Sudoku and Other Related Problems

E. Routledge

01 Nov 2022

1 Introduction

Sudoku is a simple logic game, in the standard 9×9 (or $3 \times 3 \times 3 \times 3$) one must complete the grid such that every row, column and box contains the numbers 1 to 9, that is all, yet it is filled with mathematics. Through sudoku we can explore the connections between various areas of maths: complexity theory, graph theory, group theory and information theory.

1.1 History

1.2 Defining Sudoku Notation

Def^n: A valid sudoku puzzle is a function $S : i, j \to x$ for values i,j $\in \{1, ..., D^2\}$ and $x \in \{0, ..., D^2\}$ satisfying the following:

- for all $a, b, c \in \{1, ..., D^2\}$ with $S(a, b) \neq 0$ and $S(a, c) \neq 0$, then $S(a, b) \neq$ S(a,c)
- for all $a, b, c \in \{1, ..., D^2\}$ with $S(a, b) \neq 0$ and$S(c, b) \neq 0$, then $S(a, b) \neq$ S(c,b)
- for all $a, b, c, d \in \{1, ..., D^2\}$ with $amodD = cmodD$, $bmodD = dmodD$, $S(a, b) \neq 0$ and $S(c, d) \neq 0$, then $S(a, b) \neq S(a, c)$

Def^n: A completed sudoku puzzle is a function $S : i, j \to x$ as above but with the added condition that $x \neq 0$.

2 Classic solving techniques

Def^n: A forced cell is a value pair (a, b) such that $S(a, b)$ can only be a single value call this x as $\{1, ..., D^2\}/\{x\}$ are already present in $S(a, j)$ for $j \in$

$\{1, ..., D^2\}/\{b\}$ or $S(i, b)$ for $i \in \{1, ..., D^2\}/\{a\}$ or $S(i, j)$ where $amodD = imodD$ and $bmodD = jmodD$.

define x wing define y wing

3 Sudoku is Hard

Let's imagine a sudoku of size $D^2 \times D^2$. How big does D have to be for you to need more than a day to solve it? Maybe 6 or 10 or even just 4. Don't worry if you said a smaller number than your friends, this has nothing to do with your problem solving skills, even a computer finds sudoku hard. In fact just incrementing D by 1 leads to an exponential increase in compute time and the most optimal algorithms for solving sudoku are infeasible for 100×100.

We prove sudoku's hardness by transforming it into a known 'difficult' problem; we will use SAT, a problem that has plagued computer scientists for decades.

3.1 Computational Complexity an Introduction

For those with a mathematical mind, outraged by the lack of definitions of 'difficulty' and 'hardness', let's take a detour into complexity theory.

Def^n: Let f be a function indicating the execution time for an algorithm and g a strictly positive function. $f(x) = O(g(x))$ if $\exists$ positive M and x_0 such that $|f(x)| \leq Mg(x)\ \forall\ x \geq x_0$.

Def^n: A **Reduction**, $A \leq_p B$, is a transformation in polynomial time $(O(x^c))$ from problem A to B.

Def^n: A **Turing Machine** is the mathematical model of a CPU.

What is NP completeness? We care about decision problems, these are problems that given an input produce a 'yes' or 'no' answer. We will discuss three sets of these problems:

- P is the the class of problems that can be solved in polynomial time (if the input is order n then the program halts in order n^2 steps) by a Turing machine;
- NP is the class of problems that can be verified in polynomial time and solved in polynomial time by a non-deterministic Turing machine;
- the NP-complete set has problems that any NP problem can be reduced to in polynomial time.

Problems in P are considered feasible and those in NP are infeasible as their complexity scales exponentially with respect to the input size and as it is assumed they cannot be solved in polynomial time ($P \neq NP$) and are therefore infeesible for large inputs. We can only assume that $P \neq NP$ as this problem is yet to be proven, it is in fact one of the Millennium Prize problems.

So when we state sudoku is hard we are actually saying sudoku belongs to NP.

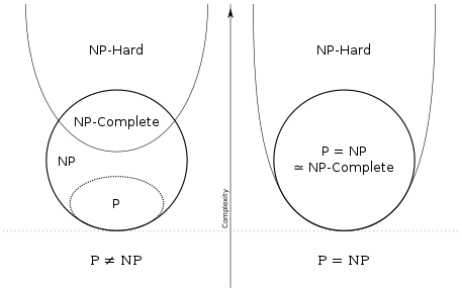

Figure 1: P, NP, NP-complete & NP-hard sets [1]

How to prove NP completeness generally? Call the problem we wish to prove is NP-complete x. First show there exists a verifier for x with a polynomial or less runtime, this is a algorithm that decides if a proposed solution to problem x is correct. Then reduce a known NP-complete problem call this y to x, one does this by transforming the input of y to the input of x in polynomial time. If there exists a polynomial time algorithm to solve x we could solve$y$ in polynomial time too, this implies P=NP, a contradiction. Therefore a polynomial time algorithm does not exist for x.

Our base NP-complete problem. If, as the above suggests, we require a NP-complete problem to prove a problem is NP-complete then we seem to have reached a paradox. Luckily we have the Cook-Levin Theorem.

Cook-Levin Theorem: SAT is NP-Complete

Proof: TO COMPLETE: CITE

Def^n: SAT is the following decision problem. Given a set of boolean variables B and a collection of clauses C does a valid truth assignment exist that satisfies C?

We now have a NP-complete problem to reduce other problems to.

Let's make this more intuitive with examples.

3.2 Verification is Easy

Verificaiton decision problem, "is the sudoku puzzle complete?":

$$\Psi(S(,)) = \begin{cases} \text{True if the puzzle is complete} \\ \text{False if the puzzle is not complete.} \end{cases} \tag{1}$$

There exists an algorithm to do this in polynomial time with respect to the dimensions of the grid.

1. For each row in the grid check there exists no repeated numbers. $O(n^2)$
2. For each column in the grid check there exists no repeated numbers. $O(n^2)$
3. For each box in the grid check there exists no repeated numbers. $O(n^2)$

If all tests pass return True else return False. This algorithm has complexity of $O(n^2 + n^2 + n^2) = O(3n^2) = O(n^2)$, this is polynomial and therefore $\Psi(S(,)) \in P$.

3.3 Finding a Solution is Hard

Finding a solution to sudoku is NP-complete, let us define the decision problem:

$$\Phi(S(,)) = \begin{cases} \text{True if a completion exists} \\ \text{False if a completion does not exist.} \end{cases} \tag{2}$$

Our question now is does there exist a function Φ that when given an instance of the problem will, in polynomial time or less, return True if it can be solved and False otherwise.

3.3.1 Proof

The **verifier** is $O(n^2)$, as will be seen in the above subsection 'Validation is Easy'.

Now we need a **reduction** from sudoku to a known NP-complete problem to prove sudoku is at least as hard. We will be creating a chain of reductions: Sudoku $\geq_p$ Latin Square $\geq_p$ Triangulated Tripartite $\geq_p$ 3SAT and then prove 3SAT is NP-complete.

3.3.2 Sudoku $\geq_p$ Latin Square

Def^n: A valid Latin Square puzzle is a function $L : i, j \rightarrow x$ for values $i, j \in \{1, .., D\}$ and $x \in \{0, ..., D\}$ satisfying the following:

- for all $a, b, c \in \{1, ..., D\}$ with $L(a, b) \neq 0$ and $L(a, c) \neq 0$ then $L(a, b) \neq L(a, c)$
- for all $a, b, c \in \{1, ..., D\}$ with $L(a, b) \neq 0$ and $L(c, b) \neq 0$ then $L(a, b) \neq L(c, b)$

It is complete or solved if for all $i, j \in \{1, ..., D\}$, $L(i, j) \neq 0$.

By observation we see this is a superset of the sudoku puzzle, we just add the restrictions that the dimension must be a square number and also add the third property of the sudoku puzzle defintion.

What is the Latin Square decision problem? Given a latin square puzzle $L(,)$, can the function be augmented, by changing only the value of the function for value pairs i, j that previously gave $L(i, j) = 0$, to get a complete latin square puzzle?

Proof idea: We must reduce a given latin square grid of size $D \times D$ to a sudoku grid size $D^2 \times D^2$ that is solvable iff the Latin square is.

Lemma: Let S_l be a Sudoku problem with the following construction

$$S_l(i, j) = \begin{cases} 0 & \text{when } (i, j) \in L_s \\ ((i - 1 \bmod n)n + \lfloor i - 1/n \rfloor + j - 1) \bmod n^2 + 1 & \text{otherwise} \end{cases} \tag{3}$$

where $L_s = \{(i, j) | \lfloor i - 1/n \rfloor = 0 \text{ and } (j \bmod n) = 1\}$. Then there exists an augmentation S'_l to complete the sudoku puzzle if and only if the square L such that $L(i, j/n) = S'_l(i, j) - 1/n + 1$ for all $(i, j) \in L_s$ is a Latin square.

Proof:

First we must show $S_l(i, j) = ((i-1 \bmod n)n+\lfloor i - 1/n \rfloor+j-1) \bmod n^2+1$ forms a complete and valid sudoku puzzle.

When $i = [1, ..., n^2]$ then:

$$0 < \lfloor i - 1/n \rfloor < n - 1 \tag{4}$$
$$0 < i - 1 \bmod n < n - 1 \tag{5}$$
$$0 < (i - 1 \bmod n)n + \lfloor i - 1/n \rfloor < n^2 - n \tag{6}$$
$$0 < (i - 1 \bmod n)n + \lfloor i - 1/n \rfloor + j - 1 < n^2 - 1 \tag{7}$$
$$1 < ((i - 1 \bmod n)n + \lfloor i - 1/n \rfloor + j - 1) \bmod n^2 + 1 < n^2 \tag{8}$$
$$1 < S_l(i, j) < n^2 \tag{9}$$

Note $\lfloor i-1/n \rfloor$ gives the row coordinate when indexed at 0 in which the larger box that (i,j) belongs to starts and $i-1 \bmod n$ gives the row within that box when indexed at 0. Therefore $(\lfloor i-1/n \rfloor, i-1 \bmod n)$ will take all value pairs of integers between 0 and $n-1$.

When j is fixed (particular column), assume two cells have the same value, that is $S_l(i,j) = S_l(i',j)$ then

$$(i-1 \bmod n)n + \lfloor i-1/n \rfloor + j - 1 = (i'-1 \bmod n)n + \lfloor i'-1/n \rfloor + j - 1 \tag{10}$$

$$(i-1 \bmod n)n + \lfloor i-1/n \rfloor = (i'-1 \bmod n)n + \lfloor i'-1/n \rfloor \tag{11}$$

from the above $i = i'$. No cell on a column has the same value.

When i is fixed (particular column) assume two cells have the same value, that is $S_l(i,j) = S_l(i,j')$ implies $j-1 = j'-1(modn)$ therefore $j = j'$.

For the third condition fix $\lfloor i-1/n \rfloor$. (i-1 mod n,j) takes all value pairs of integers 0 to n-1 so if a cell has the same value as another within the n by n square $S_l(i,j) = S_l(i',j')$ implying $(i-1 \bmod n, j) = (i'-1 \bmod n, j')$ which means $i = i'$ and $j = j'$. Therefore S_l is a valid and complete sudoku puzzle.

Now consider which integers fill the blanks in L_s. For $(i,j) \in L_s$, $S_l(i,j) - 1 = ((i-1 \bmod n)n + j - 1) \bmod n^2$ as $jmodn = 1$, $j-1modn = 0$ therfore $S_l(i,j) - 1$ is divisible by n so $S_l - 1/n + 1$ gives integers between $[1, ..., n]$. Therefore $L(i,j) \in [0, ..., n]$.

We must validate the Latin square conditions. The row constraint in S_l ensures $S'(i,j) = S'(i,j') \implies j = j'$, $S'(i,j) - 1/n + 1 = S'(i,j') - 1/n + 1 \implies j = j'$, $L(i,j/n) = S'(i,j'/n) \implies j = j'$ is equivalent to the row constraint of L. The column constraint of S_l is equivalent to the column constraint of L. The small square constraint of S_l is equivalent to the column constraint of L. □

3.3.3 Latin Square $\geq_p$ Triangulate A Tripartite Graph

Def^n: A graph $G = (V, E)$ is tripartite if a partition V_1, V_2, V_3 exists such that the vertices are split into three sets with no edges between vertices that belong to the same set, i.e for all $(v_i, v_j) \in E$ if $v_i \in V_i$ then $v_j \notin V_i$.

Def^n: A triangulation T of a graph is a way to divide edges into disjoint subsets T_i, each forming a triangle ($T_i = \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$).

If a tripartite graph can be triangulated it must be uniform, that is: every vertex in V_1 (or V_2 or V_3) has the same number of neighbour in V_2 and V_3 (or the respective sets).

What is the Triangulated Tripartite decision problem? Given a graph G that is tripartite (can be split into 3 subgroup, within these subgroups vertices should not share edges) can it be triangulated ?

Theorem: Completing a Latin square with dimensions n by n is equivalent to triangulating a tripartite graph $G = V_1, V_2, V_3$.

Proof:

Observe completing a partial Latin square is equivalent to triangulating tripartite graph, we map the Latin square to this through the following: given tripartite graph G=(V,E) label vertices in V_1 with distinct lables $\{r_1, ...r_n\}$, label vertices in V_2 with distinct lables $\{c_1, ...c_n\}$ and label vertices in V_3 with distinct lables $\{e_1, ...r_n\}$. Add edges such that:

- If $L(i, j) = 0$ then add the edge (r_i, c_j)
- If for all $i \in [0, ..., n]$ and constant j, $L(i, j) \neq k$ then add the edge (r_i, e_k)
- If for all $j \in [0, ..., n]$ and constant i, $L(i, j) \neq k$ then add the edge (c_j, e_k)

This graph has a triangulation iff L(i,j) can be solved.

Let us show every uniform tripartite graph is the above formulation of a Latin square.

Defn: A Latin framework LF for tripartite graph G, size (r,s,t) is a r by s array with values [1,...,t]. With constraints:

- Each row/column contain each element only once.
- If $(r_i, c_j) \in E$ then LF(i,j)=empty else LF(i,j)= k $k \in [1, ..., t]$
- If $(r_i, e_k) \in E$ then for constant i $LF(i, j) \neq k$
- If $(c_i, e_k) \in E$ then for constant j $LF(i, j) \neq k$

If r=s=t then LF is a latin square which can be completed iff G has a triangle partition.

Lemma: For graph G=(V,E) with —V—=n, there's a Latin framework of (n,n,2n).

Define LF an n by n array. For $(r_i, c_j) \in E$ $LF(i, j) = 0$ else $LF(i, j) = 1 + n + ((i + j) mod n)$. LF is a latin framework. □

Lemma: Latin framework L (r,s,t) for uniform tripartite graph G. R(k) = the number of times k appears in L plus half $|e_k|$. Whenever $R(k) \geq r + s - t$ for $1 \leq k \leq t$, L can be extended to (r,s+1,t) to give L' in which $R'(k) \geq r + s + 1 - t$ for all $1 \leq k \leq t$.

proof of this lemma to be added

Lemma: Latin framework (r,s,s) can be extended to (s,s,s).

We can transpose the array and do the same as the previous lemma. □

Given a tripartite graph G, if it is not uniform then no triangulation exists, else we apply above to produce a latin framework of size (2n,2n,2n) in polynomial time. This is a Latin square which can be completed iff G has a triangulation. The latin square problem has been reduced to the triangulating a tripartite graph problem. □

3.3.4 Triangulated Tripartite $\geq_p$ 3SAT

What is 3SAT? With a set of boolean variables B and a collection of clauses C, with at most 3 literals (a literal is any $b \in B$ or its negation $\bar{b}$) in each, does a valid truth assignment exist that satisfies C?

$$\phi(C, B) = \begin{cases} \text{True if a truth assignment exists} \\ \text{False if a truth assignment does not exist.} \end{cases} \tag{12}$$

This decision problem is therefore an enforced limitation of SAT as defined in the subsection Computational Complexity an Introduction.

Proof:

This reduction is a little trickier as we need to introduce the Holyer graph H which dips a toe into topology as it is a torus.

Consider the graph $H_{3,p}$, a version of which can be seen in figure 2. Observe $H_{3,p}$ is a tripartite graph iff $p \equiv 0 \pmod 3$, see the 3-colouring (p refers to the 'height' and 'width').

$H_{3,p}$ has only two triangulations, termed a true and a false triangulation, we say $G = T$ if graph G has a true triangulation and $G = F$ if it has a false triangulation. We connect graphs together by taking a set of vertices in G_1 and making them the 'same' as a set of vertices in G_2, sets are the same size.

Lemma: Connecting two $H_{3,p}$ by two A-patches then our triangulations of these graphs can be of the form (T, T), (T, F) or (F, T). Then by removing the centre triangles from both graphs we get only the triangulations (T, F) or (F, T). If we expand this to x $H_{3,p}$ we get only one false triangulation and the rest true.

Transformation process: (select p large enough to prevent patch overlap)

- For $b_i \in B$ create $H_{3,p}$ called G_{b_i}.
- For all $c_j \in C$, for each literal $l_{i,j}$ $i \in [1, 2, 3]$ create $H_{3,p}$ called $G_{i,j}$.
- For $b_i \in B$ connect an A-patch to each $G_{i,j}$ when $l_{i,j} = b_i$ or a B-patch to $G_{i,j}$ when $l_{i,j} = \neg b_i$.

- For each $c_i \in C$ connect an A-patch from $G_{1,i}$, $G_{2,i}$ and $G_{3,i}$ then delete the centre triangle.
- $G = \{G_{b_i} \mid b_i \in B\} \cup \{G_{i,j} \mid c_j \in C \text{ and } i \in [1,..,3]\}$

Now we can take a 3SAT formula and convert it to the problem of triangulating a tripartite graph.

See figure 5 for an example. By the lemma; one of $l_{1,1}$, $l_{1,2}$ and $l_{1,3}$ must be a false triangulation and therefore the variable connexted to this false triangulation must be true, this satisfies the formula.

Triangulating a Tripartite Graph is hard. □

3.3.5 3SAT is NP-Complete

Proof:

Given a truth assignment t check each clause is satisfied, if all are satisfied return True else False, this algorithm is at most the length of C multiplied by the length of B. $O(BC)$ is polynomial, a polynomial verifier exists.

Given a SAT instance with the input sets of B and C. C is in conjunctive normal form (every clause set can be converted to an equivalent set in CNF form [2]) such that $\forall c \in C$ and for some $b_1, ..., b_n \in B$, $c = b_1 \vee b_2 \vee ... \vee b_n$. For each $c \in C$ with more than 3 literals we can transform these to a new set of clauses of length 3.

For $c = b_1 \vee b_2 \vee ... \vee b_n$ we introduce a new literal: a_1 to give $b_1 \vee b_2 \vee a_1$, $\bar{b_1} \vee a_1$, $\bar{b_2} \vee a_1$ and $a_1 \vee b_3 \vee ... \vee b_n$. Then $a_1 \vee b_3 \vee ... \vee b_n$ becomes $b_3 \vee b_4 \vee a_2$, $\bar{b_3} \vee a_2$, $\bar{b_4} \vee a_2$ and $a_1 \vee a_2 \vee b_5 \vee ... \vee b_n$. This continues at most $n/2$ times to give $a_1 \vee ... \vee a_{n/2}$ or $a_1 \vee ... \vee a_{n/2} \vee b_n$ if n is odd.

Because we can convert a clause larger than 3 into multiple clauses of at most 3 literals in linear time ($O(n/2 + n/4 + ...) = O(n)$) this means we can reduce SAT to 3SAT in polynomial time.

As SAT is NP-complete by the Cook-Levin Theorem, this proves 3SAT is NP-Complete. □

Alternative reduction Sudoku $\geq_p$ Graph Colouring, to be explored later.

3.4 Determining Uniqueness is Hard

It is hard to determine if a puzzle has a unique solution. *TO COMPLETE: FIND PAPER WITH PROOF*

$(a \vee b) \wedge (\neg a \vee \neg b)$

a	b		$(a \vee b)$	$(\neg a \vee \neg b)$	$(a \vee b) \wedge (\neg a \vee \neg b)$
F	F		F	T	F
F	T		T	T	T
T	F		T	T	T
T	T		T	F	F

Figure 2: Truth Assignment Example with Highlighted Valid Assignment

4 Solving Techniques

4.1 Backtracking

The standard way to solve a 9×9 sudoku puzzle is by the backtracking algorithm. This is a brute force method with a few optimisations. One can expect to find this algorithm in a computer science course introduction to recursion, that is to say it is not a complex concept and while useful for the usual sizes, as soon as we increase to 16×16 this becomes infeesible.

Listing 1: Backtracking

```
def Backtracking(grid):
    for each row:
        for each column:
            if grid is empty at this potion:
                try a value in this position
                Backtracking(grid with new value)
                if successful:
                    return grid
                else:
                    try another value
                if no values left to try:
                    return False
    return grid
```

Why does brute force not work for larger examples? It will work *TO DO: PROVE ALG CORRECTNESS* but due to the complexity of the problem (point back to sudoku is hard chapter) it is infeesible.

4.2 Stochastic Methods

4.2.1 Simulated Annealing

[?] one of 100 most cited papers, one of the first AI algs

5 Group theory

5.1 Starting Simple

Let us analyse Shidoku which is a specific set of sudokus with dimensions 4 by 4 the smallest non trivial sudoku puzzle. Only 2 fundamentally different. One has 96 identical, other has 192. Why not the same amount?

5.2 Equivalence Classes

6 Other Related Problems

6.1 Latin Squares

- A latin square is an n by n matrix filled with n characters that must not repeat along columns or rows.
- Reduced Form - f first row and column is in the natural order
- Equivalence classes
- Number of n by n latin squares is bounded
- Latin squares can be considered a bipartite graph
- Agronomic Research
- Latin hypercube

6.2 Magic Squares

- A magic square is a matrix of numbers with each column, row and diagonal summing to the same value, this value is known as a magic constant and the degree is the number of columns/rows.
- A normal magic square is one containing the integers 1 to n^2.
- Magic Squares with repeating digits are considered trivial.

- Semimagic squares omit the diagnonal sums also summing to the magic constant.
- Truly thought to be magic Shams Al-ma'arif.
- Generation, there exists not completely general techniques. Diamond Method
- Associative Magic Squares
- Pandiagonal Magic Squares
- Most-Perfect Magic Squares
- Equivalence classes for $n <= 5$ but not for higher orders.
- The enumeration of most perfect magic squares of any order.
- 880 distinct magic squares of order four
- Normal magic squares can be constructed for all values except 2
- Preserving the magic property when transformed
- Methods of construction
- Multiplicative magic squares - produce infinite
- Sator square
- magic square of squares - Parker Square is a failed example of this

6.3 Greco-Latin Squares

- Two orthogonal latin squares super imposed, such that the pairs of values are unique.
- Group based greco latin squares
- Eulers interest came from construction of magic squares
- Exists for all but 2 and 6.

7 Generating Techniques

A polynomial generation algorithm without requiring a uniqueness checker which we have proven to be np-complete and therefore infeesible for large n.

8 17 is the Magic Number

4 for shidoku

8.1 Sparsity - information theory

Bomb sudoku/latin squares - Additional rule: the same number can not occur in adjacent or diagonally adjacent squares.

9 Topology

9.1 Torus

10 Polynomials & Constraint Programming

Use of polynomials Roots of unity Grobner Basis

References

[1] https://en.wikipedia.org/wiki/NP_(complexity)

[2] Artificial Intelligence: A modern Approach Archived 2017-08-31 at the Wayback Machine [1995...] Russell and Norvig

[3] https://www.researchgate.net/publication/251863893_A_New_Algorithm_for_Generating_U Solution_Sudoku

[4] https://fse.studenttheses.ub.rug.nl/22745/1/bMATH_2020_HoexumES.pdf.pdf

[5] http://web.math.ucsb.edu/p̃adraic/mathcamp_2014/np_and_ls/mc2014_np_and_ls_lecture3. http://web.math.ucsb.edu/p̃adraic/mathcamp_2014/np_and_ls/mc2014_np_and_ls_lecture4.

[6] https://scholar.rose-hulman.edu/cgi/viewcontent.cgi?article=1398&context=rhumj

[7] https://onlinelibrary.wiley.com/doi/10.1002/(SICI)1520-6610(1996)4:6¡405::AID-JCD3¿3.0.CO;2-J

[8] http://joas.agrif.bg.ac.rs/archive/article/59

[9] https://www.semanticscholar.org/paper/Permutation-arrays-for-powerline-communication-and-Colbourn-Kløve/7e69cfdbd2082463c66de698da1e326f0556d1d4

[10] http://www.multimagie.com/English/SquaresOfSquaresSearch.htm

[11] https://plus.maths.org/content/anything-square-magic-squares-sudoku

[12] https://link.springer.com/book/10.1007/978-1-4302-0138-0

[13] an Laarhoven, P.J.M., Aarts, E.H.L. (1987). Simulated annealing. In: Simulated Annealing: Theory and Applications. Mathematics and Its Applications, vol 37. Springer, Dordrecht. https://doi.org/10.1007/978-94-015-7744-1_2