Winker, Peter

Working Paper

Some notes on the computational complexity of optimal aggregation

Diskussionsbeiträge - Serie II, No. 184

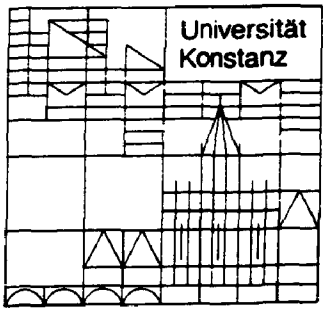
Provided in Cooperation with:
Department of Economics, University of Konstanz

Suggested Citation: Winker, Peter (1992) : Some notes on the computational complexity of optimal aggregation, Diskussionsbeiträge - Serie II, No. 184, Universität Konstanz, Sonderforschungsbereich 178 - Internationalisierung der Wirtschaft, Konstanz

This Version is available at:
https://hdl.handle.net/10419/101461

Sonderforschungsbereich 178
„Internationalisierung der Wirtschaft“

Diskussionsbeiträge



Juristische
Fakultät

Fakultät für Wirtschafts-
wissenschaften und Statistik

Peter Winker

Some Notes on the Computational Complexity of Optimal Aggregation

---

Postfach 5560
D-7750 Konstanz

Serie II — Nr. 184
Juli 1992

SOME NOTES ON THE COMPUTATIONAL COMPLEXITY OF OPTIMAL AGGREGATION*

Peter Winker

Serie II - Nr. 184

Juli 1992

* *I am grateful to W. Bob and T. Schneeweis for helpful comments on a preliminary version of this paper. Moreover, this paper benefitted from comments of the participants of a mathematical-economic workshop: W. Baur, W. Franz, W. Smolny and V. Strassen. Of course, the author is solely responsible for any errors remaining in the paper.*

Abstract

A combinatorical problem is said to be of high computational complexity, if it can be shown that every efficient algorithm needs a high amount of resources as measured in computing time or storage capacity. This paper will (1) introduce some basic concepts of mathematical complexity theory; (2) show that the problem of Optimal Aggregation is of high computational complexity; and (3) outline a possible way to obtain results good enough for practical use despite of this high computational complexity.

1 Introduction

Studying problems arising from daily practice one often reaches a level of formalisation which allows to express it as an mathematical optimization problem. For example the problem might be to maximize some function $f(x)$ under several constraints on the feasible solutions x. Then, one looks for a general method to solve this problem. In practice, when the set of feasible solutions is finite, there is a natural way to find the best solution: one calculates $f(x)$ for every feasible solution x and takes the x corresponding to the maximum value of $f(x)$. Unfortunately, even for very simple problems – for example to find a tour of shortest length connecting n cities – the set of feasible solutions may become extremly large. In the example for 100 cities there are about 10^{158} possible tours. Therefore, the second question arriving naturally is, how to compare different methods and how to find a "good" one.

In the last few decades a lot of progress has been made in the theory of computation. Many new and fast algorithms have been found, such as the Fast Fourier Transform or the Simplex algorithm. Another important example is the fast algorithm for matrix multiplication which is due to V. Strassen (1969): instead of the n^3 scalar multiplications necessary to multiply $n \times n$ matrices with the "classical" method it needs only $n^{\log_2 7}$. However, the most puzzling result of research in this area has been the discovery of certain natural problems for which all algorithms are inefficient. This means not only that there is no efficient algorithm known, but that it can be proved that there cannot exist anyone.

Another result, which will be discussed a little bit in the sequel, is the discovery of a class of problems – called NP–complete – which are supposed to be of high computational complexity. Despite of the efforts of many of the most famous researchers in this area no efficient algorithm has been found so far. It is just provable that whenever there is an efficient algorithm for one of these problems there has to be one for all. This means that whenever a given problem is in this class of problems it is "just as hard" as a large number of other problems which are commonly regarded as being difficult. The assumption that there is probably no efficient algorithm for these problems is known as *Cook's hypothesis.*

After a brief discussion of NP–completeness in section 2 the problem of Optimal Aggregation will be introduced. It comes up in a quite natural way in large scale econometric models. The goal in section 3 will be to show that this problem is NP–complete. Consequently, it is not possible to

calculate optimal solutions in practice. Finally, in section 4 an outline of a heuristic optimization algorithm will be given which already has been quite efficient in generating good approximations for many problems known to be NP–complete.

2 NP–completeness

Before discussing "inherently intractable" problems, some basic concepts of the theory of computation have to be introduced. This will be done in a concise and comprehensive way. For a more detailed and technical description the interested reader is referred to the standard literature.[1]

First, the meaning of the term "problem" has to be specified. It may be thought of as a general question depending on a set of parameters. For example, the general question could be to find a shortest tour connecting a given set of towns, where the set of parameters is given by the coordinates of these towns in some cartesian space. Any feasible set of parameters is called an instance of the problem. The size of the instance is defined as the length of some "natural" encoding[2] of the parameters. It can be shown that the computational complexity of a problem does not depend essentially on the chosen natural encoding.

In the mathematical theory of computation algorithms for problem solving are connected to theoretical computer models. The most famous among them is known as the "Turing machine" in honor of the important contributions to the theory of computation by A. Turing. Although the introduction of these models dates before the construction of the first electronic computers, they give a good description of the function of any real life computers. Hence, for the purposes of this paper, one can think of an algorithm as a computer program written in some higher computer language for some real computer. An algorithm solves a given problem if it gives a solution to the general question for any instance.

An important group of problems arises in combinatorial optimization: "Which element out of a finite set maximizes a given function $f(x)$[3]?" For these problems a natural algorithm consists in calculating the objective function for all elements of the finite set and in choosing the element correspond-

[1] See for example Aho, Hopcroft and Ullman (1974), Ferrante and Rackoff (1979), Garey and Johnson (1979), Hermes (1978), Hopcroft and Ullman (1979), Papadimitriou and Steiglitz (1982), Paul (1978), Reischuck (1990) and Wilf (1986).

[2] This means especially a concise encoding without unnecessary information or symbols.

[3] The function $f(x)$ in this context is often called "objective function".

ing to the maximum value. Unfortunately, for most problems in combinatorial optimization the finite set of possible solutions will be very large even for small problem instances. In this case, the natural algorithm as described above cannot be regarded as "efficient".

In general, one measures the efficiency of an algorithm in terms of time requirements depending on the instance size. This is a natural way, because the necessary computing time will depend on the size of the input. Consequently, after having chosen a specific encoding of the possible instances the time complexity function can be defined for any algorithm which solves the given problem. It gives for each possible instance size the largest amount of time needed by the algorithm to solve a problem instance of that size.

With the concept of time complexity functions the term "efficient" might be concretized. This is done by the distinction between algorithms working in polynomial time and algorithms with time complexity functions which cannot be bounded by polynomials. Table 1 indicates that polynomial time complexity functions grow rather modestly in their time requirements depending on the instance size, whereas the time needed by non polynomial algorithms grows explosively. The times in the table correspond to a computer facility with 1.000.000 arithmetic operations per second.

Table 1: Polynomial and non polynomial time complexity functions.

Time complexity function	Instance size n				
	10	20	30	40	100
n	0.00001 second	0.00002 second	0.00003 second	0.00004 second	0.0001 second
n^2	0.0001 second	0.0004 second	0.0009 second	0.0016 second	0.01 second
n^5	0.1 second	3.2 seconds	24.3 seconds	1.7 minutes	2.78 hours
$2^{n/\log n}$	0.00002 second	0.0001 second	0.00045 second	0.00184 second	3.4418 seconds
3^n	0.059 second	58 minutes	6.5 years	3855 centuries	$1.6 \cdot 10^{32}$ centuries

Regarding this table one might agree with computer scientists on looking

for polynomial time algorithms for a given problem. When there is possibly no polynomial time algorithm which can solve the problem it will be called intractable. Of course, there exist non–polynomial time algorithms which work quite well in practice, such as the simplex algorithm for linear programming.[4] But these examples of non–polynomial time algorithms which work well in practice are rare, not well understood so far and always risk to have enormous time requirements for some instances. As the presented measure of efficiency is a worst–case measure, i.e. the maximum amount of computing time needed for any instance of a given size is considered, only polynomial time algorithms will be regarded as efficient.

Begining with A. Turing in the 1930's, some problems have been proven to be not decidable at all, i.e. there exists no algorithm solving these problems. This means not only that no algorithm is known to solve these problems, but that there cannot exist one at all. It took some more years to find some "natural" decidable but provably intractable problems.[5] The proofs of most of these problems show that they cannot be solved in polynomial time even with a "nondeterministic" algorithm.

Such an algorithm can be viewed as being composed of two seperate steps. Given a problem instance, the first step just "guesses" a potential solution. This is done in parallel for an infinite number of potential solutions. Both the instance and the "guessed" solution are provided as inputs to the second step. The second step checks whether the provided potential solution is really a solution for the given instance. For example, for a given set of cities for the travelling salesman problem a first step guess is just one tour connecting all cities. The second step checks whether the length of this tour is less than a given value. If both the guessing and the checking step are executed in polynomial time the algorithm is called nondeterministic polynomial (NP). A problem belongs to the class of NP decision problems if it can be solved by a nondeterministic algorithm in polynomial time. Especially, any problem which may be solved by a deterministic polynomial time algorithm belongs to NP. A still unsolved problem is, whether there exist problems in NP, which cannot be solved by a polynomial time algorithm.[6]

As it showed to be very difficult to prove for a decidable problem that it is intractable or even nondeterministic intractable, the interest of researchers

[4] Another example is the quite complex algorithm for the Travelling Salesman Problem discussed in Holland (1987).

[5] See Mayr and Meyer (1982) and Winker (1991a) for the extended and formal discussion of one of them.

[6] Cook's hypothesis states that there are such problems.

concentrated on the interrelation of problems with respect to their time complexity. A reduction of one problem to another is given by a constructive transformation mapping any instance of the first into an equivalent instance of the second. If a reduction of one problem to another can be found, the first problem can be solved whenever there is an algorithm solving the second. If this reduction might be done by an algorithm working in polynomial time in the input size one speaks of "polynomial time reducibility". This term was introduced by S. Cook (1971) who proved in the same paper that the "satisfiability" problem[7] is in NP and that any other problem in NP can be polynomial reduced to it. This means that if the satisfiability problem can be solved with a polynomial time algorithm, so can every problem in NP – one just constructs the solving algorithm as a sequence of the reduction algorithm and the solving algorithm for the satisfiability problem. The other way round, if any problem in NP is intractable, the satisfiability problem must also be intractable.[8] Thus, in a certain sense, the satisfiability problem belongs to a group of "hardest" problems in NP.

A problem is called NP–complete, if every problem in NP may be reduced to it with an alogrithm working in polynomial time in the input size. As the polynomial time reducibility is obviously transitive, the following result is evident. If a NP–complete problem can be polynomially reduced to another problem in NP, the latter has to be NP–complete, too.

Meanwhile, literally hundreds of problems have been shown to belong to the class of NP–complete problems.[9] The question, however, of whether a polynomial time algorithm may be found for the NP–complete problems remains open. After many fruitless attempts it will need a major breakthrough in mathematics and computer science to decide it. For the moment, a problem in the class of NP–complete problems must be regarded as inherently difficult.

[7] This problem comes up in formal logic. The general question is, whether the variables may be assigned the values "true" and "false" such that the whole expression becomes "true".

[8] Otherwise, the sequential algorithm as described above would show that the first problem is not intractable.

[9] For an impressive list see the Appendix of Garey and Johnson (1979).

3 The computational complexity of Optimal Aggregation

In this section, it will be shown that the recognition version[10] of *OPTIMAL AGGREGATION* is NP–complete. At first, the problem of *OPTIMAL AGGREGATION* is introduced in its general version without paying much attention to the econometric backgrounds.[11] In two further subsections it will be proven that two simpler versions of *OPTIMAL AGGREGATION*, namely $GROUP_1$ and $GROUP_2$ are NP–complete. This will be done by using a polynomial reduction of the NP–complete problem *KNAPSACK*[12] to the recognition version of the considered problems $GROUP_1$ and $GROUP_2$. Finally, it will be shown that the general version of OPTIMAL AGGREGATION is at least as difficult as $GROUP_2$.

3.1 *OPTIMAL AGGREGATION*

The problem of *OPTIMAL AGGREGATION* comes up in large scale econometric models. Thus, let us regard a $n \times m$ matrix Y of observations of the dependent variables, a $n \times k$ matrix X of observations of the independent variables, a $k \times m$ matrix B of regression coefficients and the $n \times m$ matrix E of residuals. The regression model is then

$$Y = X\,B + E. \tag{1}$$

Sometimes the number of independent variables k exceeds by far the number of observations n or the model in its original version becomes to large for estimation. Then a way out of this problem might be to group the variables. The grouping or aggregation of the variables is done by regular grouping matrices G and H. In the simplest case each row of G and H contains exactly one 1 and 0 elsewhere. The reduced regression model becomes

$$Y\,H = X\,G\,B^* + E^*.$$

In Chipman (1975) the features of this aggregation are discussed in detail. The problem of *OPTIMAL AGGREGATION* is to choose G and H in an

[10] For a discussion of the different versions of a combinatorial optimization problem see Papadimitriou and Steiglitz (1982), p. 345.

[11] The interested reader is refered to Chipman (1975) and Chipman and Winker (1992).

[12] See Garey and Johnson (1979), p. 247, Papadimitriou and Steiglitz (1982), p. 374, or the original paper of Karp (1972), p. 94.

optimal way, i.e. in order to achieve a good approximation of the "true" model (1). Chipman (1975) shows that under some assumptions about the model this is done by minimizing the objective function

$$\alpha = \mathrm{tr}X(I - G\tilde{G}^{\#})\tilde{B}H(H'SH)^{-1}H'\tilde{B}'(I - G\tilde{G}^{\#})'X', \qquad (2)$$

where

$$\tilde{G}^{\#} = (G'X'XG)^{-1}G'X'X\,,$$
$$\tilde{B} = (X'X)^{-1}X'Y\,,$$
$$S = (Y - X\tilde{B})'(Y - X\tilde{B}).$$

In the sequel, it will be argued that the recognition version of this problem is already NP–complete. This means that the problem to decide whether there are grouping matrices G and H in such a way that the corresponding α lies in a given range is already very hard. Of course, the problem of finding the optimal groupings, i.e. the matrices minimizing the objective function α, is at least as difficult as the recognition version.

3.2 *GROUP*$_1$ is NP–complete

In this section, it will be shown that a simpler version of *OPTIMAL AGGREGATION* , namely *GROUP*$_1$ is NP–complete. This will be done by a polynomial reduction of *KNAPSACK* , which is well-known to be NP–complete. First, the *KNAPSACK* problem and the problem *GROUP*$_1$ considered in this subsection will be introduced. Comparing the two problems a natural nearly one to one correspondence between optimaly packed knapsacks and optimal groupings can be seen, which leads immediately to the reduction given in the proof of proposition 3.1.

Formally *KNAPSACK* is given by an integer $n > 0$, a vector of integers $\bar{x} \in (\mathbb{N}_{>0})^n$ and two positive integers k and k'. The question is whether it exists a $\bar{\delta} \in \mathbb{F}^n$ such that $k' \leq \sum_{i=1}^{n} \delta_i x_i \leq k$. Less formally, one might consider a rucksack of given volume and a set of n boxes of different volumes x_i. How can the volume of the knapsack be used in an optimal way? The question seems to be quite simple, but when experimenting with a number of 20 boxes, it is easy to see that there is no trival way to find the best possibility. Hence, it is not too astonishing that *KNAPSACK* has been proven to be NP–complete already in 1972.[13]

Unfortunately, there is no way to introduce the problem *GROUP*$_1$ in such a vivid way. It is given by the formal definition. For $\tilde{m}, m \in \mathbb{N}_{>0}$ with

[13] See Karp (1972), p. 94.

$\tilde{m} < m$ let the set of feasible grouping matrices $GR(m, \tilde{m})$ be given by

$$GR(m,\tilde{m}) := \{H \in M(\{0,1\}, m, \tilde{m}) \mid \begin{array}{l} \forall i \in [m]\ \exists!\, j \in [\tilde{m}]\ H_{ij} = 1, \\ \forall j \in [\tilde{m}]\ \exists\ i \in [m]\ H_{ij} = 1\} \end{array} ,$$

i.e. the matrices H in $GR(m, \tilde{m})$ are exactly those for the simplest case of *OPTIMAL AGGREGATION* discussed above. Furthermore, let the set of symmetric, positiv definite, regular matrices of rank m be given by:

$$P(m) := \{S \in GL(\mathbb{Q}, m) \mid S = S', S \text{ positive definite } \}$$

The elements S of $P(m)$ correspond to the S in (2). Then the grouping problem is to minimize the trace[14] of $(H'SH)^{-1}$ for given $S \in P(m)$.[15] Consequently, one has to regard the set of optimal groupings

$$\widetilde{GROUP}_1 := \{(H, S, m, \tilde{m}) \mid \tilde{m}, m \in \mathbb{N}, \tilde{m} < m, H \in GR(m,\tilde{m}), S \in P(m), \\ \operatorname{tr}(H'SH)^{-1} = \min_{\tilde{H} \in GR(m,\tilde{m})} \operatorname{tr}(\tilde{H}'S\tilde{H})^{-1}\}.$$

The optimization version of the optimal grouping problem $\widetilde{GROUP}_1$ corresponds to a recognition problem $GROUP_1$ with instance $\tilde{m} < m$, both in $\mathbb{N}_{>0}$, $S \in P(m)$ and $K \in \mathbb{Q}$ and the question, whether it exists a grouping matrix $H \in GR(m, \tilde{m})$ with $\operatorname{tr} H(H'SH)^{-1}H' \leq K$. It is obvious that the optimization version $\widetilde{GROUP}_1$ is at least as difficult as the recognition version $GROUP_1$.[16] In order to simplify the proof the following version of $GROUP_1$ will be considered: the instance is given by $\tilde{m} < m$, $S \in P(m)$ as before, and $K' < K \in \mathbb{Q}$. Now, the question is, whether there is a grouping matrix $H \in GR(m, \tilde{m})$ with $K' \leq \operatorname{tr}(H'SH)^{-1} \leq K$. $GROUP_1$ is in NP because for a given feasible grouping matrix $H \in GR(m, \tilde{m})$ the question whether $K' \leq \operatorname{tr}(H'SH)^{-1} \leq K$ is easily answered in polynomial time by straightforward matrix computations. Therefore a nondeterministic algorithm[17] would just calculate all the values of $\operatorname{tr}(H'SH)^{-1}$ and test whether at least one of them falls in the range defined by K' and K.

Proposition 3.1 *$GROUP_1$ is NP-complete.*

[14] I.e. the sum of the diagonal elements.

[15] Note that S assumed to be in $P(m)$ implies that $H'SH$ is positive definite and regular (Johnston (1972), p. 105), i.e. the inverse $(H'SH)^{-1}$ exists.

[16] See again Papadimitriou and Steiglitz (1982), p. 345.

[17] Remember, that a nondeterministic algorithm is something like a computer programm working on a computer facility with infinitely many parallel processors.

An instance of *KNAPSACK* is given by $n \in \mathbb{N}_{>0}$, $\bar{x} \in (\mathbb{N}_{>0})^n$ and two positive integers k and k'. The question is whether there exists a $\bar{\delta} \in \{0,1\}^n$ such that $k' \leq \sum_{i=1}^{n} \delta_i x_i \leq k$. Without loss of generality, it might be assumed that $k' \leq k$ and $N := \sum_{i=1}^{n} x_i \geq C$ for some given constant $C > k$.[18]

Now let us consider the instance for $GROUP_1$ given by $m := n > 2$, $\tilde{m} := 2$, $S_x := \mathrm{diag}(x_1, \ldots, x_n)$, a matrix containing nonzero elements only in the diagonal. Obviously S_x is in $P(m)$. To each $\bar{\delta} \in \{0,1\}^n$ corresponds a $H_{\bar{\delta}}$ in $GR(m, \tilde{m})$ defined by:

$$\begin{aligned} H_{i1} &:= \delta_i, & 1 \leq i \leq n \\ H_{i2} &:= 1 - \delta_i, & 1 \leq i \leq n \end{aligned}$$

Some straightforward calculations result in

$$\begin{aligned} \mathrm{tr}(H_{\bar{\delta}}' S_x H_{\bar{\delta}})^{-1} &= \mathrm{tr} \begin{pmatrix} \sum_i \delta_i x_i & 0 \\ 0 & \sum_i (1-\delta_i) x_i \end{pmatrix}^{-1} \\ &= \frac{N}{(\sum_i \delta_i x_i)(N - \sum_i \delta_i x_i)} \end{aligned}$$

It can be seen that the restricted version of $GROUP_1$ with a diagonal matrix S and only two columns for the grouping matrices H mimics *KNAPSACK*. A 1 in the first column of H corresponds to a box, which is put into the knapsack, whereas the 1s in the second column correspond to the boxes left outside. Of course, the roles of the two columns are interchangeable.

More formally, for $N \geq C := 2k$, $K' := \frac{N}{k(N-k)}$ and $K := \frac{N}{k'(N-k')}$: There is a solution for a given instance of the *KNAPSACK* problem if and only if there is one for the corresponding instance of $GROUP_1$ given by S_x, K' and K as defined above.

For the calculation of S_x out of x is polynomial, the reduction of *KNAPSACK* to $GROUP_1$ is polynomial. Therefore $GROUP_1$ is NP–complete which terminates the proof. □

3.3 $GROUP_2$ is NP–complete

Let the set of feasible grouping matrices GR and the set of symmetric, positiv definite, regular matrices P be given as defined above. Then, the grouping problem considered here is to minimize the trace of $H(H'SH)^{-1}H'$

[18] One just has to add a $x_{n+1} = C$. Obviously, δ_{n+1} has to be zero for all feasible solutions.

for a given $S \in P(m)$. As a result, the set of optimal groupings has to be regarded:

$$\widetilde{GROUP}_2 := \{(H,S,m,\tilde{m}) \mid \tilde{m},m \in \mathbb{N}, \tilde{m} < m, H \in GR(m,\tilde{m}), S \in P(m), \\ \mathrm{tr}H(H'SH)^{-1}H' = \min_{\tilde{H} \in GR(m,\tilde{m})} \mathrm{tr}\tilde{H}(\tilde{H}'S\tilde{H})^{-1}\tilde{H}'\}.$$

As in the last section, the corresponding recognition problem $GROUP_2$ will be considered with instance $\tilde{m} < m$, $S \in P(m)$ and $K' \leq K \in \mathbb{Q}$. The question will be whether there is a grouping matrix $H \in GR(m,\tilde{m})$ with $K' \leq \mathrm{tr}H(H'SH)^{-1}H' \leq K$. $GROUP_2$ is in NP for the same reasons as $GROUP_1$.

Proposition 3.2 *$GROUP_2$ is NP-complete.*

It will be shown that a restricted version of *KNAPSACK* is polynomial reducible to $GROUP_2$. Then, as this restricted version can be shown to remain NP–complete,[19] $GROUP_2$ has to be NP–complete, too.

An instance of the restricted version of *KNAPSACK* is given by an even integer n, a vector $\bar{x} \in (\mathbb{N}_{>0})^n$ and two positive integers k' and k. The question is whether there exists a $\bar{\delta} \in \{0,1\}^n$ such that $k' \leq \sum_{i=1}^n \delta_i x_i \leq k$ and $\sum_{i=1}^n \delta_i = n/2$. As for $GROUP_1$ one might assume that $k' \leq k$ and $N := \sum_{i=1}^n x_i \geq C$ for some given constant $C > k$.

Now consider the instance for $GROUP_2$ given by $m := n > 2$, $\tilde{m} := 2$ and $S_x := \mathrm{diag}(x_1,\ldots,x_n)$. To each $\bar{\delta} \in \{0,1\}^n$ corresponds a $H_{\bar{\delta}}$ in $GR(m,\tilde{m})$ as defined above. Some straightforward calculations result in:

$$\begin{aligned} \mathrm{tr}H_{\bar{\delta}}(H'_{\bar{\delta}}S_xH_{\bar{\delta}})^{-1}H'_{\bar{\delta}} &= \mathrm{tr}\,H_{\bar{\delta}} \begin{pmatrix} \sum_i \delta_i x_i & 0 \\ 0 & \sum_i (1-\delta_i)x_i \end{pmatrix}^{-1} H'_{\bar{\delta}} \\ &= \frac{\frac{n}{2}(N-\sum_i \delta_i x_i)+(n-\frac{n}{2})(\sum_i \delta_i x_i)}{(\sum_i \delta_i x_i)(N-\sum_i \delta_i x_i)} \\ &= \frac{\frac{n}{2}N}{(\sum_i \delta_i x_i)(N-\sum_i \delta_i x_i)}, \end{aligned}$$

and it follows as for $GROUP_1$ that $GROUP_2$ is NP–complete which terminates the proof. □

[19] See Garey and Johnson (1979), p. 65 and p. 223.

3.4 The computational complexity of *OPTIMAL AGGREGATION*

This section will be closed with an argument on the computational complexity of *OPTIMAL AGGREGATION* itself. Therefore, the real problem of optimal aggregation has to be regarded with the resulting objective function

$$\alpha = \mathrm{tr}X(I - G\tilde{G}^{\#})\tilde{B}H(H'SH)^{-1}H'\tilde{B}'(I - G\tilde{G}^{\#})'X'.$$

As the covariance–matrix S is symmetric and positiv semidefinite in general, S is in $P(m)$ in almost all cases. In order to see that *OPTIMAL AGGREGATION* is of high computational complexity it is enough to treat the case where S is diagonal. If the choice of G is independent of the choice of H

$$R := X(I - G\tilde{G}^{\#})\tilde{B}$$

is a constant matrix with respect to H. For the matrices R with equal sum K for the squared elements of each row an easy calculations shows that

$$\mathrm{tr}RH(H'SH)^{-1}H'R' = K \cdot \mathrm{tr}H(H'SH)^{-1}H'.$$

Thus, the general problem of *OPTIMAL AGGREGATION* is at least as difficult as $GROUP_2$ but still belongs to NP in its recognition version. Therefore, *OPTIMAL AGGREGATION* is NP–complete, also.

4 How to deal with NP–complete problems: Optimal Aggregation by Threshold Accepting

Today, no efficient algorithm solving NP–complete problems is known. After having proven that *OPTIMAL AGGREGATION* is NP–complete there is a strong temptation to abandon the search for optimal aggregations. However, it is really the only interesting result to get always the optimal solution? It might be sufficient to have an efficient algorithm for *OPTIMAL AGGREGATION* in the sense that in general it gives a good approximation to the optimal solution.

On this base many powerful multi-purpose optimization algorithms have been developed during the last few years. For example the Simulated Annealing algorithm[20] or the Threshold Accepting Algorithm[21]. These algorithms are heuristics, they do not always give optimal solutions. But many

[20] See Kirkpatrick, Gelatt and Vecchi (1983) and Aarts and Korst (1989).
[21] See Dueck and Scheuer (1990).

applications to problems of high computational complexity showed that they might achieve results good enough for practical purposes. The Threshold Accepting algorithm has been applied to the NP–complete Traveling Salesman problem, to Portfolio Optimization[22], to the problem of optimal chip-layout and many others.

The basic idea of Threshold Accepting is a local search, i.e. a current solution is compared with another feasible solution in a "neighborhood". The new solution is accpeted if it is not much worse than the old one.[23] For small examples where the optimal solution can be calculated – for example by raw computing power – Threshold Accepting tends to give the optimal value, too. But it only needs a few iterations compared to the direct optimization.

In Chipman and Winker (1992) an application of Threshold Accepting to a problem of optimal aggregation for price indices is presented. In this case, the set of feasible grouping matrices contains about $6 \cdot 10^{28}$ elements. Thus, the optimal solution cannot be calculated by raw computing power by any known algorithm. Nevertheless, the performance of Threshold Accepting allowed to find quite good solutions as for example compared to some "official" grouping.

The conclusion is that NP–complete problems are not completely hopeless to handle in practical applications. However, it remains a major task for mathematicians and computer scientists to find "efficient" algorithms for NP–complete problems or to prove that none can exist.

5 Conclusion

This paper presented a short introduction to the mathematical theory of complexity. It described a group of problems which are inherent hard to solve. The major task in this paper has been to prove that the problem of optimal aggregation as discussed in Chipman (1975) and Chipman and Winker (1992) belongs to this group of hard problems. Thus, the use of a heuristic optimization algorithm for optimal aggregation in Chipman and Winker (1992) is justified.

[22] See Dueck and Winker (1990).

[23] This means not worse then the old one plus a given threshold.

References

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, Chichester 1989

[2] A. Aho, J.E. Hopcroft and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974

[3] I. Althöfer and K.-U. Koschnick. *On the Convergence of Threshold Accepting.* Applied Mathematics and Optimization 24 (1991), 183-195

[4] J.S. Chipman. *Optimal Aggregation in Large-Scale Econometric Models.* The Indian Journal of Statistics Vol. 37 Series C (1975), 121-159

[5] J.S. Chipman and P. Winker. *Optimal Aggregation by Threshold Accepting.* Diskussionsbeiträge, Universität Konstanz, No. 180, Konstanz 1992

[6] S. Cook. *The Complexity of Theorem Proving Procedures.* Proceedings of the 3rd Annual ACM Symposium on Theory of Cmputing, Association for Computing Machinery, New York, 1971, 151-158

[7] G. Dueck and T. Scheuer. *Threshold Accepting: A General Purpose Algorithm Appearing Superior to Simulated Annealing.* Journal of Computational Physics 90 (1990), 161-175

[8] G. Dueck and P. Winker. *New Concepts and Algorithms for Portfolio Choice.* Applied Stochastic Models and Data Analysis (1992), forthcomming

[9] J. Ferrante and C. Rackoff. *The Computational Complexity of Logical Theories.* Lecture Notes in Mathematics No.718, Springer-Verlag, Berlin/Heidelberg/New York, 1979

[10] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, New York, 1979

[11] H. Hermes. *Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit.* Springer-Verlag, Berlin/Heidelberg/New York, 1978

[12] O. A. Holland. *Schnittebenenverfahren für Travelling-Salesman- und verwandte Probleme.* Dissertation, Rheinische Friedrich–Wilhelm–Universität, Bonn 1987

[13] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 1979

[14] J. Johnston. *Econometric Methods.* McGraw-Hill Kogakusha, Tokyo 1972 (2nd Edition)

[15] R.M. Karp. *Reducibility among Combinatorial Problems.* in: R.E. Miller and J.W. Thatcher. *Complexity of Computer Computations.* Plenum Press, New York, 1972, 85-103

[16] S. Kirkpatrick, C. Gelatt and M. Vecchi. *Optimization by Simulated Annealing.* Science Vol. 220 (1983), 671-680

[17] E.W. Mayr und A.R. Meyer. *The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals.* Advances in Mathematics 46 (1982), 305-329

[18] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982

[19] W.J. Paul. *Komplexitätstheorie.* B.G. Teubner, Stuttgart, 1978

[20] K. J. Reischuk. *Einführung in die Komplexitätstheorie.* B. G. Teubner, Stuttgart 1990

[21] V. Strassen. *Gaussian elimination is not optimal.* Numerische Mathematik 13 (1969), 354-356

[22] H. Wilf. *Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986

[23] P. Winker. *Die Speicherplatzkomplexität des Wortproblems für kommutative Halbgruppenpräsentationen und des Zugehörigkeitsproblems für rationale Polynomideale.* Diplomarbeit Fak. für Mathematik, Universität Konstanz, Konstanz 1991a