Proof P $\neq$ NP

Gernot Feichter

09.11.2016

Abstract

The P vs NP problem was regarded by the Clay Mathematics Institute to be one of the biggest unsolved mathematical problems of the millennium [Clay Mathematics Institute(2015)]. In this paper the proof that P $\neq$ NP is provided by showing that an optimal and exact algorithm to solve a NP-complete problem, such as the subset sum problem, has exponential worst-case complexity. The proof is structured in numbered sections, some of which are referring to other sections and in its entirety form the proof. Examples are provided in all sections to ensure the reader and author share the same interpretations. The proof defines deduction in the context of subset sums and outlines that utilizing this principle is the only possible way to reduce the complexity of a subset sum algorithm. The mathematical limits of deductibility are studied extensively and the result thereof is that an algorithm to solve the subset sum problem may not have lower than exponential complexity.

Author Information

This work was entirely done by the author without the support of any third party.

Author's address:
gernotfeichter@gmail.com

1 P vs NP problem

To prove that P $\neq$ NP it suffices to prove that the optimal and exact algorithm to solve some NP-complete problem has exponential complexity [Cook(1971)].

2 Subset sum problem

2.1 Property

The subset sum problem is NP-complete
[Weisstein and Wolfram Research(2015)].

2.2 Definition and Restriction

Given a list of numbers and a target sum, what subset of the numbers has the target sum? For the proof to be correct, only instances with a list of positive numbers may be used.

2.3 Notation

The notation of using an ascending sorted version of the list of numbers is mandatory for the proof to be valid and is used consistently throughout the paper.

2.4 Notation

To identify concrete subset sums, a binary bit is associated with each number of the sorted list of numbers (2.3), which form together a binary pattern. If there are no 0-bits between 1-bits, the pattern is referred to as coherent, otherwise it is referred to as non-coherent. For coherent binary patterns with n bits one denotes the amount of 0-bits ($= z$) on the left side of the 1-bits ($= x$) as z_left and the 0-bits on the right side of the 1-bits as z_right such that $z = z\_left + z\_right$ and $x = n - z$.

2.5 Example for 2.2 2.3 2.4

Problem: Given the list of numbers [5, 7, 14, 15, 18, 20] and the target sum 39, what subset of the given numbers has the given target sum?

Solution: The subset [7, 14, 18] - which may be identified by the binary

pattern [0,1,1,0,1,0] as well - sums to 39.

Remark: The binary pattern of the solution is non-coherent, since there is a 0-bit between 1-bits. If the binary pattern was [0,1,1,1,1,0], it would be coherent and the following statements would be true:
$n = 6$
$z\_left = 1$
$z\_right = 1$
$z = 2$
$x = 4$

2.6 Property

For subset sum problem instances with a list of n given numbers, there are 2^n possible subsets.

2.7 Proof for 2.6

Since all possible sums can be identified with all possible binary patterns and a binary pattern can also be represented as a binary number one may regard all possible binary patterns as the amount of numbers which may be represented with an n-bit binary number. Since the base of the binary number system is 2, there are 2^n possible numbers.

2.8 Exact algorithm

There exists an exact algorithm that solves the subset sum problem in $O(2^{\frac{N}{2}})$ [Horowitz and Sahni(1974)].

3 Summation Algorithm

3.1 Definition

A summation algorithm takes an amount of numbers as input and outputs the sum thereof.

3.2 Example

The definition of 3.1 includes any summation algorithm, for instance: summation as learned in school, looking up a table of pre-computed sums for given inputs or any other version that is able to perform the same operation.

3.3 Property

A summation algorithm has to perform at least some work to produce the output. That means an exact sum for given input numbers can not be known until some summation algorithm performs some work.

3.4 Proof of 3.3

From the fact that the input may be not equal to the output and is non-constant, it follows that a summation algorithm has at least non-zero complexity.

4 Lemmas for binary patterns

4.1 Lemma of binomial coefficients

For a binary pattern with n bits, where there are z 0-bits and x 1-bits, such that $x = n - z$, the binomial coefficient function may be used to calculate the amount of possible permutations, that is the amount of possible binary patterns:

$$binary\_patterns(n, x) = \frac{n!}{x! * (n - x)!}$$

4.2 Lemma of binomial coefficient sums

For a binary pattern with n bits, of which there are z 0-bits and **at least** x 1-bits, such that $x = n - z$, the **sum of** the binomial coefficient function may be used to calculate the amount of possible permutations, that is the amount of possible binary patterns:

$$binary\_patterns\_at\_least\_x\_ones(n, x) = \sum_{i=x}^{n} binary\_patterns(n, i)$$

From the latter function and 2.6 4.3 the conclusion may be drawn to define:

$$binary\_patterns\_at\_most\_x\_ones(n, x) =$$
$$binary\_patterns\_total(n) - binary\_patterns\_at\_least\_x\_ones(n, x + 1)$$

4.3 Lemma of all binomial coefficient sums

For the special case of the function $binary\_patterns\_at\_least\_x\_ones(n, x)$ (4.2.) where $x = 0$ the following identity applies due to 2.6:

$$binary\_patterns\_at\_least\_zero\_ones(n) = 2^n = binary\_patterns\_total(n)$$

4.4 Example of 4.1

Given a binary pattern with $n = 2$ bits and exactly $x = 1$ 1-bits, one may generate the following different binary patterns:
[0,1]
[1,0]

The amount of binary patterns may be verified and calculated by comparing the above result to the result of the function: $binary\_patterns(2, 1) = 2$

4.5 Example of 4.2

Given a binary pattern with $n = 2$ bits and **at least** $x = 1$ 1-bits, one may generate the following different binary patterns:
[0,1]
[1,0]
[1,1]

The amount of binary patterns may be verified and calculated by comparing the above result to the result of the function:

$$binary\_patterns\_at\_least\_x\_ones(2, 1) = 3$$

4.6 Example of 4.3

Given a binary pattern with $n = 2$ bits and **at least** $x = 0$ 1-bits, one may generate the following different binary patterns:
[0,0]
[0,1]
[1,0]
[1,1]

The amount of binary patterns may be verified and calculated by comparing the above result to the result of the function:

$$binary\_patterns\_at\_least\_x\_ones(2, 0) = 4$$

4.7 Corollary that 4.1 $\leq$ 4.2 $\leq$ 4.3

The functions in 4.1 4.2 4.3 depend on parameters n and x. Since the function of 4.2 sums up the function of 4.1 and 4.3 is regarded as the upper-bound of the function of 4.2 it can be concluded that the return values of the functions are in the following order: 4.1 $\leq$ 4.2 $\leq$ 4.3

5 Lemmas for optimal deducibility of subset sums

5.1 Uniform subset shift

5.1.1 Deduction of smaller shifted patterns

Given a binary pattern, one may deduce that by **shifting** one or more 1-bits for an arbitrary amount of positions **left**, the sum of that newly acquired pattern will be **smaller** than the one of the given binary pattern.

5.1.2 Deduction of larger shifted patterns

Given a binary pattern, one may deduce that by **shifting** one or more 1-bits for an arbitrary amount of positions **right**, the sum of that newly acquired pattern will be **larger** than the one of the given binary pattern.

5.1.3 Function to calculate amount of smaller shifted patterns (5.1.1)

The maximum amount of uniformly left shifted patterns that may be deduced from a coherent binary pattern may be calculated using the function

$$shifted\_patterns\_smaller(n, x) = binary\_patterns(n, x) - 1$$

where parameter $n = x + z\_left$, obtained from the given binary pattern, to calculate the amount of binary patterns yielding smaller sums than the given coherent binary pattern.

5.1.4 Function to calculate amount of larger shifted patterns (5.1.2)

The maximum amount of uniformly right shifted patterns that may be deduced (that is the total amount of possible uniformly right shifted patterns)

from a coherent binary pattern can be calculated using the function

$$shifted\_patterns\_larger(n, x) = binary\_patterns(n, x) - 1$$

where parameter $n = x + z\_right$ (obtained from the given binary pattern), to calculate the amount of binary patterns yielding larger sums than the given coherent binary pattern.

5.1.5 Examples of 5.1.1 5.1.2 5.1.3 5.1.4)

From the given binary pattern
[0,0,1,1,0,0]
one may deduce that the following uniformly left shifted patterns are smaller than the given binary pattern:
[1,0,1,0,0,0]
[0,1,1,0,0,0]
[0,1,0,1,0,0]
[1,0,0,1,0,0]
[1,1,0,0,0,0]

The amount of deduced shifted patterns above can be obtained by calculating the function $shifted\_patterns\_smaller(n, x)$ for $x = 2$ and $n = x + z\_left = 2 + 2 = 4$, yielding 5.

From the same given binary pattern one may deduce that the following uniformly right shifted patterns are larger than the given binary pattern:
[0,0,0,0,1,1]
[0,0,1,0,1,0]
[0,0,0,1,1,0]
[0,0,0,1,0,1]
[0,0,1,0,0,1]

The amount of deduced shifted patterns above can be obtained by calculating the function $shifted\_patterns\_larger(n, x)$ for $x = 2$ and $n = x + z\_right = 2 + 2 = 4$, yielding 5.

5.1.6 Proof that deduction using non-uniform subset shift creates a dependency on the list of associated numbers

By shifting one or more 1-bits left, while at the same time shifting one or more 1-bits right, one can not conclude the sum of the newly acquired pattern to be above, below or equal to the sum of the initial pattern, because it

depends on the associated list of numbers which of the three aforementioned cases will be met: From a list of ascending numbers $...a < b < c < d...$ and a binary pattern [...0,1,1,0...], that is non-uniformly transformed by shifting to [...1,0,0,1...], it does not follow that $b + c < a + d$, nor does it follow for the opposite or equal case, because one may always choose a large enough difference between a and b, such that the outcome of the comparison would completely depend on c and/or d. Appendix in section 5.4!

5.1.7 Proof of validity of 5.1.1 5.1.2

That the statements of 5.1.1 5.1.2 are valid statements, follows from the fact that the list of numbers, that is associated with a binary pattern, is sorted ascending (2.3).

5.1.8 Proof of optimality of 5.1.3 5.1.4

Since subset shift (5.1) can either be uniform or non-uniform, and it was proven in 5.1.6 that deduction using non-uniform subset shift is impossible, it is still necessary to prove that the maximum amount of deducible sums for uniformly shifted patterns may be described using the function $binary\_patterns(n, x)$ (4.1). Since said function describes all permutations with an amount of x 1-bits and all the shifted patterns may be regarded as such, the function is suitable. As to not include the original pattern, from which all other patterns are deduced, the given pattern is subtracted (-1) in functions of 5.1.3 5.1.4.

5.2 Uniform subset growth

5.2.1 Deduction of smaller grown patterns

Given a binary pattern, one may deduce that by **removing** one or more 1-bits, the sum of that newly acquired pattern will be **smaller** than the one of the given binary pattern.

5.2.2 Deduction of larger grown patterns

Given a binary pattern, one may deduce that by **adding** one or more 1-bits, the sum of that newly acquired pattern will be **larger** than the one of the given binary pattern.

5.2.3 Function to calculate amount of smaller grown patterns (5.2.1)

The maximum amount of uniformly shrunk patterns that may be deduced (that is the total amount of possible uniformly shrunk patterns) from a coherent (and in this case also non-coherent) binary pattern may be calculated using the function

$$grown\_patterns\_smaller(n) = binary\_patterns\_total(n) - 1$$

where parameter $n = x$ (obtained from the given binary pattern), to calculate the amount of binary patterns yielding smaller sums than the given binary pattern.

5.2.4 Function to calculate amount of larger grown patterns (5.2.2)

The maximum amount of uniformly grown patterns that may be deduced (that is the total amount of possible uniformly grown patterns) from a coherent (and in this case also non-coherent) binary pattern may be calculated using the function

$$grown\_patterns\_larger(n) = binary\_patterns\_total(n) - 1$$

where parameter $n = z$ (obtained from the given binary pattern), to calculate the amount of binary patterns yielding larger sums than the given binary pattern.

5.2.5 Examples of 5.2.1 5.2.2 5.2.3 5.2.4)

From the given binary pattern
[0,0,1,1,0,0]
one may deduce that the following uniformly shrunk patterns are smaller than the given binary patterns:
[0,0,0,0,0,0]
[0,0,0,1,0,0]
[0,0,1,0,0,0]

The amount of deduced shrunk patterns above can be obtained by calculating the function $grown\_patterns\_smaller(n)$ for $n = x = 2$, yielding 3.

From the same given binary pattern one may deduce that the following uniformly grown patterns are larger than the given binary patterns:

[0,0,1,1,0,1]
[0,0,1,1,1,0]
[0,0,1,1,1,1]
[0,1,1,1,0,0]
[0,1,1,1,0,1]
[0,1,1,1,1,0]
[0,1,1,1,1,1]
[1,0,1,1,0,0]
[1,0,1,1,0,1]
[1,0,1,1,1,0]
[1,0,1,1,1,1]
[1,1,1,1,0,0]
[1,1,1,1,0,1]
[1,1,1,1,1,0]
[1,1,1,1,1,1]

The amount of deduced grown patterns above can be obtained by calculating the function $grown\_patterns\_larger(n)$ for $n = 4$, yielding 15.

5.2.6 Proof that deduction using non-uniform subset growth can be described using subset shift or creates a dependency on the list of associated numbers

By removing a certain part of a pattern of 1-bits and by adding the exact same removed part at a different position, where there were only 0-bits before, one would also be able to describe that case as subset shift, which was already discussed in 5.1. However, by adding and removing different parts during one transformation, with different amounts of 1-bits, one may not conclude the sums thereof, because the outcome of the comparison depends on the concrete values of the associated list of numbers and the sums thereof. Appendix in section 5.4!

5.2.7 Proof of validity of 5.2.1 5.2.2

That the statements of 5.2.1 5.2.2 are valid statements, follows from the fact that the list of numbers, that is associated with a binary pattern, is sorted ascending (2.3).

5.2.8 Proof of optimality of 5.2.3 5.2.4

Since subset growth (5.2) can either be uniform or non-uniform, and it was proven in 5.2.6 that deduction using non-uniform subset shift is impossible, it is still necessary to prove that the maximum amount of deducible sums for grown patterns may be described using the function $binary\_patterns\_total(n)$ (4.3). Since said function describes all binary patterns with an arbitrary amount of x 1-bits and the grown patterns may be regarded as such, the function is suitable. As to not include the original pattern, from which all other patterns are deduced, the given pattern is subtracted (-1) in functions of 5.2.3 5.2.4.

5.3 Unification of uniform subset shift (5.1) and growth (5.2)

5.3.1 Deduction of smaller patterns

Given a binary pattern, one may deduce that by **shifting** one or more 1-bits for an arbitrary amount of positions **left and/or** by **removing** one or more 1-bits, the sum of that newly acquired pattern will be **smaller** than the one of the given binary pattern.

5.3.2 Deduction of larger patterns

Given a binary pattern, one may deduce that by **shifting** one or more 1-bits for an arbitrary amount of positions **right and/or** by **adding** one or more 1-bits, the sum of that newly acquired pattern will be **larger** than the one of the given binary pattern.

5.3.3 Function to calculate amount of smaller patterns (5.3.1)

The maximum amount of uniformly left shifted and/or shrunk patterns that may be deduced (that is the total amount of possible uniformly left shifted and/or shrunk patterns) from a coherent binary pattern may be calculated using the function

$$patterns\_smaller(n, x) = \\ shifted\_patterns\_smaller(n, x) \cup grown\_patterns\_smaller(n)$$

which unites the functions of 5.1.3 5.2.3 using the specified parameters such that one can also define it as:

$$patterns\_smaller(n, x) = binary\_patterns\_at\_most\_x\_ones(n, x) - 1$$

where parameter $n = x + z\_left$, to calculate the amount of binary patterns yielding smaller sums than the given binary pattern. Appendix in section 5.5!

5.3.4 Function to calculate amount of larger patterns (5.3.2)

The maximum amount of uniformly right shifted and/or grown patterns that may be deduced (that is the total amount of possible uniformly right shifted and/or grown patterns) from a coherent binary pattern may be calculated using the function

$$patterns\_larger(n, x) = \\ shifted\_patterns\_larger(n, x) \cup grown\_patterns\_larger(n)$$

which unites the functions of 5.1.4 5.2.4 using the specified parameters such that one can also define it as:

$$patterns\_larger(n, x) = \\ binary\_patterns\_total(n) * binary\_patterns\_at\_least\_x\_ones(n, x) - 1$$

where parameter $n = z\_left$, for the former function and parameter $n = x + z\_right$ for the latter function to calculate the amount of binary patterns yielding larger sums than the given binary pattern. Appendix in section 5.5!

5.3.5 Examples of 5.3.1 5.3.2 5.3.3 5.3.4)

From the given binary pattern
[0,0,1,1,0,0]
one may deduce that the following uniformly left shifted and/or shrunk patterns are smaller than the given binary pattern:
[1,0,1,0,0,0]
[0,1,1,0,0,0]
[0,1,0,1,0,0]
[1,0,0,1,0,0]
[1,1,0,0,0,0]
[0,0,0,0,0,0]
[0,0,0,1,0,0]
[0,0,1,0,0,0]
[0,1,0,0,0,0]
[1,0,0,0,0,0]

The amount of deduced patterns above can be obtained by calculating the function $patterns\_smaller(n, x)$ for $x = 2$ and $n = x + z\_left = 2 + 2 = 4$ yielding 10.

From the same given binary pattern one may deduce that the following uniformly right shifted and/or grown patterns are larger than the given binary patterns:
[0,0,0,0,1,1]
[0,1,0,0,1,1]
[1,0,0,0,1,1]
[1,1,0,0,1,1]
[0,0,0,1,0,1]
[0,1,0,1,0,1]
[1,0,0,1,0,1]
[1,1,0,1,0,1]
[0,0,0,1,1,0]
[0,1,0,1,1,0]
[1,0,0,1,1,0]
[1,1,0,1,1,0]
[0,0,0,1,1,1]
[0,1,0,1,1,1]
[1,0,0,1,1,1]
[1,1,0,1,1,1]
[0,0,1,0,0,1]
[0,1,1,0,0,1]
[1,0,1,0,0,1]
[1,1,1,0,0,1]
[0,0,1,0,1,0]
[0,1,1,0,1,0]
[1,0,1,0,1,0]
[1,1,1,0,1,0]
[0,0,1,0,1,1]
[0,1,1,0,1,1]
[1,0,1,0,1,1]
[1,1,1,0,1,1]
[0,1,1,1,0,0]
[1,0,1,1,0,0]
[1,1,1,1,0,0]
[0,0,1,1,0,1]
[0,1,1,1,0,1]

[1,0,1,1,0,1]
[1,1,1,1,0,1]
[0,0,1,1,1,0]
[0,1,1,1,1,0]
[1,0,1,1,1,0]
[1,1,1,1,1,0]
[0,0,1,1,1,1]
[0,1,1,1,1,1]
[1,0,1,1,1,1]
[1,1,1,1,1,1]

The amount of deduced patterns above can be obtained by calculating the function

$$patterns\_larger(n, x) = binary\_patterns\_total(n) * binary\_patterns\_at\_least\_x\_ones(n, x) - 1$$

for $x = 2$, $n = z\_left = 2$ ($binary\_patterns\_total$) and $n = x + z\_right = 2 + 2 = 4$ ($binary\_patterns\_at\_least\_x\_ones$), yielding 43.

5.3.6 Proof that deduction using a unification of non-uniform subset shift and non-uniform subset growth creates a dependency on the list of associated numbers

By combining the breakage of the principle of uniformity it is also not possible to deduce any additional binary pattern because both 5.1 and 5.2 depend already on the concrete associated list of numbers in case of non-uniform transformations (5.1.6 5.2.6). Appendix in section 5.4!

5.3.7 Proof of validity of 5.3.1 5.3.2

That the statements of 5.3.1 5.3.2 are valid statements follows from 5.1.7 5.2.7.

5.3.8 Proof of optimality of 5.3.3 5.3.4

Since the unification of uniform subset shift and growth (5.3) can either be uniform or non-uniform, and it was proven in 5.3.6 that deduction using non-uniform transformation is impossible, it is still necessary to prove that the maximum amount of deducible sums for arbitrary transformed coherent

patterns may be described using the functions
$binary\_patterns\_at\_most\_x\_ones(n, x)$ (4.2),
$binary\_patterns\_at\_least\_x\_ones(n, x)$ (4.2) and
$binary\_patterns\_total(n)$ (4.3).

Since $binary\_patterns\_total(n)$ (4.3) already appears in 5.2 it should be clear that it may also be used to calculate parts of 5.3, as 5.3 unites 5.1 5.2.

Since $binary\_patterns\_at\_most\_x\_ones(n, x)$ and
$binary\_patterns\_at\_least\_x\_ones(n, x)$ are both summations of
$binary\_patterns(n, x)$ (4.1 4.2.), it can be concluded that they include
$binary\_patterns(n, x)$ as well as all shrunk and/or grown variations in the corresponding context they are used in.

The multiplication symbol between the functions in 5.3.3 5.3.4 denotes to combine all partial patterns described by function A with all partial patterns of function B in order to describe all complete patterns.

By using the concepts of shifting and growth, it can be shown that one may transform any n-bit binary pattern into any other n-bit binary pattern, that means with both concepts combined, there are all possible transformations covered. Since both concepts (5.1 5.2) restricted to their corresponding transformation, and also the unification of both concepts (5.3) in an unrestricted manner, were proven to be optimal in deducibility, and the opposite of the concepts were proven indeducible at all (5.4), it would yield a contradiction, if by some other method than 5.3, one could deduce more sums without calculating more sums.

5.4 Appendix of non-uniformity for 5.1.6 5.2.6 5.3.6

The creation of dependencies on the list of associated numbers in 5.1.6 5.2.6 5.3.6 due to non-uniformity renders deducibility impossible, since deducibility is explicitly defined to deduce sums (6.6) and not to calculate sums, which would be necessary for further deductions. If one would perform such an additional calculation during a deduction (= non-uniform deduction) to extend the amount of deduced patterns, it can be concluded that there may not be more information derived than from performing individual uniform deduction for each of the calculated additional sums and combining the derived information, since that would contradict 5.1.6 5.2.6 5.3.6 5.1.8 5.2.8 5.3.8.

5.5 Appendix of 5.3.3 5.3.4

The unification of functions that return sets of binary patterns ($a(x)$ and $b(x)$), denoted as $a(x) \cup b(x)$ (used in 5.3.3 5.3.4), is meant to not only yield all results of a and b, but also those results that occur when b is applied to all results of a and vice versa, till the resulting set is complete.

5.6 Appendix of 5.1.8

As the functions of 5.1.3 5.1.4 are claimed to optimally describe the amount of deducible sums (5.1.8) by shifting, while they are only defined for coherent binary patterns, it needs to be shown that from a comparable non-coherent pattern, there can not be more patterns deduced. Shifting may only be uniform to perform a deduction and that means that the 0-bits within the 1-bits restrict the range of motion for one or more 1-bits. It can therefore be deduced that non-coherent patterns are inferior in the amount of patterns that may deduced from them.

5.7 Example of 5.6

From the given binary pattern
[0,0,1,1,0,0]
one may deduce that the following uniformly left shifted patterns are smaller than the given binary patterns:
[1,0,1,0,0,0]
[0,1,1,0,0,0]
[0,1,0,1,0,0]
[1,0,0,1,0,0]

[1,1,0,0,0,0]

However, from the comparable non-coherent pattern
[1,0,0,1,0,0]
one may only deduce the following uniformly left shifted patterns to be smaller:
[1,0,1,0,0,0]
[1,1,0,0,0,0]
as there are 0-bits between the 1-bits, which restrict the range of motion more than it is the case for the comparable coherent pattern.

6 Theorem of optimal deducibility of subset sums

6.1 Claim

An algorithm that solves the subset sum problem, must either know the exact sum, or be able to deduce the sum as being either too small or too large, for each of the 2^n possible subset sums (2.6).

6.2 Proof of 6.1

Since the solution of the subset sum problem must have per definition (2.2) the exact sum of the asked target sum, at least one target sum, if such exists, must be known and therefore calculated (3.3) exactly. That some sums may be concluded as being either too small or too large for the asked target sum, and that this is the only possible way to reduce the complexity of an algorithm for the subset sum problem, can also be shown by verifying the following claim: If there exists an algorithm that solves worst-case instances of the subset sum problem more optimal than 2^n, it must have concluded some subsets to be irrelevant for the solution, and that is the case if they are unequal to the target sum and therefore either too small or too large.

6.3 Example of 6.1 6.2

Knowing the exact sum for each of the 2^n possible subset sums (2.6) would have at least a complexity of 2^n (3.3). However, there exists an algorithm that is less complex, because it does not calculate each possible subset sum, but is able to deduce some of those sums to be either too small or too large (2.8).

6.4 Corollary of 6.1 6.2

It follows from 6.1 6.2 that an optimal algorithm to solve the subset sum problem must take optimal advantage of the principle of deducibility.

6.5 Optimal subset sum deducing algorithm

Given a binary pattern and the corresponding sum thereof, a subset sum deducing algorithm first compares the corresponding sum to the given target sum (2.2), which may then yield the following possible outcomes with their assigned conclusions:

$corresponding\_sum = target\_sum$ In case a subset sum algorithm would encounter this state during a deduction, it would have solved the problem. For the sake of this proof the optimal algorithm for worst-case scenarios is studied and this state would represent the best-case and is therefore of minor relevance to this proof.

$corresponding\_sum > target\_sum$ See section 6.7.

$corresponding\_sum < target\_sum$ See section 6.8.

6.6 Property of 6.5

It should be noted that a subset sum deducing algorithm as defined in 6.5 performs the deduction entirely on basis of the binary pattern. The list of concrete numbers, that is associated to the binary patterns, is unknown to the forecited algorithm.

6.7 Pattern for optimal deducibility of larger corresponding sums (Corollary of 6.5 5.3.4)

In case the $corresponding\_sum > target\_sum$, one may calculate the maximum amount of sums that is possibly deducible from the given binary pattern using the function $patterns\_larger(n, x)$ (5.3.4). The function $patterns\_smaller(n, x)$ (5.3.3) may not be used, as it only calculates the amount of binary patterns smaller than the given binary pattern, not the actual sums of each of those patterns. It can therefore not be known how those sums would compare to the target sum, without calculating them.
By examining the underlying functions of $patterns\_larger(n, x)$ (5.3.4) it

should become clear that the output of the given function is maximized if the given binary pattern has a maximum amount of 0-bits (z), starting from the left side of the pattern, ideally filling the entire pattern to the right side, such that $z\_left = z = n$.

6.8 Pattern for optimal deducibility of smaller corresponding sums (Corollary of 6.5 5.3.3)

In case the $corresponding\_sum < target\_sum$, one may calculate the maximum amount of sums that is possibly deducible from the given binary pattern using the function $patterns\_smaller(n, x)$ (5.3.3). The function $patterns\_larger(n, x)$ (5.3.4) may not be used, as it only calculates the amount of binary patterns larger than the given binary pattern, not the actual sums of each of those patterns. It can therefore not be known how those sums would compare to the target sum, without calculating them.
By examining the underlying functions of $patterns\_smaller(n, x)$ (5.3.3), it should become clear that the output of the given function is maximized if the given binary pattern has a maximum amount of 1-bits (x), starting from the right side of the pattern ($z\_right = 0$), ideally filling the entire pattern to the left side, such that $n = x$.

6.9 Pattern for optimal deducibility of smaller or larger corresponding sums (Corollary of 6.5 6.7 6.8)

It can be argued that it is unknown which of the conditions of 6.5 will be met prior execution. From mentioned uncertainty it follows, that an optimal subset sum algorithm could choose a binary pattern as input for the subset sum deducing algorithm (6.5), that would return an equal amount of deduced sums for both main cases (6.7 6.8). One may determine the optimal pattern by calculating the identity $patterns\_smaller(n, x) = patterns\_larger(n, x)$, such that one would find the closest match for a given n and variable x. From 6.7 6.8, it follows that such a pattern would be coherent, starting with 0-bits and ending with 1-bits ([0,...,1]) for a large enough n.

6.10 Example of 6.7 6.8 6.9

For a given $n = 3$, trying the pattern [1,1,1] with a corresponding sum of 1989 being smaller than a target sum of 2000, would be ideal, since even the maximum sum of the pattern is too small for the target sum and it could be deduced that also all other sums would be too small as well.

For a given $n = 3$, trying the pattern [0,0,0] with a corresponding sum of 0 being larger than a target sum of -10, would be ideal, since even the minimum sum of the pattern would be too large for the target sum and it could be deduced that also all other sums would be too large as well.

For a given $n = 3$, trying the pattern [0,0,1] with an unknown associated list of numbers, would be ideal, since the amount of binary patterns smaller:
[0,0,0]
[0,1,0]
[1,0,0]
matches the amount of binary patterns larger:
[0,1,1]
[1,0,1]
[1,1,1]
The optimal pattern for the unknown associated list of numbers is starting with 0-bits and ending with 1-bits, as described in 6.9

6.11 Simplification of identity of 6.9 for optimal pattern

The identity $patterns\_smaller(n, x) = patterns\_larger(n, x)$ (6.9) may be simplified, because the sub-function $binary\_patterns\_at\_least\_x\_ones(n, x)$ of the function

$$patterns\_larger(n, x) =$$
$$binary\_patterns\_total(n) * binary\_patterns\_at\_least\_x\_ones(n, x) - 1$$

(5.3.4) would always yield 1 in the case of an optimal pattern, such that one may use the simplified version with

$$patterns\_larger(n, x) = binary\_patterns\_total(n) - 1$$

where $n = z$.

6.12 Determination of parameters of optimal pattern

After having determined the optimal pattern by solving the identity of 6.9 6.11 for a given n, the values x and z may be determined as well from that pattern as described in 2.4. One may then calculate the optimal amount of deducible sums with those parameters.

6.13 Complexity order of functions for smaller and larger patterns

By investigating the parameters that are passed to the functions of the identity for an optimal pattern, it becomes apparent that the input of $patterns\_smaller(n, x)$ (5.3.3) is larger than the input of $patterns\_larger(n, x)$ (5.3.4). However, since the outputs on both sides of the identity should approach equality, it can be concluded that the complexity of the function $patterns\_larger(n, x)$ must be greater.

6.14 Complexity of functions for smaller and larger patterns

While it is trivial that the output of $patterns\_larger(n, x)$ has exponential dependency on its input parameters (6.11) due to the underlying term 2^z (4.3), it is also the case for $patterns\_smaller(n, x)$, as the underlying function uses a factorial (4.1), which can be shown to grow faster than an exponential with a constant base.

To summarize, when performing an optimal deduction using an optimal pattern (6.9) for a given n, n is split up into x and z, which are passed alongside n to one of two exponential functions, which need to provide a similar return value, so either case can be considered almost identical in terms of complexity on n. One may deduce only an amount of 2^z numbers to be either too small or too large for a given target sum. This is also the case when using the function $patterns\_smaller(n, x)$, as this function must provide approximately the output of $patterns\_larger(n, x)$ due to the identity.

6.15 Final Conclusion

Even if the optimal subset sum algorithm could use the optimal pattern (6.9) for every single optimal deduction, the algorithm would still have exponential complexity, because it would be necessary to perform that maximum deduction of an amount of 2^z subsets (6.14) out of 2^n possible subsets (2.6) for at least 2^{n-z} times (the number of all possible subsets was divided by the number of maximum amount of subsets to be deduced from a single deduction), which is equal to 2^x (6.9) and therefore it follows that $P \neq NP$.

References

[Clay Mathematics Institute(2015)] Clay Mathematics Institute. P vs NP Problem, 2015. URL http://www.claymath.org/millenium-problems/p-vs-np-problem.

[Cook(1971)] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*, pages 151–158, New York, New York, USA, May 1971. ACM Press. doi: 10.1145/800157.805047. URL http://dl.acm.org/citation.cfm?id=800157.805047.

[Horowitz and Sahni(1974)] Ellis Horowitz and Sartaj Sahni. Computing Partitions with Applications to the Knapsack Problem. *Journal of the ACM*, 21(2):277–292, April 1974. ISSN 00045411. doi: 10.1145/321812.321823. URL http://dl.acm.org/citation.cfm?id=321812.321823.

[Weisstein and Wolfram Research(2015)] Eric W. Weisstein and Inc. Wolfram Research. Subset Sum Problem – from Wolfram MathWorld, 2015. URL http://mathworld.wolfram.com/SubsetSumProblem.html.