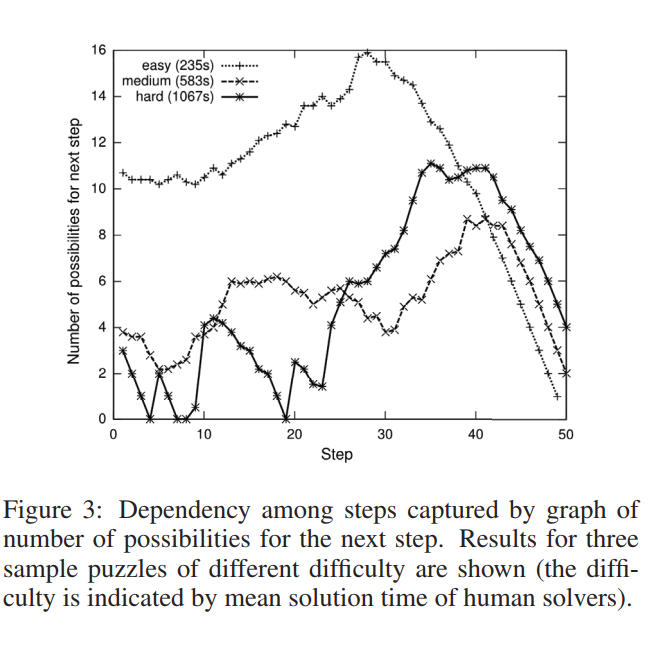
* Document is chaotisch: will explain the important stuff in person probs (most important things already discussed on whapp)
* Eerst heb ik mijn algemene aantekeningen
* Daarna info over heuristics
* Daarna info over sudoku
  + Strategies (easy 🡪difficult)
  + Differences easy 🡪 difficult

Dus combi van 1-3

Easy vs difficult sudokus

* Beste performance?
  + DP?
    - Ik denk idd langer duren (of andere parameter) voor difficult puzzles omdat er dan steeds meer bottlenecks komen (je kan maar één keuze maken, niet meerdere)
    - Terwijl voor ezpz puzzles je meerdere wegen naar Rome hebt 🡪 er zullen minder conflicten zijn
      * PLUS. Random variabele kiezen zal sneller tot succes leiden (want veel opties)
    - (the easy methods are basically DP?)
      * With this I mean: in easy sudokus heb je vaak rijen/columns met al 8 cijfers ingevuld 🡪 kan je heel makkelijk de 9e invullen
      * Maar dit wordt al opgelost in de simplification phase van DP
      * Dus DP is al ‘geoptimaliseerd’ voor easy puzzles in feite
  + 1 heuristic speciaal voor easy
  + 1 heuristic speciaal voor difficult
  + NOTE
    - Non chronological backtracking comes at a cost. This cost might only be worth it when you have a difficult puzzle?
  + Combi even? 1-2-3?
    - Always first 2, then do 3, then 2, then 3 idk
    - “Once you’ve satisfied yourself that there aren’t any easy methods you can apply to move forward,”

Or…

* Easy: non-chronological backtracking
  + - Not so sure about this one yet.
    - Het kan het makkelijker maken…
    - Maar ook moeilijker.
    - Hier moet ik nog over nadenken.
  + Difficult: the clause-based heuristic
    - Moeilijke sudokus hebben weinig opties. En meer bottlenecks. Dus meer punten waarop je maar één juiste keuze kan maken om naar de volgende te gaan.
    - De CBH zou daarom handig kunnen zijn. Deze neemt de meest recente beslissingen/conflicts in acht en bouwt daarop door (ipv naar de oudere conflict clauses te kijken). Het is dus dynamischer.
* (explanation for these later)
* AANTAL GEGEVEN CIJFERS GELIJK PLS
  + Future research?
* Indien tijd over
  + Kijk naar verschillende gegevens
* NOTE
  + Als we idd kiezen voor iets van non chrono backtracking…
  + Hoeveel extra conflict clauses willen we? 🡪 zoek in literatuur voor iets dat adequaat lijkt (%)
* Some scientific papers
  + <https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS11/paper/viewFile/2517/3077>
  + Same as
  + <https://pdfs.semanticscholar.org/f0ff/d2aa27b1699c7ba866371cf00f3c60e18a88.pdf>
    - Difficulty Rating of Sudoku Puzzles by a Computational Model∗
    - Abstract
      * We discuss and evaluate metrics for difficulty rating of Sudoku puzzles. The correlation coefficient with human performance for our best metric is 0.95. The data on human performance were obtained from three web portals and they comprise thousands of hours of human solving over 2000 problems. We provide a simple computational model of human  
        solving activity and evaluate it over collected data. Using  
        the model we show that there are two sources of problem  
        difficulty: complexity of individual steps (logic operations)  
        and structure of dependency among steps. Beside providing  
        a very good Sudoku-tuned metric, we also discuss a metric  
        with few Sudoku-specific details, which still provides good  
        results (correlation coefficient is 0.88). Hence we believe that  
        the approach should be applicable to difficulty rating of other  
        constraint satisfaction problems
    - Content
      * Classical Sudoku can be easily solved computationally by backtracking search (Pelanek 2011). However, backtracking ´ search is not very relevant to estimating human performance. Another approach to solving CSP is constraint propagation.
      * Another approach to solving CSP is constraint propagation. This method tries to to find values of (some) variables by reasoning about constraints. For each variable xi we define a current candidate set – a set of such values that do not lead to direct violation of any constraint (see Figure 1). By reasoning about candidate sets and constraints we can often derive solution without any search
      * Constraint propagation is not guaranteed to find a solution, but it may be more efficient than backtracking search and can also be combined with backtracking search to produce superior results. We are interested in constraint propagation particularly because this is the natural way how humans try to solve CSPs
      * 
      * So….
        + Easy puzzles have way more possibilities than difficult puzzles
        + Difficult puzzles first have very very few possibilities, and it then increases (eventually slightly beyond easy… but this takes some time)
        + Medium stays relatively same throughout, kinda binominal distribution though
        + So….

Maybe the non chrono backtracking best for difficult at first, and then matters a bit less…

(as in, because there are so few possibilities, it’d be quite easy to get to a dead end of the search space, compared to easy puzzles)

And…

Because easy sudokus have so many possibilities almost always…

Benefits most from choosing a strategy that appears in the most clauses?

**Types of heuristics**

* Choosing which conflict to go to
  + VSIDS
    - “Start with all counters initialized to 0.
    - On conflict, increase the counter of all variables involved in the conflict by cadd.
    - Every j conflicts, decrease the counter of all variables by multiplying it with coefficient cdecay.” (<https://codingnest.com/modern-sat-solvers-fast-neat-and-underused-part-3-of-n/>)
  + There is also an adaptVSIDS variant
    - See above link. Decay value depends on usefulness of learnt clauses
  + Or conflict history based…
  + Also mentioned in: <http://www.cs.tau.ac.il/~nachum/papers/CBH.pdf>
  + 31 Nachum Dershowitz , Ziyad Hanna , Alexander Nadel, A clause-based heuristic for SAT solvers, Proceedings of the 8th international conference on Theory and Applications of Satisfiability Testing, June 19-23, 2005, St Andrews, UK [doi>10.1007/11499107\_4]
    - Our proposal, which we call the **clause-based heuristic (CBH),** maintains a clause list containing both the initial and the conflict clauses, thus increasing the chances of picking interrelated variables. The next decision literal is picked from the top-most unsatisfied clause. No secondary heuristic is required. We propose various methods of initially organizing the clause list and for moving clauses within it. Our approach results in a significant performance boost over both VSIDS and Berkmin’s heuristic
    - Method:
      * In our clause-based heuristic (CBH), all clauses (both the initial and the conflict clauses) are organized in a list. After each conflict, the conflict clause is prepended to the top of the list. Conflict-responsible clauses, that is, clauses visited during 1UIP conflict-clause identification, are placed just after the new conflict clause. The next decision literal is picked from the topmost unsatisfied clause of the list. One can see that CBH is highly dynamic, since recently visited clauses are placed at the top of the list. Also, CBH organizes the list in such way that clauses that were responsible for a recent conflict are placed together. Hence, when one picks a series of decision variables after backtracking, it will tend to embrace interrelated variables. Indeed, when literals are picked from the same clause they must be related, even if the clause is an initial clause. When literals are picked from closely-placed clauses, they also tend to be related, since the list is organized in such a way that interrelated clauses are near each other, by placing conflict clauses at the top and moving conflict-responsible clauses towards the top.
      * Choosing the literal itself by:
        + Making counter of occurrence literal in ALL clauses (occasionally = divided by 2)
        + Making counter of occurrence literal in recent clauses
      * CBH chooses the decision variable from the topmost unsatisfied clause using the following algorithm: A variable p with maximal lcv(p) is chosen, so as to give preference to variables that participated in recent conflicts. Ties are broken by preferring variables with maximal global score gcv(p). According to the next criterion, variables that used to have the maximal decision level when assigned the last time are preferred. (If there still is a tie, it is broken by picking the lexicographically smallest variable.)
  + Berkman
    - Might be good, it’s dynamic
      * Chooses from clauses most recently used
      * Could be useful! Because in sudokus the rules kind of build upon each other
      * **🡪 maybe good for the difficult puzzles too actually… Since one decision needs to be made per step only. So the one you JUST solved MUST give information you need to solve the rest of the puzzle?**
    - DF
  + Kies de clauses met meeste false waarden (want easy strategy in sudoku)
    - Disadvantage: one clause at a time? (Semi?) Is it efficient enough?
  + Choose the one with the most filled in values that are false
    - For instance, -1 -2 -3 -4 5 6 -7 -8 -9 would be a clause that would be chosen first
    - So it’s a variation on the CP/CN theme
    - It is mentioned in the Nachum paper… sort of
      * “As a variant, CBH can also move clauses found to have exactly two unassigned literals during BCP to the top of the list. We refer to this strategy as 2LitFirst. The added value of this strategy is that: (1) more implications are learned during BCP; (2) short and potentially contradictory clauses tend to be immediately satisfied. The first point guides the solver to find conflicts in an unsatisfiable area, and the second one is useful to eliminate conflicts in a satisfiable area. The disadvantages of 2LitFirst are that: (1) it tends to separate between clauses that contain interrelated variables; (2) it may promote clauses that have never been responsible for conflicts.”
      * But… Is not that effective:
        + “**We found experimentally that while usually 2LitFirst hurts performance, it may be helpful for instances having high clause/variable ratio**. This can be explained by the fact that, in instances having a high clause/variable ratio, variables tend to appear in a greater number of clauses, since there are fewer variables per clause overall. Hence, two chains of decisions taken using different decision strategies tend to contain more common variables. This gives more weight to the order between variables and the local context of the search. One should prefer variables whose assignment can have an immediate impact; this is exactly what 2LitFirst does. The default version of CBH invokes 2LitFirst on instances where the clause/variable ratio exceeds 10”
  + Neural networks..
    - “Abstract: Solvers for the Boolean satisfiability problem are an important base technology for many applications. The most efficient SAT solvers for industrial applications are based on the DPLL algorithm with clause learning and conflict analysis dependent decision heuristics. The solver MINISAT V1.14 was modified to use a neural-net-based decision heuristic and search strategy. The weights and biases of the multilayer feedforward neural net are generated by an evolution strategy which is trained on a sample set of SAT problems. Problems solved with the evolved solutions encounter a similar number of conflicts as the original program, but require a higher number of decisions. View less
    - Metadata
    - Abstract:
    - Solvers for the Boolean satisfiability problem are an important base technology for many applications. The most efficient SAT solvers for industrial applications are based on the DPLL algorithm with clause learning and conflict analysis dependent decision heuristics. The solver MINISAT V1.14 was modified to use a neural-net-based decision heuristic and search strategy. The weights and biases of the multilayer feedforward neural net are generated by an evolution strategy which is trained on a sample set of SAT problems. Problems solved with the evolved solutions encounter a similar number of conflicts as the original program, but require a higher number of decisions.” (<https://ieeexplore.ieee.org/document/4371054>)
  + Learning Rate Based Branching Heuristic for SAT Solvers
    - <https://cs.uwaterloo.ca/~ppoupart/publications/sat/learning-rate-branching-heuristic-SAT.pdf>
    - Is also mentioned in the codingnest.com link
    - LRB in general turns up in many publications
    - In this paper, we introduce a general principle for designing branching heuristics wherein online variable selection in SAT solvers is viewed as an optimization problem. The objective to be maximized is called the learning rate (LR), a numerical characterization of a variable’s propensity to generate learnt clauses. The goal of the branching heuristic, given this perspective, is to select branching variables that will maximize the cumulative LR during the run of the solver. Intuitively, achieving a perfect LR of 1 implies the assigned variable is responsible for every learnt clause generated during its lifetime on the assignment trail. We put this principle into practice in this paper. Although there are many algorithms for solving optimization problems, we show that multi-armed bandit learning (MAB) [31], a special-case of reinforcement learning, is particularly effective in our context of selecting branching variables. In MAB, an agent selects from a set of actions to receive a reward. The goal of the agent is to maximize the cumulative rewards received through the selection of actions. As we will describe in more details later, we abstract the branching heuristic as the agent, the available branching variables are abstracted as the actions, and LR is defined to be the reward. Abstracting online variable selection as a MAB problem provides the bridge to apply MAB algorithms from the literature directly as branching heuristics. In our experiments, we show that the MAB algorithm called exponential recency weighted average (ERWA) [31] in our abstraction surpasses the VSIDS and CHB branching heuristics at solving the benchmarks from the 4 most recent SAT Competitions in an apple-to-apple comparison. Additionally, we provide two extensions to ERWA that increases its ability to maximize LR and its performance as a branching heuristic. The final branching heuristic, called learning rate branching (LRB), is shown to dramatically outperform CryptoMiniSat [29] with VSIDS.
    - ………….
    - Uhm
    - <https://cs.uwaterloo.ca/~ppoupart/publications/sat/learning-rate-branching-heuristic-SAT.pdf>
    - Too much to summarise
* Dealing with conflict clauses
  + “Let's start with a very naive strategy, *first in, first out* (FIFO). In this strategy, we decide on an upper limit of learnt clauses, and when adding a newly learnt clause exceeds this limit, the oldest learnt clause is deleted. This strategy avoids the problem with the ballooning number of learnt clauses, but at the cost of discarding potentially useful clauses. In fact, we are guaranteed to discard useful clauses because every learnt clause has a deterministic lifetime.
  + Let's consider a different naive strategy, *random removal*. In this strategy, we again decide on an upper limit of learnt clauses, but this time the clause to remove is picked completely randomly. This has the advantage that while we *might* remove a useful clause, we are not *guaranteed* that we remove useful clauses. While this distinction might seem minor, the random pruning strategy usually outperforms the FIFO one.” <https://codingnest.com/modern-sat-solvers-fast-neat-and-underused-part-3-of-n/>
  + Deletion based on clause usefulness
    - Activity
    - Etc.
* Restarts
  + Keep learnt conflict clauses, but start all over for the rest (variables)
  + And keeps last assigned truth value of each variable and assigns same value next time they are picked to be assigned
  + An aggressive example = glucose restarts
    - Instead of waiting for N amount of conflicts 🡪 restart when last few learnt clauses are not that high quality
  + “So what restart strategy is the best? There only correct answer is neither because while glucose restarts have been very successful in SAT competitions, they are heavily optimized towards the handling of industrial (real world problems encoded as SAT) unsatisfiable instances at the expense of being able to find solutions to problems that are actually satisfiable. In a similar vein, the Luby restarts heavily favor finding solutions to satisfiable industrial instances, at the expense of finding solutions to problems that are unsatisfiable[21].
  + In practice, the current state of the art sat solvers use various hybrids of these techniques, such as switching between periods with glucose restarts and Luby restarts, where the lengths of the periods increase geometrically, or switching between glucose restarts and running without any restarts, and so on. There have also been some experiments with machine learning to restart” (<https://codingnest.com/modern-sat-solvers-fast-neat-and-underused-part-3-of-n/>)
  + Machine learning
    - <https://docs.google.com/a/gsd.uwaterloo.ca/viewer?a=v&pid=sites&srcid=Z3NkLnV3YXRlcmxvby5jYXxtYXBsZXNhdHxneDo0Njg5YjZhM2JjNDhkZTJh>
    - (example)
* Preprocessing/inprocessing
  + E.g. self subsumption
  + Aka
  + Just a bunch of extra ‘simplification’ steps to make the amount of clauses more bearable/limited in number
* Backtracking
  + Non-chronological

Or…

* Easy: non-chronological backtracking
* Difficult: the clause-based heuristic

**Verschillen tussen sudokus:**

Method

* <http://www.aisudoku.com/index_en.html>
* Kan sudokus maken/raten op difficulty level
* Dus als we mooie curve willen van makkelijk 🡪 moeilijk = handig
* Anderzijds dichotomie easy/difficult (might still be good to do for sure)
  + Dan kunnen we low key manier hebben om kinda te quantifyen dat de sudokus in onze test set makkelijk of moeilijk zijn
  + (want er is een set genaamd damn hard online, maar geen specifieke easy puzzles, so we need to either look for some or use this to classify 🡪 this might be better because classification will be done the same way for all puzzles)
* Funnily enough, I think this works using backtracking (bias though?)

Causes of difficulty

* “There is no connection between the difficulty of a Sudoku puzzle and the number of clues it contains. “ <https://www.conceptispuzzles.com/index.aspx?uri=info/article/2>
  + It is the number of logic decisions, their redundancy and their complexity which determines how hard the puzzle is going to be.
  + Take Ultra Easy, Very Easy and Easy Conceptis Sudoku puzzles for example. They require just the simplest “Scanning” solving techniques as described in our website under Sudoku >> Solution Examples. There is no need to count missing numbers in rows and columns, to note down pencilmarks, or to make any assumptions.
* Assumption logic
* Medium and Medium Plus puzzles require the use of more advanced techniques. Here you need to use pencilmarks as well as count rows and columns, checking the numbers to see which ones are missing and where they can be placed. Again, the difference between Medium and Medium Plus is the frequency of logic decisions and the number of bottlenecks encountered during the solution process.
* A bottleneck is a situation in the solution flow where only a single move is possible to continue the progress of the puzzle. Thus, if a puzzle has many bottlenecks then it becomes harder to solve even if the solving techniques are easy. M

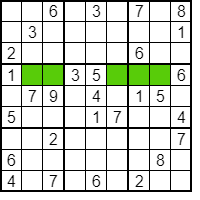
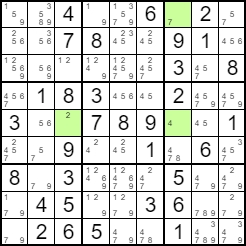
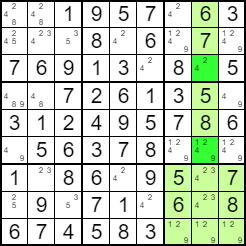
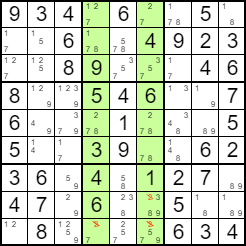
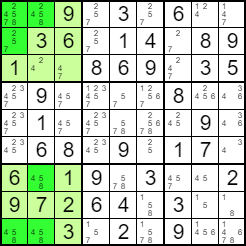
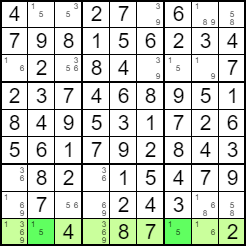
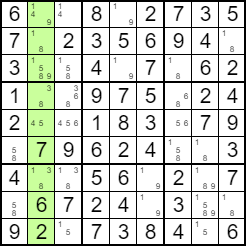
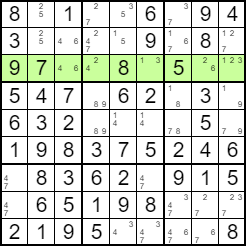
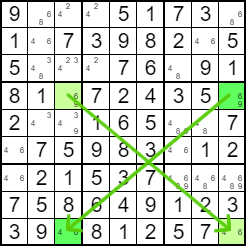
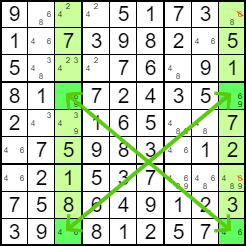
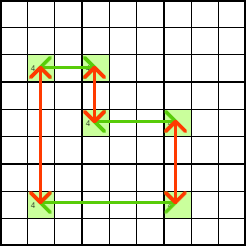
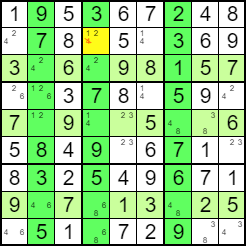
So…

* Easy ones
  + Many decisions can be made, can be made quickly, ‘scanning’
* Medium
  + At some point you have to make the exact right decision to progress further, cannot take multiple approaches to solve it. One good answer.
  + Includes pencil marking, row/column counting, scanning
* Hard
  + Now you have to presume a cell contains a number and check if this could be valid or not

So far:

* Maybe DP works quite well for hard puzzles actually 🡪 it’s just ‘trying something’ and seeing whether it works or not
  + With the non-chronological dynamic backtracking perhaps
* Maybe for easy puzzles…
  + You can do the CP/CN thing (although primarily for medium I guess)

Sudoku strategies

* <http://www.sudokuwiki.org/Strategy_Families>
* <http://www.sudokuoftheday.com/techniques/>
  + Easy stragies
    - Single position/hidden single
      * Vanwege posities van andere waardes is er maar één plek waar een nummer in een rij/column kan staan (dus ezpz invullen)
      * 
      * E.g. een 7 in groene lijn kan maar op één plek
      * Note:
        + Hoe meer waardes er al in een gebied zijn, hoe hoger de kans op succes
    - Single candidate
      * Rule out possibilities of values in one cell 🡪 only one left? Fill it in
      * 
      * (note: zouden dit unit clauses worden?)
    - Pencilmarks (the grey numbers of options)
      * Just makes life easier
      * NOTE:
        + Could fill in ALL pencilmarks from the start and use that…
        + And maybe then start the split procedure with the ones that have the fewest available options?
      * Note: maybe more of a human method? Useful for humans but not as much for computers? Idk. Humans COULD track the pencilmarks automatically though. (but efficiently?)
  + Medium techniques
    - Candidate lines
      * Helps determine places where you CANNOT put numbers
      * 
      * 🡪 4 must be in the middle line of right grid even though the exact box is still uknown (because of the bottom line), so it cannot be in the dark green spaces 🡪 remove these
    - Double pairs
      * 
      * If candidates in two grids are only on two lines 🡪 rule out those 2 lines from remaining grid (bottom)
    - Multiple lines
      * 
      * Same as previous but with multiple lines within 1 grid
  + Advanced techniques
    - Naked pairs/triples
      * 
      * Two cells with as only options 1 and 5 🡪 thus 1 and 5 cannot be in other cells
      * Triplet:
      * three cells which between them **must** contain 1,3,8 and **no others!** Because they must contain just those three values, it means you can remove them as candidates from other cells
      * 
      * Can also have quads
    - Hidden pairs/triples (or: hidden/unique subset)
      * 
  + Master techniques
    - X-wing
      * 
      * E.g. putting 6 left right forces right top to be 9 and left bottom to be 4. That again forces a 6 into bottom right.
      * 
      * Use it to remove candidates
    - Swordfish
      * 
      * 
    - Forcing chains
      * Similar to guessing, but …
      * Not quite – what you’re doing is simultaneously looking at the implications of either choice, and seeing if any other cells will turn out the same whichever your choice would be. If you were to have just guessed one, and worked from there, you would actually fill in the same result for the second cell, but depending on whether you guessed correctly or not, you might have made a whole batch of mistakes on the way.
* What makes this method hard is that you might have to follow chains a long way, and you will have a lot of testing to do. Longer chains don’t make it conceptually any harder, but they do make it more likely that you’ll make mistakes along the way.
* Sticking to working with just the pairs generally keeps it fairly simple, but there’s nothing stopping you considering the effects of triples or other techniques as you go!
  + Extreme (involve guessing, not necessarily logic!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!)
    - Nishio
      * Nishio is a form of guessing, where you look for a guess causing a contradiction, meaning that you can rule it out. Going on from this, it is possible to solve entire Sudoku puzzles from guesses alone, but it can take a long time!
      * Not always – but often! Sometimes you’ll see the contradiction (error) within just a couple of placements, sometimes you’ll get nearly to the end.
      * It is worth reminding here that Nishio only works by finding a contradiction, from which you know to pick the other option. Only if it actually leads you to complete the grid can you be sure that you had the right option to begin with. If you can only fill in a few cells, and there are still more to determine, then it isn’t enough for you to be sure. *A positive isn’t proof, but a negative****is****a disproof!*.
    - Guessing
      * Finally, and here’s the worst part, one guess might not be enough! It might be that along the route to completion you have to make several guesses, each one leading you down a different path with different choices to make. Better have an eraser handy!
      * If multiple guesses are required, you’ll find yourself needing to track back if you find you did make a wrong turn (and end up in a dead end). For that reason, a guess-based technique that follows from Nishio is known as “Ariadne’s Thread” – meaning following guesses, but each time you find an error backtrack to your last choice, and take a different path, like Ariadne of legend! Yes, you’ll eventually get to the end of the Sudoku, but it could take a very long time with a great deal of wrong turns! Working things out with logic is much simpler when you can! *(Ariadne was the daughter of the Cretan King Minos, who helped Theseus by giving him a sword with which to kill the Minotaur, and a thread which he used to find his way back out of the labyrinth, winding the thread back up every time he made a decision ending in a dead-end, and taking a different path.)*
      * The technique of guessing (or trial and error) is also known as bifurcation – and many computer based solvers only include this technique! That may seem strange, but it is very easy for a computer program to *brute-force* run through each of the guesses to complete the puzzle, and trivial for it to backtrack to a previous choice – humans just don’t work this way! (But then, we can make deductive leaps of logic that computers can’t… yet!)

Easy to difficult

* Differences
  + Technique difficulty necessary to solve it
    - Modern sudoku puzzles are ranked according to the difficulty of the techniques required to generate a solution. It's important to note that there is no standardized system or metric, but rather, these are based on qualitative assessments.
    - [playr.co.uk](http://www.playr.co.uk/sudoku/ratings.php) lists these requirements for its various ratings:
    - Level 1 puzzles can be solved simply by reducing the possibilities of a square to one, and filling it in.
    - Level 2 puzzles use the occurrence of pairs of numbers within a block (cell, row, or column). This includes pairs of numbers, number chains, and hidden pairs and chains.
    - Level 3 uses the [X-wing](http://www.janko.at/Raetsel/Sudoku/Loesung/X-Wing1.jpg) and [Y-wing](http://www.brainbashers.com/gifs/sudokuxywing2.gif) techniques, as well as
    - Level 4 typically requires Nishio or Forcing Chains (which effectively involve making assumptions and tracing their results)
    - Level 5 requires flat-out trial and error or lookup tables.
    - Aka from ezpz strategies to flat out guessing
* Easy sudokus…
  + Easy strategies honestly look like the simplification steps
    - E.g. if there’s just one possibility in a line that’d be the simplification step

‘although proper sudokus may not necessitate guessing in order to complete the puzzle, progressively more difficult techniques would need to be utilised in comparison to easily solvable sudokus. Nevertheless, several of these more difficult techniques require at least a degree of guesswork or backtracking even if guessing is not done outright. Therefore, …….’

* Furthermore: er is dat ding met dat er weinig opties tegelijkertijd zijn voor difficult puzzles (compared to easy) 🡪 makes backtracking strategy especially important
* 🡪 this text is old