Lecture Notes in Computer Science 1982

Stefan Näher Dorothea Wagner (Eds.)

# Algorithm Engineering

4th International Workshop, WAE 2000
Saarbrücken, Germany, September 2000
Proceedings

Springer

Stefan Näher    Dorothea Wagner (Eds.)

# Algorithm Engineering

4th International Workshop, WAE 2000
Saarbrücken, Germany, September 5-8, 2000
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Stefan Näher
Universität Trier, Fachbereich IV – Informatik
54286 Trier, Germany
E-mail: naeher@informatik.uni-trier.de

Dorothea Wagner
Universität Konstanz, Fakultät für Mathematik und Informatik
78457 Konstanz, Germany
E-mail: Dorothea.Wagner@uni-konstanz.de

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Algorithm engineering : 4th international workshop ; proceedings / WAE 2000, Saarbrücken, Germany, September 5 - 8, 2000. Stefan Näher ; Dorothea Wagner (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 1982)
ISBN 3-540-42512-8

CR Subject Classification (1998): F.2, G.2, E.1, C.2, G.1

ISSN 0302-9743
ISBN 3-540-42512-8 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN 10781307 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 1982
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
*Berlin*
*Heidelberg*
*New York*
*Barcelona*
*Hong Kong*
*London*
*Milan*
*Paris*
*Tokyo*

# Preface

This volume contains the papers accepted for the 4th *Workshop on Algorithm Engineering* (WAE 2000) held in Saarbrücken, Germany, during 5–8 September 2000, together with the abstract of the invited lecture given by Karsten Weihe. The Workshop on Algorithm Engineering covers research on all aspects of the subject. The goal is to present recent research results and to identify and explore directions for future research. Previous meetings were held in Venice (1997), Saarbrücken (1998), and London (1999).

Papers were solicited describing original research in all aspects of algorithm engineering, including:

- Development of software repositories and platforms which allow the use of and experimentation with efficient discrete algorithms.
- Novel uses of discrete algorithms in other disciplines and the evaluation of algorithms for realistic environments.
- Methodological issues including standards in the context of empirical research on algorithms and data structures.
- Methodological issues regarding the process of converting user requirements into efficient algorithmic solutions and implementations.

The program committee accepted 16 from a total of 30 submissions. The program committee meeting was conducted electronically. The criteria for selection were originality, quality, and relevance to the subject area of the workshop. Considerable effort was devoted to the evaluation of the submissions and to providing the authors with feedback. Each submission was reviewed by at least four program committee members (assisted by subreferees). A special issue of the ACM *Journal of Experimental Algorithmics* will be devoted to selected papers from WAE 2000.

We would like to thank all those who submitted papers for consideration, as well as the program committee members and their referees for their contributions. We gratefully acknowledge the dedicated work of the organizing committee, and the help of many volunteers. We thank all of them for their time and effort.

July 2001

Stefan Näher
Dorothea Wagner

## Invited Lecturer

| | |
|---|---|
| *Karsten Weihe* | Institut für Diskrete Mathematik, Bonn |

## Program Committee

| | |
|---|---|
| *Michael Goodrich* | John Hopkins University |
| *Dan Halperin* | Tel Aviv University |
| *Mike Jünger* | Universität zu Köln |
| *Thomas Lengauer* | GMD Bonn |
| *Joe Marks* | MERL, Cambridge |
| *Stefan Näher*, Co-chair | Universität Trier |
| *Mark Overmars* | Universiteit Utrecht |
| *Steven Skiena* | State University of NY, Stony Brook |
| *Jack Snoeyink* | University of North Carolina, Chapel Hill |
| *Roberto Tamassia* | Brown University |
| *Dorothea Wagner*, Co-chair | Universität Konstanz |
| *Peter Widmayer* | ETH Zürich |

## Organizing Committee

| | |
|---|---|
| *Uwe Brahm* | MPI Saarbrücken |
| *Hop Sibeyn*, Chair | MPI Saarbrücken |
| *Christoph Storb* | MPI Saarbrücken |
| *Roxane Wetzel* | MPI Saarbrücken |

## Referees

Arne Andersson, Lars A. Arge, Stephen Aylward, Sanjoy Baruah, Mark de Berg, Hans L. Bodlaender, Matthias Buchheim, Adam Buchsbaum, Irit Dinur, Shlomo Dubnov, Matthias Elf, Martin Farach-Colton, Uriel Feige, Andrew Goldberg, Leslie Hall, Eran Halperin, Sariel Har-Peled, Han Hoogeveen, Kevin Jeffay, Lutz Kettner, Vladlen Koltun, Marc van Kreveld, Joachim Kupke, Olivier Lartillot, Frauke Liers, Bernard Moret, Michal Ozery, Oded Regev, A.F. van der Stappen, Cliff Stein, Sivan Toledo, Marinus Veldhorst, Remco Veltkamp, Bram Verweij, Karsten Weihe, Thomas Willhalm, Martin Wolff

# Table of Contents

# On the Differences between "Practical" and "Applied"

Karsten Weihe

Forschungsinstitut für Diskrete Mathematik
Lennéstr. 2, 53113 Bonn, Germany
weihek@acm.org

The terms "practical" and "applied" are often used synonymously in our community. For the purpose of this talk I will assign more precise, distinct meanings to both terms (which are not intended to be ultimate definitions). More specifically, I will reserve the word "applied" for work

> whose crucial, central goal is finding a feasible, reasonable (*e.g.* economical) solution to a concrete real–world problem, which is requested by someone outside theoretical computer science for his or her own work.

In contrast, "practical" then will refer to every other sort of implementation–oriented work. Most of the work published so far in WAE proceedings, AL(EN)EX proceedings, and "applied" tracks of other conferences in theoretical computer science is practical, not applied, in this spirit.

Many people got the fundamental experience that applied work is different and obeys its own rules. Since both practical and applied work is very versatile, it is hard to catch the exact differences. To get to the point, I will reduce the versatility of practical work to a "pattern" of sound practical work, which (to my feeling) reflects the current practice in our community quite well. The discussion of the differences between "practical" and "applied" is then broken down into discussions of the individual items in this pattern, which are more focused and hopefully more substantiated.

The methodology of the talk is to formulate one (or more) differences between "practical" and "applied" for each item and to substantiate them by "war stories" (in the sense of Skiena [13]). For brevity, most details (and all figures) are omitted. Selected aspects are discussed in greater detail in [15].

## Practical vs. Applied

In this talk and abstract, the difference between "practical" and "applied" is roughly identical to the difference between natural sciences and engineering sciences:

- In natural sciences, the "dirty" details of reality are abstracted away, and a simplified, coherent scenario is analysed under "lab conditions."
- Engineering sciences aim at a feasible, reasonable (*e.g.* economical) solution to the use case at hand.

For example, the restriction to numerical criteria such as the run time and the space consumption is already an abstraction; the effort for implementing an algorithm and maintaining the implementation is often much more important and must be taken into account in engineering work.

## "Pattern" for Sound Practical Work

So far, there is no standardized pattern of sound practical work, although there are various attempts to specify rules how to perform sound practical studies [2, 3, 5–10]. However, it seems to me that the following preliminary, immature pattern comes close to the de–facto standard established by the majority of practical papers specifically from our community:

– Choose an exact, clean, well–formalized problem.
– Implement algorithms from the theoretical literature and (optionally) own new variants/algorithms.
– Collect input instances from random generators and (optionally) public benchmark sets.
– Run the algorithms on these instances and collect statistical data on CPU time, operation counts, etc.
– Evaluate the statistics to compare the algorithms.

Here is my personal list of significant differences:

– Choose an exact, clean, well–formalized problem...
  ... but the problem may be highly underspecified and volatile.
– Implement algorithms from the (theoretical) literature...
  ... but the algorithms from the literature may be inappropriate.
– Collect input instances from random generators...
  ... but random instances may be very different from real–world instances and thus mislead the research.
– Make statistics on CPU time, operation counts, etc...
  ... but what to measure is not always clear.
  ... but non–quantifiable characteristics are often more important.
– Compare the algorithms...
  ... but the algorithms may solve different specifications of the underspecified problem.

## Detailed Differences

### Difference 1:

Choose an exact, clean, well–formalized problem...
... but the problem may be highly underspecified and volatile.

Very often, crucial details of the problem definition are not understood even if the problem is purely mathematical in nature. Even worse, the problem may involve psychological, sociological, political, etc., aspects, which might hardly be expressible in mathematical terms.[1]

If we can find a mathematical model whose theoretical outcome comes close to the empirical observations (and is efficiently computable), fine. However, personally, I was mainly confronted with applications from the real world in which such a model was beyond our analytical skills. An insightful example was presented in [16, 17]: the surface of a CAD workpiece is given as a *mesh*, that is, as a set of elementary surface patches (think of continuously deformated, curved polygons in the three–dimensional space). If two patches are intended to be neighbored on the approximated surface, they do not necessarily meet but are placed (more or less) close to each other. The problem is to reconstruct the neighborhood relations.

The methodological problem is this: the neighborhood relations are a purely subjective ingredient, namely the intention of the designer of the workpiece. In [16] we presented a computational study, which suggests that the data is too dirty to allow promising ad–hoc formalizations of the problem.[2] On the other hand, there is no evidence that a more sophisticated problem definition comes reasonably close to reality.

For a better understanding it might be insightful to compare this problem with another, more common class of problems (which stands for many others): *graph drawing*. The problem addressed in Difference 1 is certainly not new. Graph drawing is also an example for subjective ingredients, because the ultimate goal is not mathematical but aesthetical or cognitive. Nonetheless, various adequate mathematical models have been used successfully for real–world problems in this realm.

However, there is an important difference between these two real–world problems, and this difference is maybe insightful beyond this contrasting pair of specific examples. In the CAD problem from [16, 17], there is exactly one correct solution. All human beings will identify it by a brief visual inspection of a picture of the whole workpiece, because the overall shape of a typical workpiece is easily analysed by the human cognitive apparatus. In other words, everybody recognizes the correct neighborhood relations intuitively,[3] but nobody can specify the rules that led him/her to the solution. In particular, every deviation from this solution could be mercilessly identified by a punctual human inspection.

Thus, although the problem definition is fuzzy, the evaluation of a solution by the end–user is potentially rigorous. In contrast, the aesthetical evaluation of

[1] L. Zadeh, the inventor of *fuzzy logic*, found a pregnant formulation of Difference 1: precision is the enemy of relevance (citation translated back from German and thus possibly not verbatim).

[2] For instance, one ad–hoc approach, which is (to our knowledge) the standard approach outside academia, is to regard two patches as neighbored if their "distance" according to some distance measure is smaller than a fixed threshold value.

[3] Except for rare pathological cases, in which it was hard or even impossible to guess the designer's intention.

a graph drawing is as fuzzy as the problem definition itself. Roughly speaking, it suffices to "please" the end–user. To rephrase this difference a bit more provocatively: the strength of the meaning of "successful" may vary from application to application.

**Difference 2:**

Implement algorithms from the (theoretical) literature...
... but the algorithms from the literature may be inappropriate.

For example, a lot of work on various special cases of the general *scheduling* problem has been published throughout the last decades. Most theoretical work concentrates on polynomial special cases and makes extensive use of the restriction to a hand–picked collection of side constraints and objectives.

However, a typical real–world scheduling problem might involve a variety of side constraints and objectives. At least for me, there is no evidence (not to mention a concrete perspective) that any of these theoretical algorithms can be generalized to complex problem versions such that it truly competes with meta–heuristics like genetic algorithms and simulated annealing.

**Difference 3:**

Collect input instances from random generators...
... but random instances may be very different from real–world instances and thus mislead the research.

From the above example: what in the world is a random CAD workpiece?

Of course, we could try to identify certain "typical" statistical characteristics of CAD workpieces and then design a random generator whose outputs also have these characteristics. However, the profit from such an effort is not clear. If the "realistic" random instances reveal the same performance profile as the real–world instances, they do not give any additional information. On the other hand, if they provide a significantly different profile, does that tell us something about the expected performance of our algorithms in the application – or about the discrepancy between the real–world instances and our randomized approximation of reality?

Another result [14] may serve as an extreme, and thus extremely clarifying, example: for a given set of trains in some railroad network, the problem is to find a set of stations of minimal cardinality such that every train stops at one (or more) of them. We can regard each train as a set of stations, so the problem amounts to an application of the *hitting–set problem* (which is $\mathcal{NP}$–hard [4]).

In a preprocessing phase, two simple data–reduction techniques were applied time and again until no further application is possible:

1. If all trains stopping at station $S_1$ also stop at station $S_2$, then $S_1$ can be safely removed.

2. If a train $T_1$ only stops at stations where train $T_2$ also stops, then $T_1$ can be safely removed.

The computational study in [14] evaluated this technique on the timetables of several European countries (and their union) and in each case on various selected combinations of train classes. The message of [14] is the impressing result of this study: each of these real–world instances – without any exception! – was reduced to isolated stations and a few, very small non–trivial connected components. Clearly, all isolated stations plus an optimal selection from each non–trivial connected component is an optimal solution to the input instance. A simple brute–force approach is then sufficient.

Due to this extreme result, the general methodological problem is obvious: since this behavior occurred in all real–world instances with negligible variance, "realistic" random instances should also show this behavior. However, if so, they do not give any new insights.

**Difference 4:**

Make statistics on CPU time, operation counts, etc...
... but what to measure is not always clear.

It is good common practice in statistics to specify the goals of a statistical evaluation before the data is collected. In algorithmics, the favorite candidates are run time (in terms of raw CPU time, operation counts, cache misses, etc.) and space consumption. This is also the usual base for comparisons of algorithms.

However, sometimes the "right" measure to capture the quality of an algorithm is only known afterwards, and it may be specific for a particular algorithmic approach. The second concrete example in the discussion of Difference 3, covering trains by stations, may also serve as an illustration of Difference 4.

In this example, the space consumption and run time of the preprocessing phase and the brute–force kernel are negligible compared to the space requirement of the raw data and the time required for reading the raw data from the background device. What really counts is the number of non–trivial connected components and the distribution of their sizes. Of course, this insight was not anticipated, so the only relevant statistical measures could not be chosen a priori. It goes without saying that these measures are specific for the application and for the chosen algorithmic approach. It is not clear what a reasonable, fair measure for comparisons with other algorithmic approaches could look like.

**Difference 5:**

Make statistics on CPU time, operation counts, etc...
... but non–quantifiable characteristics are often more important.

Here are three examples of non–quantifiable characteristics:

- *Flexibility*:
  The typical algorithmic result in publications from our community is focused on one specific problem, very often a very restricted special case of a general problem (*e.g.* a classical problem such as disjoint–paths or Steiner–tree reduced to a restricted class of graphs). However, in an application project, one cannot assume that the formal model can be once designed in the first phase and is then stable for the rest of the development phase (not ot mention the maintenance phase). In Difference 1, two frequently occurring causes were stated:

  - The problem in which the "customer" is interested may change (volatile details).
  - Our understanding of the (underspecified) problem changes.

  Simple, artificial example for illustration: suppose we are faced with a problem on embedded planar graphs, and after a lot of hard theoretical work, we found an equivalent problem in the dual graph, which can be solved incredibly fast by a smart implementation of an ingenious algorithm. Then you feed the implementation with the data provided by the company, and it crashes...
  Error tracing reveals that there are edge crossings in the data. You call the responsible manager to complain about the dirtiness of the data. However, the manager will maybe tell you that the graph may indeed be slightly non–planar. Our assumption that the graph always be planar was the result of one of these unavoidable communication problems.
  Our imaginary algorithm relied on the dual graph, so it can probably not be generalized to the new situation. In other words, our algorithm was not flexible enough, and chances are high that we have to start our research from scratch again.
- *Error diagnostics*:
  Most algorithms in the literature simply return "sorry" in case the input instance is unsolvable. Some algorithms also allow the construction of a certificate for infeasibility. However, the answer "sorry" and a certificate for infeasibility are not very helpful for the end–user. The end–user needs to know what (s)he can do in order to overcome the problem. This information may be quite different from a certificate.
  The main example for Difference 1, reconstructing the neighborhood relations among the patches of a CAD model, may also serve as an example for this point. As mentioned above, there is no perspective for a realistic formal model. Nonetheless, algorithmics can make a significant contribution here, if the problem definition is changed slightly in view of error diagnostics.
  Since no satisfactory algorithmic solution is available, the result of an algorithm must anyway be corrected by the end–user. Hence, a more realistic objective would be to provide an approximation of the solution which only requires a minor revision effort from the end–user. Of course, a small number of errors is helpful for that, so one could be tempted to simply relax the

problem to its optimization version: errors are admitted but should be kept to a minimum.
However, the revision effort of the end–user is mainly determined by the problem to *find* the errors. This insight gives rise to another objective: construct an (erroneous) solution that can be presented visually such that the errors are easily found by human beings. It has turned out [16] that this problem is indeed treatable. Roughly speaking, it suffices to construct an overestimation of the correct solution. A simple coloring of a picture of the workpiece then allows the end–user to find all errors through a brief glance over the picture. Of course, the number of errors in the overestimation should still be as small as possible. However, this is not the main point anymore. It is more important to ensure that all details are on the "safe" (overestimating) side. To guarantee this,[4] a larger number of errors must potentially be accepted.
To summarize, only the deviation from the classical algorithmic viewpoint (which means that a problem should be solved fully automatically) allowed an algorithmic treatment of the problem.

– *Interactivity*:
An algorithm is not necessarily implemented as a stand–alone module, but is often intended as a core component of a larger software package. Nowadays, software packages are typically interactive, and this may have an impact on the feasibility of an algorithmic approach.
An example is production planning in a factory.[5] Here are three concrete examples of requirements imposed by interactivity.

  • *Additional input*:
  If a new customer order is accepted, the schedule must be computed again. It is often highly desirable that the schedule does not change significantly. For example, most random approaches might be excluded by this requirement.

  • *Additional restrictions imposed interactively*:
  For example, for reasons that are outside the underlying formal problem specification, the end–user (the supervising engineer–on–duty) may wish to fix the execution time of a job in advance. For example, a so–called *campaign* is a certain period in which a certain machine is only allowed to execute specific operations. Campaigns are possibly outside the formal model, simply because they are hard to handle algorithmically. To handle them manually, the end–user must be able to fix the execution times of selected operations to the campaign time.
  Note that there is a significant difference to the first item: a new customer order is nothing but additional input; time restrictions may not be part

[4] Of course, a rigorous mathematical guarantee is beyond our reach, so an empirical "guarantee" must suffice.

[5] This example was taken from on–going work. No quotable manuscript is available so far.

of the original problem definition and thus requires an extension of the model.

- *Fast termination required*:
  Sometimes the end–user needs a fairly good solution (which need not be totally feasible) very quickly. For example, the end–user must decide quickly whether an additional customer order can be accepted or not (imagine the customer calls the engineer–on–duty by phone). Then the engineer needs a raw estimation of the production plan. This requires something like an iterative algorithm that produces fairly good intermediate results right from the beginning. In other words, algorithmic approaches that do not have such a property are excluded.

**Difference 6:**

Compare the algorithms...

... but the algorithms may solve different specifications of the underspecified problem.

*Mesh refinement* [11, 12] is an example of Difference 6. In the CAD design process, this is the next step after the reconstruction of the neighborhood relations (which was the main example for Difference 1 and for item "failure handling and error diagnostics" of Difference 5). The problem is to decompose each patch into quadrilateral patches such that a mathematical analysis (finite–element analysis) is efficient and accurate. This problem is also an example for Difference 1 because it is only roughly understood how algorithmically treatable criteria such as the shapes of the quadrilaterals affect the performance of the finite–element method.

Not surprising, the individual algorithms proposed in the literature are based on various, completely different formal models (which are often not even stated explicitly), and they are tuned in view of the objectives of the corresponding model. A comparison of two or more algorithms must also be based on some formal model. However, there seems to be no "neutral" formal model, which is fair to all approaches.

This problem also occurs silently in comparisons with manual work. Many algorithmic problems from the industry were first solved by hand, and the first algorithmic results are compared to the manual solutions in use. However, many decisions in the manual design of solutions might be due to criteria that have never been made explicit, but were applied "intuitively." If the comparison of the algorithmic and the manual results is based on the side costraints obeyed by the algorithm and the objective function optimized by the algorithm, the evaluation is inevitably unfair to the manual solution.

## Conclusion

*Meta–heuristics* such as *simulated annealing*, *genetic algorithms*, and *neural networks* are very popular and in wide–spread use outside academia. From personal