Volume 21 Number 1

ACTA CYBERNETICA

Szeged, 2013

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Manuscripts must be written in good English. Contributions are accepted for review with the understanding that the same work has not been published elsewhere. Papers previously published in conference proceedings, digests, preprints are eligible for consideration provided that the author informs the Editor at the time of submission and that the papers have undergone substantial revision. If authors have used their own previously published material as a basis for a new submission, they are required to cite the previous work(s) and very clearly indicate how the new submission offers substantively novel or different contributions beyond those of the previously published work(s). Each submission is peer-reviewed by at least two referees. The length of the review process depends on many factors such as the availability of an Editor and the time it takes to locate qualified reviewers. Usually, a review process takes 6 months to be completed. There are no page charges. An electronic version of the puplished paper is provided for the authors in PDF format.

Manuscript Formatting Requirements. All submissions must include a title page with the following elements:

- title of the paper
- author name(s) and affiliation
- name, address and email of the corresponding author
- An abstract clearly stating the nature and significance of the paper. Abstracts must not include mathematical expressions or bibliographic references.

References should appear in a separate bibliography at the end of the paper, with items in alphabetical order referred to by numerals in square brackets. Please prepare your submission as one single PostScript or PDF file including all elements of the manuscript (title page, main text, illustrations, bibliography, etc.). Manuscripts must be submitted by email as a single attachment to either the most competent Editor, the Managing Editor, or the Editor-in-Chief. In addition, your email has to contain the information appearing on the title page as plain ASCII text. When your paper is accepted for publication, you will be asked to send the complete electronic version of your manuscript to the Managing Editor. For technical reasons we can only accept files in LaTeX format.

Subscription Information. Acta Cybernetica is published by the Institute of Informatics, University of Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. Subscription rates for one issue are as follows: 5000 Ft within Hungary, €40 outside Hungary. Special rates for distributors and bulk orders are available upon request from the publisher. Printed issues are delivered by surface mail in Europe, and by air mail to overseas countries. Claims for missing issues are accepted within six months from the publication date. Please address all requests to:

Acta Cybernetica, Institute of Informatics, University of Szeged
P.O. Box 652, H-6701 Szeged, Hungary
Tel: +36 62 546 396, Fax: +36 62 546 397, Email: acta@inf.u-szeged.hu

Web access. The above informations along with the contents of past issues are available at the Acta Cybernetica homepage http://www.inf.u-szeged.hu/actacybernetica/ .

Ferenc Gécseg
Department of Computer Algorithms
and Artificial Intelligence
University of Szeged
Szeged, Hungary
gecseg@inf.u-szeged.hu

Jozef Gruska
Institute of Informatics/Mathematics
Slovak Academy of Science
Bratislava, Slovakia
gruska@savba.sk

Tibor Gyimóthy
Department of Software Engineering
University of Szeged
Szeged, Hungary
gyimothy@inf.u-szeged.hu

Helmut Jürgensen
Department of Computer Science
Middlesex College
The University of Western Ontario
London, Canada
helmut@csd.uwo.ca

Zoltan Kato
Department of Image Processing
and Computer Graphics
Szeged, Hungary
kato@inf.u-szeged.hu

Alice Kelemenová
Institute of Computer Science
Silesian University at Opava
Opava, Czech Republic
Alica.Kelemenova@fpf.slu.cz

László Lovász
Department of Computer Science
Eötvös Loránd University
Budapest, Hungary
lovasz@cs.elte.hu

Gheorghe Păun
Institute of Mathematics of the
Romanian Academy
Bucharest, Romania
George.Paun@imar.ro

András Prékopa
Department of Operations Research
Eötvös Loránd University
Budapest, Hungary
prekopa@cs.elte.hu

Arto Salomaa
Department of Mathematics
University of Turku
Turku, Finland
asalomaa@utu.fi

László Varga
Department of Software Technology
and Methodology
Eötvös Loránd University
Budapest, Hungary
varga@ludens.elte.hu

Heiko Vogler
Department of Computer Science
Dresden University of Technology
Dresden, Germany
Heiko.Vogler@tu-dresden.de

Gerhard J. Woeginger
Department of Mathematics and
Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
gwoegi@win.tue.nl

Conference of PhD Students in Computer Science

Guest Editor:

Kálmán Palágyi
Department of Image Processing and Computer Graphics
University of Szeged
Szeged, Hungary
palagyi@inf.u-szeged.hu

The conference and this special issue are supported by the European Union and co-funded by the European Social Fund. Project title: "Broadening the knowledge base and supporting the long term professional sustainability of the Research University Centre of Excellence at the University of Szeged by ensuring the rising generation of excellent scientists". Project number: TÁMOP-4.2.2/B-10/1-2010-0012.

Preface

The eighth Conference for PhD Students in Computer Science (CSCS) was organized by the Department of Computer Science of the University of Szeged (SZTE) and held in Szeged, Hungary from June 28 to 30, 2012. The members of the Scientific Committee were the following representants of the Hungarian doctoral schools in computer science: András Benczúr (ELTE), Hasszan Charaf (BME), Tibor Csendes (SZTE), László Cser (BCE), János Csirik (Chair, SZTE), János Demetrovics (ELTE), József Dombi (SZTE), Zoltán Fülöp (SZTE), Aurél Galántai (ÓE), Tibor Gyimóthy (SZTE), Zoltán Horváth (ELTE), Csanád Imreh (SZTE), Zoltán Kató (SZTE), Zoltán Kása (Sapientia EMTE), László Keviczky (SZIE), János Kormos (DE), László Kozma (ELTE), János Levendovszky (BME), Eörs Máté (SZTE), Attila Pethő (DE), András Recski (BME), Lajos Rónyai (SZTAKI), Endre Selényi (BME), Tamás Szirányi (SZTAKI), Péter Szolgay (PPKE), and Tibor Tóth (ME). The members of the Organizing Committee were Balázs Bánhelyi, Rudolf Ferenc, Tamás Gergely, Zoltán Kincses, and Kálmán Palágyi.

There were more than 60 participants and 46 talks in several fields of computer science and its applications. The talks were going in sections in computer graphics, computer networks, database theory, discrete mathematics, distributed computing, image and signal processing, numerical analysis, optimization, software engineering, and stochastic processes. The talks of the students were completed by two plenary talks of leading scientists: András Kornai (Department of Computer Science, Boston University) and Horst R. Thieme (Department of Mathematics and Statistics, Arizona State University).

The scientific journal Acta Cybernetica offered students to publish the paper version of their presentations after a selection and review process. Eighteen manuscripts were submitted for publication. The present special issue of Acta Cybernetica contains 13 such papers.

The full program of the conference, the collection of the abstracts and further information can be found at `http://www.inf.u-szeged.hu/~cscs`.

On the basis of our repeated positive experiences, the conference will be organized in the future, too. According to the present plans, the next meeting will be held in the end of June 2014 in Szeged.

Kálmán Palágyi
Guest Editor

Acta Cybernetica 21 (2013) 5–20.

Low Level Conditional Move Optimization*

Artyom Antyipin‡ Attila Góbi‡ and Tamás Kozsik†

Abstract

The high level optimizations are becoming more and more sophisticated, the importance of low level optimizations should not be underestimated. Due to the changes in the inner architecture of modern processors, some optimization techniques may become more or less effective. Existing techniques need, from time to time, to be reconsidered, and new techniques, targeting these modern architectures, may emerge.

Due to the growing instruction pipeline of modern processors, recovering after branch mis-predictions is becoming more expensive, and so avoiding that is becoming more critical. In this paper we introduce a novel approach to branch elimination using conditional move operations, namely the `CMOVcc` instruction group. The inappropriate use of these instructions may result in sensible performance regression, but in many cases they outperform the sequence of a conditional jump and an unconditional move instruction.

Our goal is to analyze the usage of `CMOVcc` in different contexts on modern processors, and based on these results, propose a technique to automatically decide whether the conditional move or the sequence of a conditional jump and an unconditional move should be performed in a given situation.

Keywords: assembly, low level optimization, compilers

1 Introduction

Low level optimization has always been an important part of code generation. Sensible performance improvements can be achieved simply by reordering instructions or using an alternative, but equivalent, instruction sequence. Modern compilers support numerous optimization techniques applied to the generated code. Upcoming microprocessors are usually designed to run existing code faster without any adaptation. To achieve this, instruction processing is split into several stages, forming the so-called instruction pipeline. Each stage of the pipeline depends on the output of its predecessor, hence the processor starts to process the instruction several clock cycles prior to the actual execution. In order to keep the processor

*Supported by the European Union and co-financed by the European Social Fund (grant agreement no. TAMOP 4.2.1./B-09/1/KMR-2010-0003).

†Dept. Programming Languages and Compilers, Eötvös Loránd University, Budapest, Hungary, E-mail: {artyom,gobi,kto}@elte.hu

running, it is essential to keep the pipeline full. However, if the code being processed contains conditional branches, the processor has to choose one execution path. If there is a mis-prediction, the processor abandons the fetched instructions, which leads to several lost cycles, while the first instruction of the mis-predicted branch reaches the execution stage. During these cycles the executing engine is likely to be idle which, beside wasting time, also increases power leakage of the processor. The power gating technique has been proposed to address this issue but has not yet been adopted by any modern microprocessor [11].

Although modern processors use sophisticated branch prediction algorithms, prediction is practically impossible when the branch condition depends on random data. This makes *Worst-Case Execution Time* (WCET) estimation of the code containing such branches very hard, as the exact value of mis-prediction penalty does not depend solely on the pipeline length [10]. Therefore, a decrease in the number of conditional branches in the code may results in improvements in WCET estimations, and in making better use of the instruction pipeline. These ideas motivated us to look for possible approaches to branch elimination.

The paper is organized as follows. The next section gives an overview of the examined processor architectures and the instructions related to our approach. In Section 3, a first optimization attempt is detailed. The idea is to replace two possibly mis-predicted conditional jumps with a single, but unpredictable indirect jump. This method and its impact on the execution time is detailed there. Section 4 introduces the better approach of ours – total branch elimination in code generated for `if/else` constructions by manipulating operations performed within branches. Section 5 discusses related work. Finally, Section 6 concludes with pointing out future directions of work.

2 Preliminaries

The rest of the paper assumes that the reader has working knowledge on how processors work. Hence, in this section a short introduction is presented to the examined architectures (Section 2.1), the relevant (i.e. conditional) instructions (Section 2.2), with the conditional move detailed (Section 2.3). Section 2.4 demonstrates a trivial optimization, which can also be found in a recent version of the *GNU Compiler Collection* and the *clang* compiler.

2.1 Processor architecture overview

The microprocessor architecture overview provided in this section is rather simplified. Its intention is to provide enough information to understand motivation behind our attempts, while keeping information not related to this paper uncovered. Complete technical documentation is available publicly at the websites of the corresponding vendors [12, 3].

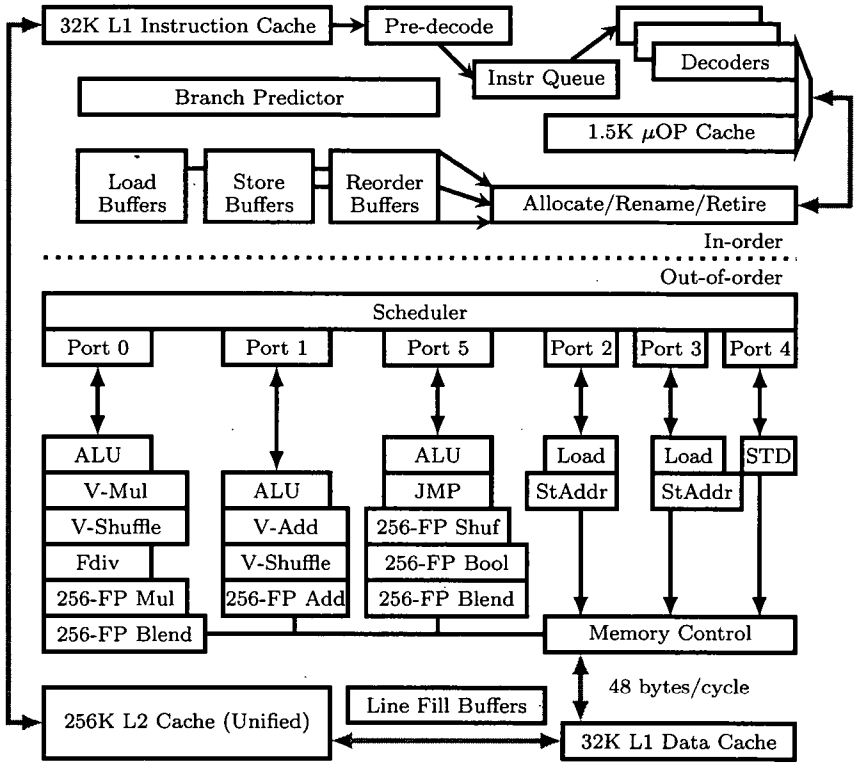


Figure 1: Intel microarchitecture with code name Sandy Bridge: Pipeline Functionality from [12]

2.1.1 Intel Sandy Bridge

Figure 1 depicts the pipeline and the major components of a processor core that is based on the Intel microarchitecture with code name Sandy Bridge. The pipeline consists of the following parts:

- In-order issue front-end, which includes
  - the branch prediction unit,
  - the instruction cache (L1i or ICache),
  - the instruction pre-decoder (4 units capable of micro and macro fusion),
  - the decoded ICache and
  - the micro-op queue, which decouples the front end and the out-of-order engine.
- Out-of-order execution engine which comprises of

- the renamer,
- the scheduler and
- the execution core.

Branch mis-predictions affect both the front-end (directly) and the execution engine (indirectly). According to the technical manual [12] "mis-predicted branches can disrupt streams of μops, or cause the execution engine to waste execution resources on executing streams of μops in the non-architected code path", i.e. the micro-op queue of the front-end is emptied, and either instructions from the mis-predicted execution path are decoded, or, if these instructions were already decoded and cached within the decoded ICache, the queue is re-filled using the cached micro-ops. In both cases the execution engine is suspended until the first micro-op is queued.

2.1.2 AMD K10 and K12

The structure of the AMD Family 10h and 12h (also called K10 and K12 respectively) based microprocessors is similar in many ways to that of Sandy Bridge described above. Instruction processing is split into several phases:

- The *Branch Prediction Unit* decides which instructions are to be fetched from the L1 instruction cache.
- Instructions are fetched and decoded into macro-ops by the *Fetch-Decode Unit.*
- The macro-ops then are passed to the ICU (i.e. *Instruction Control Unit*) which is responsible for
  - macro-op dispatch,
  - macro-op retirement,
  - register and flag dependency resolution and renaming,
  - execution resource management,
  - interrupts and exceptions and
  - branch mis-prediction handling.
- Macro-ops are dispatched either to *Integer Unit* or *Floating-Point Unit.* Both of them consist of a scheduler and an execution unit. The execution unit in both cases contains three execution pipes capable of executing instructions of the appropriate type.

No mechanism of branch mis-prediction handling is described by the documentations [3], but the mis-prediction penalty is said to be at least 10 cycles.

The functionality of the AMD Family 10h and 12h microprocessors seems to be less complex than that of the microprocessors based on the Intel architecture with

code name Sandy Bridge. As a consequence, the use of the code generation methods introduced in this paper produces less sensible, but still measurable, impact on the execution time on AMD Family 10h and 12h microprocessors.

2.2 Conditional instructions overview

Before introducing conditional instructions, the corresponding functionality of microprocessors based on the x86 architecture must be clarified. Among other registers, the x86 architecture includes the probably most frequently used special-purpose register – the so-called `FLAGS` register. (The name `FLAGS` refers to the 16-bit register of the basic x86 architecture. The 32-bit and 64-bit extensions of the architecture also affect this register. The 32-bit and 64-bit extensions of the `FLAGS` register are called `EFLAGS` and `RFLAGS`, respectively). `FLAGS` represents the state of the processor. Its bits are called flags, and each of them has a different purpose. Generally, these flags can be split into two separate groups – the ones representing the state of the processor after executing a particular instruction (called *status flags*), and the ones that can be modified in order to change the state of the microprocessor. Whether the operation described by a conditional instruction is performed, depends on the state of the *status flags*, as explained below.

In assembly language, conditional instructions are usually written in the form `OPCODEcc`, where `OPCODE` is a conditional instruction itself, and `cc` (called *condition code*) is one of the predefined conditions over the state of the status flags. If the actual state of the status flags satisfies this predefined condition, the operation described by the conditional instruction is performed, otherwise no action is taken. As a consequence, in order to take advantage of using a particular conditional instruction, the status flags should be adjusted prior the execution of the instruction. Modification of the status flags is possible in the following ways.

- Some of the flags (`CF`,`DF`,`IF`) can be adjusted explicitly with an appropriate instruction.

- The value of the lower byte of `FLAGS` can be transferred into `AH`, modified, and transferred back to `FLAGS`.

- The whole value of `FLAGS`, `EFLAGS` or `RFLAGS` (depending on the current processor mode) can be transferred into stack, adjusted, and then transferred back.

- Status flags are also adjusted implicitly, when a particular instruction is executed. Generally, most of the arithmetic, logic and bit shifting instructions implicitly adjust these flags. Furthermore, the x86 architecture provides two special instructions – `TEST` and `CMP` – which perform the same operation as `AND` and `SUB`, respectively, but their result is not stored, but only flags are adjusted. This latter facility is used to explicitly compare values.

2.3 Conditional move instruction

The `CMOVcc` instruction was introduced in the P6 processor family (Intel Pentium II) and usually described using the syntax below. It should be noted that in this paper the AT&T assembly syntax is used. See [9] for details about differences between the AT&T and the Intel syntax.

`CMOVcc` *source*, *destination*

Here, *source* can be either a general-purpose register or an in-memory variable; *destination* is a general-purpose register, and *cc* is the condition code (see Section 2.2). The operation performed by `CMOVcc` is detailed below.

temp ← *source*
IF *condition* TRUE
 THEN
  destination ← *temp*;
FI;

The operation can be split into three sub-operations – namely loading the value of the *source* operand, evaluation of the condition, and storing the loaded value into the *destination* operand. Note that the load sub-operation is performed unconditionally, i.e. even if the condition is not satisfied. As a consequence, if an in-memory variable is used as a source operand, it is loaded to cache – which is likely to be unnecessary if the condition is not satisfied and the variable is not used by other instructions. Furthermore, is this case the address of the variable must be valid (i.e. point to memory accessible by the program) or else processor exception will be raised, even if no move operation is to be performed. So `CMOVcc` with an in-memory variable would rather be used only when the variable is also used by other unconditional instructions. This restriction makes `CMOVcc` useless for optimization in several cases, as loading the variable into a register and using that register instead always results in better performance. Despite this, in our research we investigated ways to achieve better performance by using `CMOVcc` instructions with both registers and in-memory variables as the first operand.

2.4 A trivial case

Consider the C code fragment 1. It contains a single conditional branch that depends on a single condition, and has a single assignment operation within its body. Without any optimization, this code may be compiled to the assembly code shown in code fragment 2. Two variables are compared using the `CMP` instruction (2), which adjusts the status flags as if y was subtracted from x. If x was less than y, i.e. arithmetic borrow has been generated out of the most significant bit position, then the `CF` flag was set, otherwise the `CF` flag was reset. If `CF` was not set, the conditional should be skipped (3), i.e. the conditional jump to the end of the body of the branch (5) should be performed. If `CF` was set (i.e. x was less than y),

Code fragment 1 Trivial case (C/C++)

```
1  unsigned int x, y;
2  if (x < y)
3  {
4    x = y;
5  }
```

Code fragment 2 Trivial case (conditional jump + unconditional move)

```
1  # assume x = %rcx, y = %rdx
2    cmpq $rdx, %rcx
3    jnc 1f
4    movq %rdx, %rcx
5  1:
```

the jump operation is not performed, and line (4), namely the body of the branch, is executed.

The code, produced by a compiler without optimization, provides the expected functionality, but the conditional jump has a good chance of causing branch mis-prediction, and of wasting 14 cycles[1] each time the code is executed. Due to macro-fusion and out-of-order execution, the net execution time of the instructions in the code above is either 1, 2 or 3 cycles, depending on the position of the code in memory and the preceding instructions. However, together with the branch mis-prediction penalty, the execution of the code is expected to take 15-18 cycles.

Note that with random input the prediction is likely to fail. Fortunately, the code fragment 1 can be easily optimized using an *if-conversion* [17]. With a minimal effort, the code generator notices that a `CMOVcc` instruction can be used, as the expected functionality matches perfectly the definition of `CMOVcc`, as described in section 2.3. In this case the code generator can generate the code shown in code fragment 3: the comparison operation is kept unchanged, and the sequence of the conditional jump and the unconditional move is replaced with a single conditional instruction. This code has exactly the same functionality, and has no branches – i.e. no branch mis-prediction can ever happen. As a consequence, the execution of this code will take constantly 2 cycles. Using this single optimization in algorithms with constructions similar to the one shown in code fragment 1 can dramatically increase performance of the generated code. A good example is the *maximum* algorithm: improvements of using this optimization are shown in figure 2.

This case is trivial to optimize, because the used high-level construct perfectly

[1]There is no official information about the mis-prediction penalty, but different Internet sources [2, 4] agree on the same value of at least 14 cycles on microprocessors with Sandy Bridge architecture. On processors with AMD K10 and K12 architecture this penalty is defined to be at least 10 cycles [3].

Code fragment 3 Trivial case (conditional move)

```
1  # assume x = %rcx, y = %rdx
2    cmpq $rdx, %rcx
3    cmovb %rdx, %rcx
```

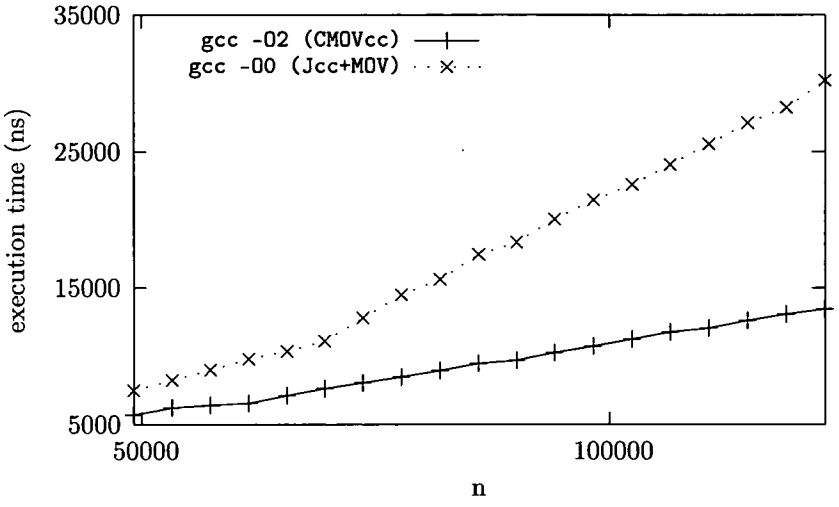


Figure 2: Maximum algorithm

fits the definition of `CMOVcc`. Popular compilers, like *gcc* [18] and *clang* [16], already support this optimization. It is worth mentioning that, probably because of problems discussed in Section 2.3, all optimizations involving the use of `CMOVcc` were disabled by default in older version of *gcc*. Newer versions (such as those above 4.5) of *gcc* have this optimization enabled – it is hard to tell exactly which versions, since no official announcement about this have ever been made.

In all tests included in this paper, the performance of our solutions was compared to the performance of the code generated by *gcc* with optimization enabled (`-O2`). Furthermore, we experienced no significant differences between code generated by *gcc* and *clang* for our test cases, and thus we assumed that the code generated by *clang* performs similarly to the one generated by *gcc*

3 Our first attempt

Consider the C++ function in code fragment 4. This function is given a pointer to some data, the length of the data and some threshold number x. It returns a tuple, containing the sum of the data items which are less than, greater than or equal to x, respectively. When the function is called, the body of the loop is executed

length times. When generating code for the body of the loop, popular compilers do recognize that a single compare operation is sufficient in this case, thus assembly code similar to code fragment 5 is generated. This code contains two conditional jump instructions, and if neither is taken, the third branch is executed. The main problem here is that the result of the comparison depends on potentially random data, and thus branch prediction is hardly possible in this case. As a consequence, this code contains two possibly mis-predicted jumps, which can be very costly, especially when executed within the loop.

Code fragment 4 Conditional sum (C++)

```
 1  std::tuple<int, int, int> sum(int *ptr, int length, int x)
 2  {
 3    int eq = 0, lt = 0, gt = 0;
 4    while (int i = 0; i < length; ++i)
 5    {
 6      if (ptr[i] < x)
 7        lt += ptr[i];
 8      else if (ptr[i] > x)
 9        gt += ptr[i];
10      else
11        eq += ptr[i];
12    }
13    return std::make_tuple(lt, eq, gt);
14  }
```

Code fragment 5 Three-way branch, generated code

```
 1    # assume %rdi = i,  %rsi = ptr, %ecx  = x
 2    #        %r8d = eq, %r9d = lt,  %r10d = gt
 3    movl (%rsi,%rdi,4), %edx
 4    cmpl %ecx, %edx
 5    jg do_gt
 6    je do_eq
 7  do_lt:
 8    addl %edx, %r9d
 9    jmp done
10  do_gt:
11    addl %edx, %r10d
12    jmp done
13  do_eq:
14    addl %edx, %r8d
15  done:
```

The main problem of the code generated for the body of the loop is the presence of two possibly mis-predicted conditional jumps. So our first intention was to decrease the number of the conditional jumps. Our main idea can be described as follows. Instead of performing a conditional jump, the pointer to the branch that should be taken is calculated using conditional move operations, and then an unconditional jump to this pointer is taken. This gives us the code shown in code fragment 6.

Code fragment 6 Three-way branch, single jump

```
 1    # assume %rdi = i,  %rsi = ptr, %ecx  = x
 2    #        %r8d = eq, %r9d = lt,  %r10d = gt
 3    movl (%rsi,%rdi,4), %edx
 4    leaq do_gt(%rip), %r11
 5    leaq do_eq(%rip), %r12
 6    leaq do_lt(%rip), %r13
 7    cmpl %ecx, %edx
 8    cmovg %r11, %r13
 9    cmove %r12, %r13
10    jmp *%r13
11  do_lt:
12    addl %edx, %r9d
13    jmp done
14  do_gt:
15    addl %edx, %r10d
16    jmp done
17  do_eq:
18    addl %edx, %r8d
19  done:
```

Unfortunately, the branch prediction unit of the examined processors cannot predict the single jump (in line 10), and hence this code constantly suffers from a single branch mis-prediction penalty. As a consequence, execution time of this code, opposed to the code shown in code fragment 5, depends neither on the input data nor on the inner state of the branch prediction unit.

After performing a series of tests, we came to the conclusion that the execution time of the code created using this method either equals to, or differs insignificantly from, the execution time of the code generated by *gcc*. As our attempt to minimize the number of branches did not lead to significant performance improvement, our goal changed to complete branch elimination, which led us to the approach we are to introduce.

4 Our approach

As we mentioned before, the main problem with the code generated by *gcc* for the function shown in code fragment 4 is the presence of two conditional jumps, and our goal is to completely eliminate branching, so that branch mis-prediction can never happen. Our idea is to rearrange the code in the following way. All the branches are executed unconditionally, and all the parameters used in the branches are assigned conditionally, i.e. depending on the condition, either set to the original value or to some neutral value determined by the operation (see code fragment 7). The new code provides the same functionality, and does not contain a single branch. It is worth to note that although this code performs poorly in the cases where branches can be predicted (e.g. if the function discussed in this section is used on sorted data), the execution time is halved in the general case.

Code fragment 7 Three-way branch, our approach

```
 1  # assume %rdi = i,  %rsi = ptr, %ecx  = x
 2  #        %r8d = eq, %r9d = lt,  %r10d = gt
 3  xorl %r11d, %r11d
 4  xorl %r12d, %r12d
 5  xorl %r13d, %r13d
 6  movl (%rsi,%rdi,4), %edx
 7  cmpl %ecx, %edx
 8  cmove %edx, %r11d
 9  cmovl %edx, %r12d
10  cmovg %edx, %r13d
11  addl %r11d, %r8d
12  addl %r12d, %r9d
13  addl %r13d, %r10d
```

In general, any `if/else if/else` construction that satisfies the restrictions listed below will perform better, if optimised according to our approach.

- All the conditions must use the result of a single assembly comparison operation, or at least two conditions must use the result of an assembly comparison operation.
- Overall cycles needed to execute all the operations of all the branches must be less than the overall possible branch mis-prediction penalty.
- All the operations of all the branches must have neutral values.

Note that our approach sets no restrictions on the exact number of parameters of the operations within the branches. Although the number of general-purpose registers is restricted, in-memory variables can also be used, as the penalty of the memory access is insignificant when compared to the penalty of a single mis-prediction.

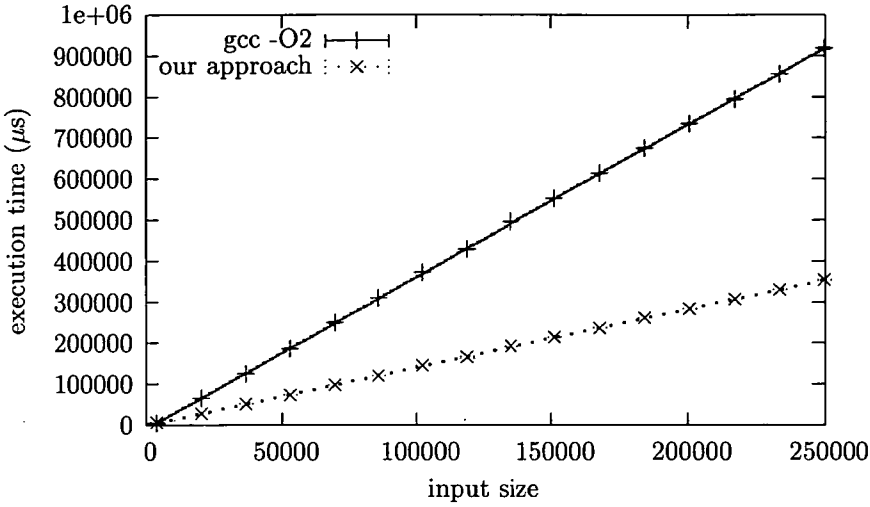


Figure 3: Execution time of the function specified in code fragment 4

Figure 3 shows results on the execution time of 1000 iterations of the code optimized by our approach and the one generated by *gcc*. The results are measured on an Intel Core i7-2620M processor, and a vector filled with pseudo-random data was used as an input. Table 1 contains further results on measuring the execution time of the function, including also the sorted input case, and the results taken on an AMD K10 processor as well.

For the measurements sorted and random input vectors of different sizes were used. Input sizes range through the rows of Table 1, while measurements on sorted and random input are depicted on the left and on the right, respectively. For every input size, experiments were carried out with *gcc* and with our hand-optimized code ("cmovcc"). The columns tagged "ratio" displays the execution time of our optimized code divided by the execution time of the one generated by *gcc*.

For each case the execution time was measured 12 times. In each experiment the first measurement was systematically larger than the others (probably because of cashing effects), therefore it was dropped. The remaining 11 measurements were averaged, and the variance was calculated. The variances were usually less than 1%, and never exceeded 5%, and thus they can hardly be observed on Figure 3. Assuming that the results are linear to the size of the input, we fitted a line on the measured data using the least squares method. In the case of input containing random data, the ratio in the slope of the lines are 2.619 ± 0.008. This allows us to conclude that our optimization yields 2.6 asymptotic speedup for the general case.

Our measurements are in accordance with the analysis presented in Section 2.4 on page 10. One can easily see that the code generated by gcc – i.e. the code using conditional jumps – performs very well when executed on sorted data (due to successful branch prediction) but shows dramatic performance decrease when

Table 1: Execution time of 1000 iterations (µs)

i7-2620M (based on microarchitecture code name Sandy Brigde)						
Input size	Sorted input			Random input		
	gcc	cmovcc	ratio	gcc	cmovcc	ratio
20480	12557	28969	230.70%	65543	28509	56.50%
69632	43238	99127	229.26%	250939	98447	60.77%
118784	72571	169463	233.51%	428812	164987	61.52%
167936	115746	239143	206.61%	614162	237441	61.34%
217088	160287	309680	193.20%	795009	307895	61.27%
249856	193510	356918	184.44%	918140	353916	61.45%
Phenom II X4 945 (AMD K10)						
Input size	Sorted input			Random input		
	gcc	cmovcc	ratio	gcc	cmovcc	ratio
20480	27420	32067	116.95%	79913	32073	59.87%
69632	93890	109755	116.90%	270162	109155	59.60%
118784	167020	193562	115.89%	461258	187265	59.40%
167936	226755	275454	121.48%	660903	273295	58.65%
217088	282321	357646	126.68%	853350	358102	58.04%
249856	323537	414665	128.17%	984064	414703	57.86%

random data is supplied. In contrast, the code created with our approach using CMOVcc shows no significant difference between the cases with sorted input and random input.

On random input the code optimized with CMOVcc was more than twice as fast as the one generated by *gcc*, both on the Intel and the AMD machines. On sorted input, *gcc* code performed twice as good as ours on Intel, and about 20% better on AMD.

5 Related work

To our best knowledge, all researches related to the usage of CMOVcc target only microprocessors with architecture different from x86, namely IA32, IA64 and Alpha [6, 19, 14, 17] and thus the technical documentations [3, 12, 13] provided by the vendors of the particular microprocessors remain the main source for the optimization techniques.

A study has been made in [5] on well-known optimization techniques including the one discussed in Section 2.4, but only the techniques already implemented and used by the particular compilers were considered. Furthermore, CMOVcc was used to optimize a HMMer search algorithm [15], and to mitigate timing-based side-

channel attacks by eliminating control flow dependencies [8]. In both papers the instruction was used within very specific cases, and no optimization technique has been proposed.

Another source of possible optimization techniques is the documentation of the popular compilers, but they are usually based on the mentioned technical documentations provided by the vendors of the microprocessors. Furthermore, in many cases implementation differs from the documentation as it contains modifications based on the feedbacks and proposals of the end-users.

6 Conclusions and Future work

In this paper we have studied branch elimination techniques based on replacing conditional jumps with conditional move operations. We have discussed the methods of branch number reduction and total branch elimination in code generated for the higher-level `if/else` constructions. The former one has proved to achieve only insignificant impact on the execution time of the produced code. The code created by using the latter method never suffers branch mis-prediction penalty, and hence outperforms the code generated by the popular compilers in the general case. Still, because of the increased complexity of the code, it performs poorly in some special cases, i.e. when no branch mis-prediction is caused by the compiler-generated code.

Although performance is not improved in the case of sorted input, the execution time of the code no longer depends on branch predictions. This has positive impact on WCET analysis, and makes its estimation more straightforward. This property can be extremely important in real-time systems [7].

In the future we will define the exact set of cases when our method could be used. Afterwards we plan to integrate our method into the popular compilers (e.g. *gcc* and *clang/llvm*) by providing appropriate plug-ins. This can serve as a convenient test-bed, and can speed up further research in this area.

References

[1] Зубков, С.В. *Ассемблер для DOS, Windows и UNIX.* Для программистов. ДМК Пресс, 2004.

[2] 7-Zip LZMA Benchmark, Intel Sandy Brigde. `http://www.7-cpu.com/cpu/SandyBridge.html`.

[3] Advanced Micro Devices, Inc. *Software Optimization Guide for AMD Family 10h and 12h Processors*, February 2011. Publication Number: 40546.

[4] Anandtech - the bulldozer aftermath: Delving even deeper. `http://www.anandtech.com/show/5057/the-bulldozer-aftermath-delving-even-deeper/2`.

[5] Bik, A.J.C., Kreitzer, D.L., and Tian, X. A case study on compiler optimizations for the Intel® Core TM 2 Duo Processor. *International Journal of Parallel Programming*, 36(6):571–591, 2008.

[6] Chuang, W. and Calder, B. Predicate prediction for efficient out-of-order execution. In *Proceedings of the 17th Annual International Conference on Supercomputing*, pages 183–192. ACM, 2003.

[7] Colin, A. and Puaut, I. Worst case execution time analysis for a processor with branch prediction. *Real-Time Systems*, 18(2):249–274, 2000.

[8] Coppens, B., Verbauwhede, I., De Bosschere, K., and De Sutter, B. Practical mitigations for timing-based side-channel attacks on modern x86 processors. In *30th IEEE Symposium on Security and Privacy*, pages 45–60. IEEE, 2009.

[9] Dean Elsner, Jay Fenlason & friends. Using the GNU Assembler for the family. `http://www.cs.utah.edu/dept/old/texinfo/as/as.html#SEC150`, March 1993.

[10] Eyerman, S., Smith, J.E., and Eeckhout, L. Characterizing the branch misprediction penalty. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 48–58. IEEE, 2006.

[11] Hu, Z., Buyuktosunoglu, A., Srinivasan, V., Zyuban, V., Jacobson, H., and Bose, P. Microarchitectural techniques for power gating of execution units. In *Proceedings of the 2004 international symposium on Low power electronics and design*, pages 32–37. ACM, 2004.

[12] Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, June 2011. Order Number: 248966-025.

[13] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual*, March 2012. Order Number: 325462-042US.

[14] Klauser, A., Austin, T., Grunwald, D., and Calder, B. Dynamic hammock predication for non-predicated instruction set architectures. In *International Conference on Parallel Architectures and Compilation Techniques, Proceedings*, pages 278–285. IEEE, 1998.

[15] Landman, J., Ray, J., and Walters, JP. Accelerating HMMer searches on Opteron processors with minimally invasive recoding. In *20th International Conference on Advanced Information Networking and Applications*, volume 2. IEEE, 2006.

[16] Lattner, C. LLVM and Clang: Next generation compiler technology. In *The BSD Conference*, 2008.

[17] Mahlke, S.A., Hank, R.E., McCormick, J.E., August, D.I., and Hwu, W.M.W. A comparison of full and partial predicated execution support for ILP processors. In *Proceedings of 22nd Annual International Symposium on Computer Architecture*, pages 138–149. IEEE, 1995.

[18] Mitchell, Mark and Samuel, Alexander. Gcc 3.0 state of the source. In *4th Annual Linux Showcase and Conference*, 2000.

[19] Wang, P.H., Wang, H., Kling, R.M., Ramakrishnan, K., and Shen, J.P. Register renaming and scheduling for dynamic execution of predicated code. In *The Seventh International Symposium on High-Performance Computer Architecture*, pages 15–25. IEEE, 2001.

Acta Cybernetica 21 (2013) 21–35.

Barcode Detection Using Local Analysis, Mathematical Morphology, and Clustering*

Péter Bodnár† and László G. Nyúl†

Abstract

Barcode detection is required in a wide range of real-life applications. Imaging conditions and techniques vary considerably and each application has its own requirements for detection speed and accuracy. In our earlier works we built barcode detectors using morphological operations and uniform partitioning with several approaches and showed their behaviour on a set of test images. In this work, those ideas have been extended with clustering, contrast measuring, distance transformation and probabilistic Hough transformation. Using more than one feature for localization leads to better accuracy, which makes detectors based on simple features, a competitive solution for commercial softwares and helps to fulfill the requirements of industrial applications even more.

Keywords: barcode detection, computer vision, clustering, feature extraction, morphological filters, distance transformation, Hough transformation

1 Introduction

Barcodes are 1D codes that consist of a well-defined group of parallel lines aiming easy automatic identification of carried data with endpoint devices such as PoS terminals, smartphones, or computers. Barcode decoding is fast and most barcode standards provide redundant information for error correction purposes. 2D codes are also referred to as barcodes, but in this paper, only codes in Fig. 1 are discussed.

Barcode localization methods have two main objectives, speed and accuracy. On smartphones, fast detection of barcodes is desirable, but accuracy is not so critical since the user can easily reposition the camera and repeat the scan. Accuracy is critical for industrial environment (e.g. postal services), where false negatives cause loss of profit. Speed is also a secondary desired property in those applications.

*This work was supported by the European Union and co-financed by the European Regional Development Fund within the project TÁMOP-4.2.1/B-09/1/KONV-2010-0005.

†Department of Image Processing and Computer Graphics, University of Szeged, E-mail: {bodnaar,nyul}@inf.u-szeged.hu

(a) Code 128 (b) EAN-13 (c) UPC (d) UPC-A

Figure 1: Barcode patterns

This is an important problem because the localization step is probably the most difficult part of general barcode detection and once localization is completed, decoding the barcode is relatively straightforward.

For 1D barcodes, the basic approach for localization is scanning only one, or just a couple of lines of the whole image. This is common for hand-held PoS laser scanners or smartphone applications. Scanned lines form an 1D intensity profile, and barcode-detector algorithms [1, 11, 16] work on those profiles to find an ideal binary function that represents the original encoded data. The main idea is to find peak locations in blurry barcode models, then thresholding the intensity profile adaptively to produce binary values.

Valley tracing (or bar tracing) [16] is a method for finding barcodes in blurry, low resolution images, mostly on live smartphone camera frames. It consists of three steps. At first, starting points have to be found on the picture, then "valleys" are followed, and finally, ends of the valleys (bars) are reached.

Algorithms with morphology [3, 4, 9, 13, 14] use the combination of basic morphological operation like erosion and dilation. White blobs on those images (Fig. 3) show the possible barcode locations. Further processing, like segmentation and filtering of small blobs are required on those difference images. It can be used on both 1D and 2D barcodes. Our work also involves morphology for efficient barcode localization.

Methods based on wavelet transformation [15] look at images for barcode-like appearance by a cascaded set of weak classifiers. Each classifier working in the wavelet domain narrows down the possible set of barcodes, decreasing the number of false positives while trying to keep the highest possible accuracy.

Variants of Hough transformation [2] detect barcodes by working on the edge map of the image. The two most common methods are standard and probabilistic Hough transformation. Both transform edge points into Hough space first, and make decisions of line locations. We are also experimenting with the idea of probabilistic Hough transformation extended with decisions about the features by projections of the Hough-space.

In the following section, simple features are presented to track for barcode localization. They are based on Hough-transformation, morphological operations, and uniform partitioning with distance transformation, contrast measuring and clustering.

2 Proposed Barcode Localization Approaches

In this section several different approaches are introduced for barcode localization. In most cases, image is examined in small, disjoint tiles (see Fig. 5 for optimal tile size), and local measurements are made. In this paper distance transformation [8] and contrast variance is used for these measurements. Code parts, like other textures, have well-traceable features. One of them is neighbor similarity, which means code parts in close proximity share similar local statistics with a well-chosen tile size. Thanks to the repeating patterns in the codes codes can be localized by observing how many code-like image parts (tiles) can be found. Finally, compactness of code parts is also important, which influences the final decision about code localization. Contrast information of the tiles and the number of clusters at pixel level are also examined.

With probabilistic Hough transformation, single lines of barcode can be detected for further processing. Clustering those lines helps to decide if a line is part of a barcode.

The way of clustering can also be applied at pixel level. A measure can be introduced with the help of the number and shape of pixel clusters.

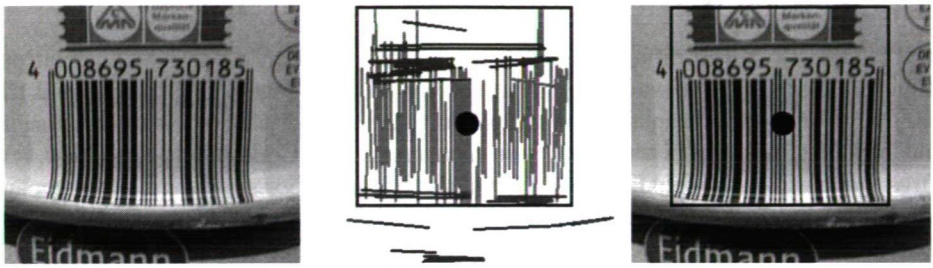
Lastly, with morphological operations, processing also leads to high accuracy. Features extracted with basic morphological operations, form a considerable base of more complex barcode localization algorithms.

2.1 Preprocessing

Digital images acquired from a camera often need preprocessing because of device flaws or environmental difficulties. In images having low contrast, intensity levels should be normalized. Unsharp masking is used in this paper, which is the weighted addition of the original image pixel intensities to the inverted pixel values of the gaussian-blurred version of the image. The blurring Gaussian filter is adjusted to not to destroy the narrowest line of the barcode, which parameter can be estimated by the specific endpoint application. Since some features need to be extracted from binary images, thresholding is necessary. A simple threshold is sufficient on images with uniform lighting, otherwise adaptive threshold [17] is required.

Image resolution does not have to be high, barcodes having the narrowest line of two pixels (px is used for pixels later in this work) is sufficient (3×3 px median filters can be applied to eliminate salt-and-pepper noise). Higher resolution yields better results, but also increases computation time. The least time-consuming solution for downsampling such images is the nearest neighbor interpolation, which is also a good choice because it preserves strong edges. However, at least 3 px minimum line width is desired for accurate code detection.

Color information could also be taken into account, however, most visual codes maximize the contrast by using only black (or dark blue) and white colors. Furthermore, industry hardware are often set to operate and record only in specific frequency ranges. According to these, only intensity values are processed and color information is dropped.



(a) original image (b) feature image (c) overlay

Figure 2: Canny edge detector with Probabilistic Hough transform. In (b), detected lines that are part of a barcode-like cluster are shown in red while the other detected lines are shown in blue

2.2 The Canny + Hough Method

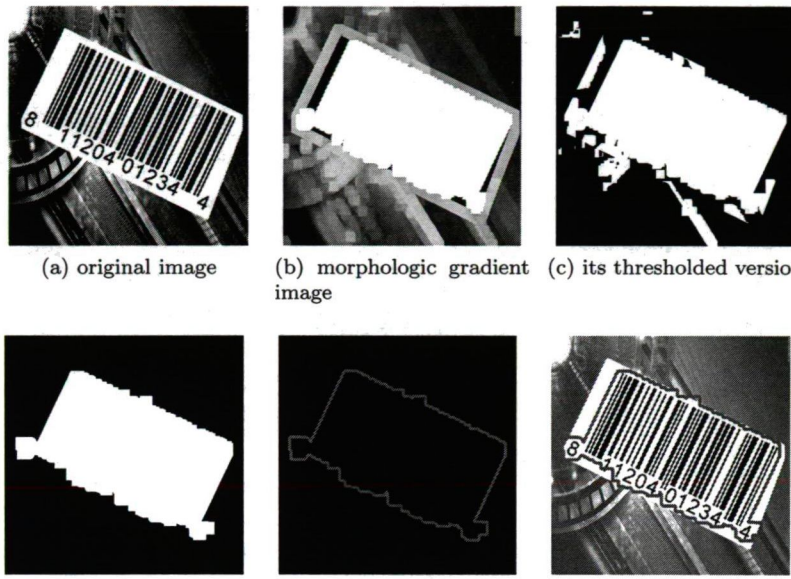
This method applies general image processing methods like Canny [6] edge detection and Probabilistic Hough transform [12], as barcodes consist of roughly equally long, parallel lines in a small area. With the help of the edge points, it gives a probabilistic estimation for line segments in the image, thus outperforming the standard Hough [18] transform, which only gives a set of lines as result. For preprocessing, a blur filter is recommended, since smooth images are desired for Canny edge detector. Since all barcodes in the test suite (see Subsection 3.1) have at least 50 px bar height, we set the minimum line length to 40 in the Hough transformation.

After a list of lines with their center point is obtained, length, and orientation, we can group them to decide whether they constitue a barcode or not. The minimum number of lines, the proximity needed for the lines to be in the same group, and the tolerance for length and orientation from the means inside the group vary for the final application. Since our barcodes consist of at least 25 parallel lines, we defined the minimum number of lines as 20. In the final step, group centers are returned, and the image can be cropped for decoding with known barcode decoding implementations (Fig. 2).

2.3 MIN–MAX

There are several works in the state of the art that use binary or grayscale morphology, see e. g. [10]. The two basic operations, erosion and dilation, and more complex operators such as bottom-hat are sufficiently robust and show good accuracy for barcode-like feature extraction.

The approach labeled as MIN–MAX treats the image as a whole, and therefore requires a fair amount of RAM and computation time. Supposed that intensity levels have already been normalized, no other preprocessing operations are required since this method manages well noisy, blurry or distorted images. Knowing the maximum bar width of a barcode, the morphologic gradient (`dilate()` − `erode()`)



(a) original image (b) morphologic gradient image (c) its thresholded version

(d) opening of image (e) contour image (f) contours overlaid

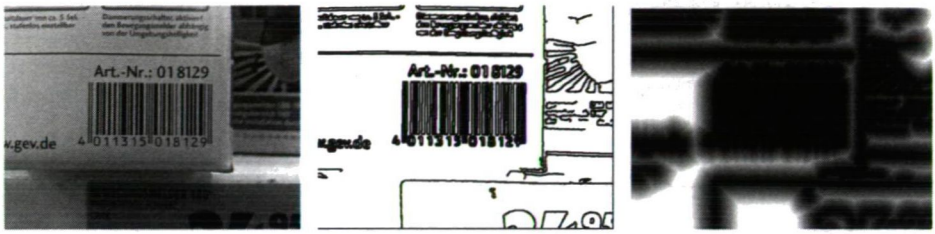
Figure 3: Stages of MIN–MAX method

operator is applied on the image with a box kernel of size $2 \times \lfloor \texttt{max\_barwidth}/2 \rfloor + 1$ (Fig. 3b). The next step is removing components that consist pixels with low intensity or being small. For component removal by intensity, the simplest way is thresholding (Fig. 3c). A good threshold can be at 75 % of the full intensity scale (e.g. 192 for 8-bit grayscale images), since barcodes produce areas close to the maximum intensity. After that, morphologic opening operation (`dilate(erode())`) is performed on the feature image (Fig. 3d), with the previously defined kernel. This ensures filtering out components that are too small or thin. Contours are extracted to the final step, which is to give a bounding box for image parts that might contain barcode (Fig. 3e).

Filtering of small areas can also be performed by connected component analysis. Each component size is measured and ones having smaller area than the defined minimum, are dropped. Experiments showed that setting the minimum component area to $0.75wh$ or smaller is satisfactory (where w and h are barcode width and height respectively).

2.4 Uniform Partitioning with Distance Transformation

Most barcodes, like regular textures, can be easily identified by observing only small parts of them. Those barcode parts together form the desired barcode region with known height and width. The first part of the method is partitioning the image to square tiles and look at each tile for barcode-like appearance. Each tile is



(a) original image (b) edge map (c) distance map

Figure 4: Canny edge map (b) and distance map (c) of a real-life example image (a). Barcodes have compact dark areas. Note: the values in the distance map are scaled for visualization.

assigned a value that indicates the grade of the presence of this feature. Globally, a matrix is formed from those values. Texture parts have similar local statistics in their neighborhood, so searching this matrix for compact areas defines image ROIs representing a barcode with high likelihood.

The assigned value showing barcode-like appearance is based on distance transformation of the edge map (Fig. 4). Canny edge detector is applied for providing the point set to the transformation. For each tile of the distance map, means and standard deviations are calculated. For 1D codes, distance values spread between half of the minimum and half of the maximum line width.

After evaluating all tiles locally, clusters are looked up in the feature matrix. Experiments show that the mean of distance values spread around half of the average line width. Thresholding is performed for the values to classify whether or not an area contains a barcode segment. Letting 25 % tolerance for these idealistic values, detection accuracy becomes satisfactory. For end-user applications we have to take noise, scratches and reflections into consideration. For images containing heavy noise, distance means drop. Barcodes suffering from scratches, dust, handwriting or reflections, change the distance means significantly according to the dark or bright intensity values of the flaw. Tolerance should be set according to amounts of these distracting properties and exact values can also be measured via trial and error. Tolerance value is a compromise between accuracy and the rate of false positive detections, so it should be set with respect to the final application.

The resulting binary matrix can be further analized via connected component labelling [7]. Finally, small components are dropped, and momentums of the remaining components are returned. A component is considered small if it contains less than N tiles (Eq. (1)), where h is barcode height, w is barcode width and s is the tile size, respectively.

$$N = \max\left(4, \frac{|h - s| \times |w - s|}{s^2}\right) \tag{1}$$

Since the smallest barcode in the test suite (see Subsection 3.1) has a 60 px

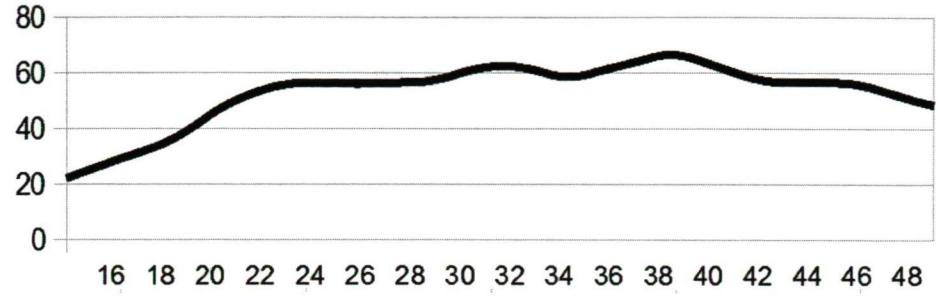


Figure 5: Detection accuracy with respect to the tile size. X-axis: proportion of tile size and barcode height; Y-axis: detection accuracy (both expressed in percent).

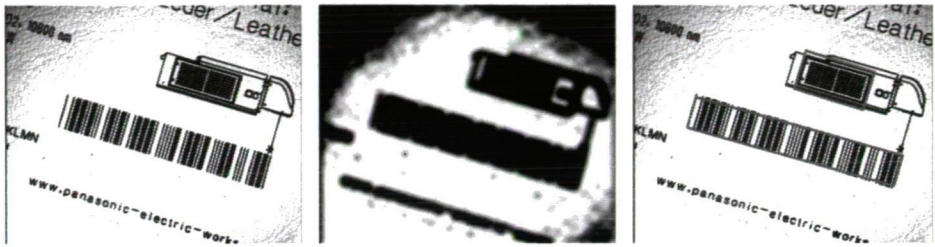
height, 30×30 px or greater tile sizes have poor recognition capability. However, very small tile sizes also lead to larger error for computing the center of the codes, because of the characters appearing below the code with code pieces nearby also have a barcode-like property (plain text is not affected). Also, choosing the tile size below two times the width of the widest barcode line leads to poor accuracy, since only two clusters can be detected on the tile, and that does not characterize a barcode part well. The best tiling size appears to be about 1/3 of the barcode height (Fig. 5). Since all examined codes consist of the same pattern (parallel lines), we looked in this paper for the optimal tile size for all types of codes together.

It is possible to run this method with disjunct or overlapping tiles, but overlapping does not improve the method's accuracy, only increases computation time. Offsetting the tiles has no significant effect, since it just produces some blocks to be more and others to be less "barcode-like" at the barcode's opposite sides.

This method gives a moderate rate of false positives (Fig. 6). Text with a comparable font size to barcode line thickness reacts similar to distance transformation, and distance values have similar mean and standard deviance in both cases. However, distance transformation can be used as a weak "classifier" of image areas, and its output is a good starting point for more accurate methods. The protocol to test optimal parameters is highly experimental, however, in end-user applications, expected element size or bar width of the barcode can be expected within a certain range. In case if the approximate distance between the camera and its observed plane of interest, object type (letter, box having dimensions in a limited range), and code dimensions are known, code size can be estimated in the acquired image. The intensity and characteristics of illumination is also limited in most cases, especially in industrial environment.

2.5 Contrast Measuring with Uniform Partitioning

Using the approach of image tiling, contrast information is examined for each tile. One-dimensional barcodes usually have high variance in contrast at one specific direction, while having low variance perpencidularly to that. The discussed rules



(a) original image (b) distance map (c) detected rectangular ROIs

Figure 6: Real-life example of a bar-coded product case with non-uniform illumination. Original image (a), distance transformation (b), and the detected rectangular ROIs based on the distance map (c)

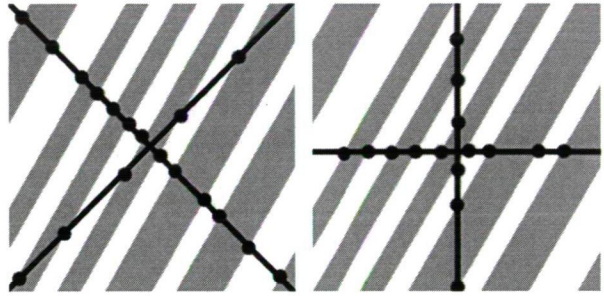
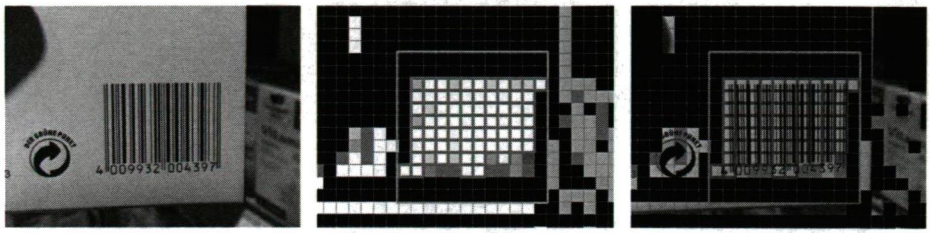


Figure 7: Two pairs of scanlines sweep through the image. One gives significant difference in contrast variance between perpendicular directions. The first pair of scanlines shows well barcode-like property of image tile, while the other one does not.

of tiling and forming the final decision also apply to the approach of contrast measurement, only the value of each tile is assigned differently.

Each tile is checked locally for barcode-like appearance with a modified version of the scan-line analisys. Two pairs of perpendicular scanlines are used, one pair at 0° and 90° and the other at 45° and 135° (Fig. 7). A measure is derived from contrast variance along the scanlines (Eq. (2)), i.e., a horizontally aligned barcode has a lot of contrast changes when scanned with a horizontal scanline, but has few or none in case of a vertical scan (Fig. 7). With the 2 pairs of scanlines, barcode pieces of any orientation can be safely recognized. The final measure assigned to a tile is the maximum of the two values. This measure gives 1 if parallel lines are present on an image tile, and 0 if a tile contains homogeneous area or noise.

$$C_i = \frac{|V_{i1} - V_{i2}|}{\max(V_{i1}, V_{i2})} \tag{2}$$



(a) Original image (b) feature image with bounding box (c) overlay

Figure 8: Contrast measuring on a real-life example.

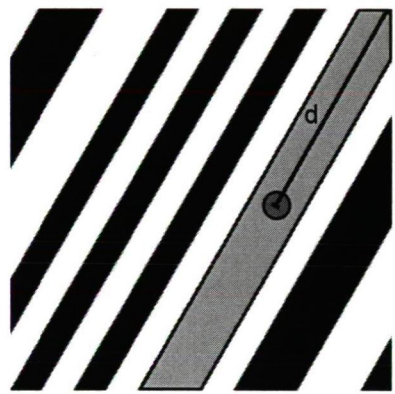


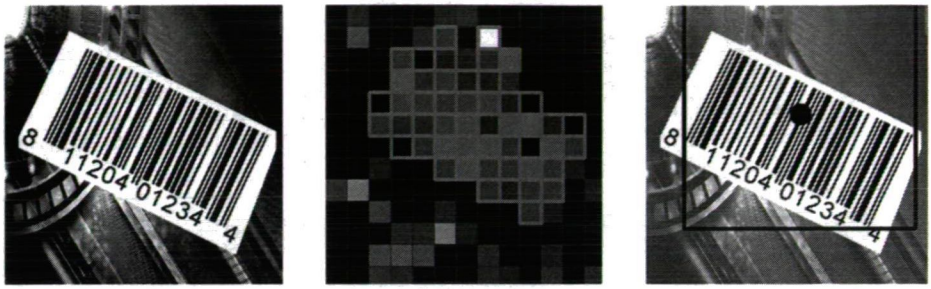
Figure 9: The idea of Local clustering. Here d is the maximum distance from the cluster center, i.e. the half of the cluster diameter

where V_{ij} is the contrast variance along a scanline j in a scanline pair i.

The rest of the method is the same as it is presented at Section 2.4.

2.6 Local Clustering

The main idea of local clustering is that an image region that contains a barcode segment has many similar stretched pixel clusters (Fig. 9). The minimum count of expected clusters can be derived from the widest bar of the barcode. Degree of stretch can be measured with the diameter of the cluster (defined as twice the distance of the furthest cluster point from the cluster center). For exactly horizontal or vertical lines, the largest cluster diameter is the tile size, in oblique situations, the largest cluster diameter is expected to be longer than that. Furthermore, stretched separate clusters need to be aligned approximately in the same direction, otherwise one cluster would touch another, thereby decreasing the number of separate clusters in a tile below our threshold. On real-life images that have low contrast at barcode areas, adaptive thresholding is necessary.



(a) original image

(b) feature matrix visualized as gray values. Red squares are above threshold.

(c) code center and bounding box

Figure 10: Stages of Local clustering

Another important property is the minimum cluster size measured in pixels. This can be easily computed from `min_barwidth` × `tile_size`.

After assigning the value to each tile, the same rule as in Section 2.4, applies for making the final decision, like dropping small groups of tiles and analyzing connected components.

Bounding boxes in our examples do not enclose the whole barcode in every case. This is because only the lower and upper bounds for the clusters in the feature matrix are calculated, and barcode corner pieces are too weak to signify a feature. The bounding boxes can be simply improved by finding aligned rectangles instead.

Running the method on the same scenario with different offsets yield different detection accuracy, which shows that this approach is sensitive to the choice of tiling. Further investigations are currently in progress on how to select the best tiling offset, as well as possible setups with overlapping tiles (which obviously increases computational requirements). This 2-phase approach works as follows. In the 1st phase, Local clustering is performed with zero-offset tiling, and in a 2nd phase the same step is done using an offset of half the tile size in both directions. The code centers detected by the 2 phases are pooled together with an extra filtering wherein those code centers that are detected in both phases and are close to each other are merged into one (the larger cluster is kept), because they are likely to correspond to the same code.

3 Evaluation

The discussed methods were tested on 100 images and accuracy has been compared to known barcode detection techniques, like the one based on bottom-hat filter [9], and another using morphological gradient [16].

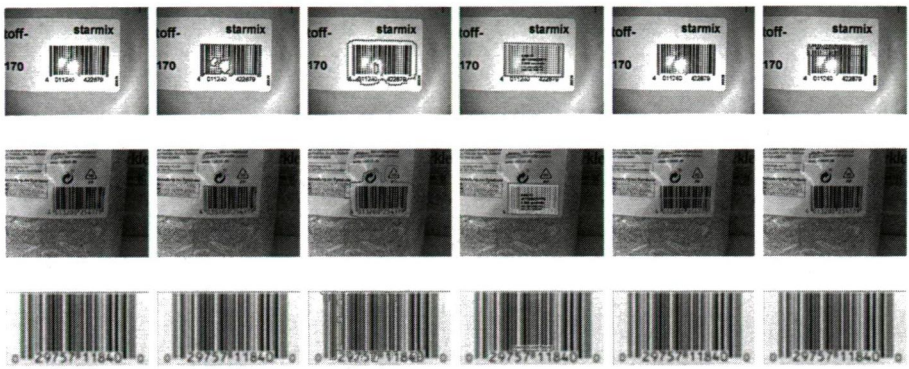


Figure 11: A qualitative test of barcode detectors from the state of the art and the market. From left to right: original image, Tuinstra et al. [16], Juett et al. [9], DataSymbol software, DTK barcode reader, BC tester. Each row represents a different test image. Results show that Datasymbol has the most robust localization approach, while BCTester had very poor accuracy even with clean and sharp images. The Tuinstra et al. and Juett et al. algorithms worked reasonably well, both has performed at least on the level of commercial softwares.

3.1 Test Suite

Since we have not found many official barcode detection test image databases, about 100 images of grocery product barcodes are made, with a Nokia N95 smartphone camera. Images has been downsampled to 640×480 px with bilinear interpolation. Minor reflections, blur, scratches and distortions were present in these images. We also found one barcode image database for comparative assessment, which was created by Tekin and Coughlan[1]. Ground truth to those images had been made manually, without marking the "quiet" zones and the digits that belong to the code.

There are several barcode detection software and frameworks on the market, like the DTK Barcode Reader SDK[2], BC Tester[3] and Barcode Recognition SDK of DataSymbol[4]. They do not indicate the applied algorithm behind their detection mechanism, however, a brief qualitative comparison has been made and results are shown in Fig. 11, and our results are shown in Fig. 12.

3.2 Implementation and Test Environment

The core of the proposed method has been implemented in `C++`, with the help of the `OpenCV` library. `C++` provides convenient object oriented approach and fast code execution, while `OpenCV` has all the functions needed for image preprocessing

[1] http://www.ski.org/Rehab/Coughlan_lab/Barcode
[2] http://www.dtksoft.com/
[3] http://www.bctester.de/
[4] http://www.datasymbol.com/

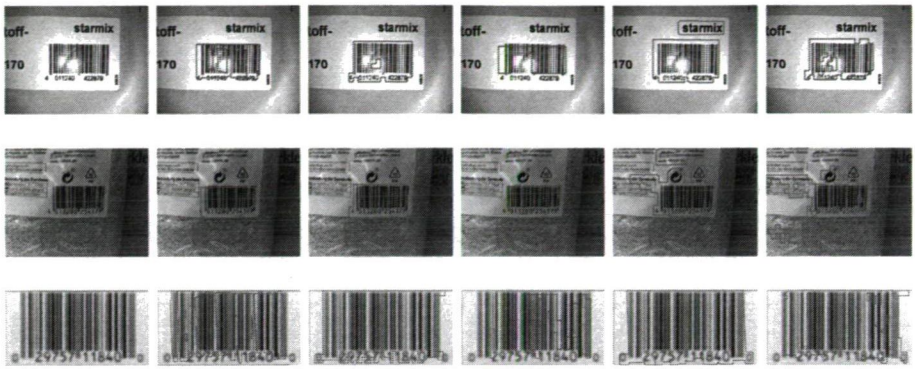


Figure 12: A qualitative test of barcode detectors based on our features. Contours of the possible barcodes are shown in red. From left to right: original image, Hough-transformation, distance transformation, local clustering, MIN–MAX, contrast measuring. Each row represents a different test image. Hough transformation is more tolerant to blur than noise, while MIN–MAX seemed to be the most robust approach. Distance transformation and local clustering also performed well, however, they showed a moderate amount of false-positives.

and manipulation. Evaluation is performed on a computer with Intel Core 2 Duo 3.00GHz CPU.

3.3 Accuracy and Detection Speed

For comparing the effectiveness of the proposed method, the most common measures like precision, recall, accuracy and F-measure (Eq. (3)) are used. The values are based on the Jaccard index (Eq. (4))

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3}$$

$$J(G, D) = \frac{\sum_{x,y}(G(x,y) \wedge (D(x,y))}{\sum_{x,y}(G(x,y) \vee (D(x,y))} \tag{4}$$

where G and D give binary 0–1 values based on the pixel intensity of the ground truth and the detector output images (both are binary).

The average performance indicators of the detectors are shown in Table 1. Ensemble efficiency of these features are presented in [5].

Distance transformation is better to be used as a weak "classifier" instead of on its own. It produces the highest amount of false positives, however, it comes with high recall. It is more like an exclusion filter for image parts than a detector.

The Probabilistic Hough method is a robust choice to localize barcodes, because it can be parametrized to minimum line length and maximum gap between line

Table 1: Average detection performance of the proposed method

Algorithm	Precision	Recall	Accuracy	F-measure	Runtime
Tuinstra et al. [16]	57.08 %	85.29 %	84.19 %	48.39 %	160 ms
Juett et al. [9]	34.26 %	94.08 %	72.76 %	36.13 %	230 ms
Hough trans.	64.83 %	85.07 %	84.22 %	58.76 %	230 ms
Distance trans.	20.00 %	95.85 %	54.52 %	23.54 %	190 ms
Local clustering	81.68 %	72.34 %	89.22 %	62.12 %	120 ms
MIN–MAX	43.36%	85.38 %	77.47 %	36.82 %	360 ms
Contrast measuring	51.17 %	88.02 %	82.58 %	49.07 %	140 ms

segments. It also handles noise well to a certain level, but it is quite sensitive to distortions.

Thanks to the nature of the Distance Transformation and Local Clustering methods, they are reliable on images with minor distortions, unlike the Hough transformation, which detects barcodes based on line angles.

The least sensitive method is MIN–MAX. Because of the morphological approach, it handles well noise, blur and distortions up to a relatively high level. However, the convolutions used in the steps of the algorithm make it realtively slow.

Partitioning the image, assigning a measure to each partition, and looking for homogenous areas in the feature image is a general approach to detect patterns. With different features, like contrast variance, histogram information or distance information, it can be used well as a barcode locator method.

4 Concluding Remarks

Several features have been presented for barcode localization in raster images using various features. We studied their behavior on a set of images showing different barcode types.

In industrial setups, processing of the image tiles and parallel execution of different methods may also be possible for further improve detection speed. Furthermore, intermediate feature data, like edge map can be used as input for other, more accurate classifiers discussed in the first section.

An ensemble of detectors specially devised for certain code types can significantly improve the overall accuracy.

References

[1] Adelmann, Robert. Toolkit for bar code recognition and resolving on camera. In *Phones – Jump Starting the Internet of Things. In: Informatik 2006 workshop on Mobile and Embedded Interactive Systems*, 2006.

[2] Ballard, D.H. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

[3] Bodnár, Péter and Nyúl, László G. Barcode detection with uniform partitioning and morphological operations. In *Conference of PhD Students in Computer Science, Proceedings of Conference*, pages 4–5, 2012.

[4] Bodnár, Péter and Nyúl, László G. Efficient barcode detection with texture analysis. In *Signal Processing, Pattern Recognition, and Applications, Proceedings of the Ninth IASTED International Conference on*, pages 51–57, 2012.

[5] Bodnár, Péter and Nyúl, László G. Improving barcode detection with combination of simple detectors. In *The 8th International Conference on Signal Image Technology (SITIS 2012)*, 2012.

[6] Canny, John. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, nov. 1986.

[7] Dillencourt, Michael B., Samet, Hannan, and Tamminen, Markku. A general approach to connected-component labeling for arbitrary image representations. *J. ACM*, 39:253–280, April 1992.

[8] Felzenszwalb, Pedro F. and Huttenlocher, Daniel P. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science, 2004.

[9] James Juett, Xiaojun Qi. Barcode localization using bottom-hat filter. *NSF Research Experience for Undergraduates*, 2005.

[10] Jean Paul Serra, Pierre Soille, editor. *Mathematical morphology and its applications to image processing.* Kluwer Academic, 1994.

[11] Joseph, Eugene and Pavlidis, Theo. Bar code waveform recognition using peak locations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(6):630–640, 1994.

[12] Kiryati, Nahum, Eldar, Yuval, and Bruckstein, Alfred M. A probabilistic hough transform. *Pattern Recognition*, 24(4):303–316, 1991.

[13] Lin, Daw-Tung and Lin, Chin-Lin. Multi-symbology and multiple 1d/2d barcodes extraction framework. In Lee, Kuo-Tien, Tsai, Wen-Hsiang, Liao, Hong-Yuan, Chen, Tsuhan, Hsieh, Jun-Wei, and Tseng, Chien-Cheng, editors, *Advances in Multimedia Modeling*, volume 6524 of *Lecture Notes in Computer Science*, pages 401–410. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-17829-0-38.

[14] Lin, Daw-Tung, Lin, Min-Chueh, and Huang, Kai-Yung. Real-time automatic recognition of omnidirectional multiple barcodes and dsp implementation. *Machine Vision and Applications*, 22:409–419, 2011. 10.1007/s00138-010-0299-3.

[15] Oktem, R. Bar code localization in wavelet domain by using binary morphology. In *Signal Processing and Communications Applications Conference, 2004. Proceedings of the IEEE 12th*, pages 499–501, april 2004.

[16] Tuinstra, Timothy R. *Reading Barcodes from Digital Imagery.* PhD thesis, Cedarville University, 2006.

[17] Wu, Sue and Amin, Adnan. Automatic thresholding of gray-level using multistage approach. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 493–497 vol.1, 2003.

[18] Youssef, Sherin M. and Salem, Rana M. Automated barcode recognition for smart identification and inspection automation. *Expert Systems with Applications*, 33(4):968–977, 2007.

Acta Cybernetica **21** (2013) 37–51.

Approximations of the Generalized Cascade Model

András Bóta*, Miklós Krész†, and András Pluhár‡

Abstract

The study of infection processes is an important field of science both from the theoretical and the practical point of view, and has many applications. In this paper we focus on the popular Independent Cascade model and its generalization. Unfortunately the exact computation of infection probabilities is a #P-complete problem [8], so one cannot expect fast exact algorithms. We propose several methods to efficiently compute infection patterns with acceptable accuracy. We will also examine the possibility of substituting the Independent Cascade model with a computationally more tractable model.

Keywords: computer science, infection process, heuristics

1 Introduction

The study of infection processes has many roots in various fields of research. The idea comes from the medical science for the purpose of modeling the spread of epidemics [13]. Boguña et al. [2] combined the infection processes with the emerging theory of Small World graphs, and gained new insights considering the survival of virulent diseases. Similar infection models can describe the spread of a behavior in social networks. One of the earliest models in sociometry, Granovetter's Linear Threshold model has proven to be an accurate description of information diffusion, see [15]. An important model in economics was developed by Domingos and Richardson [12] for the purpose of viral marketing. A form of the Independent Cascade model was later turned out to be an equivalent form of Granovetter's Linear Threshold model, see Kempe et al. [16, 17]. The exact computation of the vertex infection probabilities is a #P-complete problem [8], so applications are mainly using heuristics. Nevertheless, the IC model was also adopted to many other applications including costumer churn and the spread of credit default, [11, 4].

In later applications generalized IC models were used, in which the vertices become infected according to a general *a priori* probability distribution, before infecting their neighbors. Moreover, in these models an "infection" does not necessarily mean infection in the original sense, e.g. a bankruptcy of a company may

*University of Szeged, E-mail: bandras@inf.u-szeged.hu
†University of Szeged, E-mail: kresz@jgypk.u-szeged.hu
‡University of Szeged, E-mail: pluhar@inf.u-szeged.hu

be caused by the poor economic state of its partners, but one cannot pinpoint the most responsible infector [19].

The initial problem described by Domingos and Richardson was influence maximization, that is to find a specific set of k individuals for a given $k \in N$ yielding the largest expected infection. Kempe et al. [16] proved that the exact solution is an NP-hard problem, but a greedy algorithm results in a k element set with expected influence at least $(1 - 1/e)Opt_k$ where Opt_k is the maximum influence for all set of size k. Still, a considerable amount of effort was spent to improve both the quality of the solution and the speed of the heuristics, [7, 8].

The computation of the infection process requires an input graph with edge weights corresponding to infection probabilities. However, in most practical applications the edge weights are unknown a priori, thus they are randomly generated or guessed. A more systematic approach appeared in the paper by Bóta et al. [5], in which the authors have proposed the *inverse infection problem.* That is, given the set of initial infectors (or a priori infection probability distribution in the general case), the output of the infection model (a posteriori infection probability distribution) and a graph structure, can we compute the infection probabilities? A different approach for the Linear Threshold model was developed by Cao et al. [6].

Both the infection maximization and inverse infection algorithms are based on the repeated computation of the IC model. Thus, the fast computation of this model is a requirement for any algorithm that tries to solve the above mentioned problems.

There are several existing algorithms for this purpose, each focusing on a different aspect of the model. We propose new methods, motivated by our observations on real economical data [11, 4]. These networks function fundamentally differently than social or interaction networks. More precisely, the spreading of credit defaults is quite different compared to the behavior of influenza or information diffusion. In general, a single defaulted company has little effect on its neighbors, since most companies have a wide array of dependencies both on business partners, customers and subcontractors. However, if a large number of companies go default at the same time, the effect on other companies becomes gradually greater. According to this, on the examined networks the edge infection probabilities are low, typically below 0.2, or even smaller.

In this paper we are going to describe three methods exploiting the above aspect of the problem.

- The Edge Simulation method is a combination of both simulation and exact computations that decreases the standard deviation appearing in other simulations.
- If the infection probabilities are small, then the infections typically do not travel far from the source of infection. Neighborhood Bound Heuristics exploits this property.
- The Independent Cascade model itself can be substituted for a similar, but a computationally more tractable model.

The rest of this paper is organized as follows. First we define the original IC model and generalize it to suit complex models. We will also give a short description of the inverse infection problem. Then we describe the above mentioned methods in detail. Finally we compare the results with respect to the speed and accuracy of the computations.

2 Problem definition

In this section we will define the Independent Cascade model, the Generalized Cascade model and the inverse infection problem.

The pair $G = (V, E)$ is a directed graph, where V denotes the set of nodes and $E \subset V \times V$ denotes the set of directed links. If there is a $w_{u,v}$ defined for each edge (u, v), we have a weighted graph. When dealing with infection models, we restrict the values of the weights to be $w_{u,v} \in [0, 1]$ for each edge (u, v). We will also refer to the weights as infection probabilities.

The *Independent Cascade model* is an iterative method based on the process of active nodes infecting inactive ones. The process starts with an initially active set of nodes $A_0 \subset V(G)$. Let $A_i \subseteq V(G)$ be the set of nodes newly activated in iteration i. In iteration $i + 1$, every node $u \in A_i$ has one chance to activate each of its inactive out-neighbors $v \in V \setminus \cup_{0 \leq j \leq i} S_j$ according to $w_{u,v}$. If the attempt is successful, then v becomes active in iteration $i+1$. If more than one node is trying to activate v in the same iteration, the attempts are made in an arbitrary order and independently of each other still in iteration $i + 1$. The process terminates at step t if $A_t = \emptyset$.

The *Generalized Cascade model* extends this process in the following way. Each $v \in V(G)$ has an *a priori infection probability* w_v, and the vertices become active independently of each other with their assigned probabilities at the beginning. After this, the infection process of the IC model applies with a randomly selected active set. In this process, each vertex gets an *a posteriori infection probability* w'_v corresponding to the probability of infection at the end. In other words, the process can be interpreted as a transformation of probability distributions over the graph G.

The Generalized Cascade model can be summarized as follows: Given a graph G, the edge infection probabilities $w_{u,v}$ for all $(u, v) \in E(G)$, and the a priori probabilities w_v for all $v \in V(G)$, we are looking for the a posteriori infection probabilities w'_v for all vertices $v \in V(G)$.

All these applications of infection models require the edge probabilities to be given beforehand. However, in practice these values are unknown. Therefore it is necessary to develop an algorithm that is capable of *learning* these weights. That is, if we have the a priori and a posteriori distributions of the GC model as the input of an algorithm, it should assign edge weights $w_{u,v}$ for all edges (u, v), such that the a priori distribution is transformed into a good approximation of the a posteriori distribution. Some algorithms for the above problem were investigated in [5]. Note that all of these methods require the repeated computation of the

Generalized Cascade model.

The exact computation of the vertex infection probabilities is difficult [8]. Some authors have tried to circumvent this by developing algorithms for special classes of graphs, like trees or DAG-s [7]. In this paper, we will describe four methods for approximating the a posteriori infection probabilities in the GC Model.

3 Methods

The methods presented below fall into three categories. *Simulations* are Monte Carlo generators, i.e. they compute multiple realizations of the probabilistic infection process, and count the relative frequency of vertex infections.

In contrast to this, *heuristics* use an approximation of the GC model, circumventing the #P-completeness, but still dealing with both the a priori and the edge infection probabilities. Finally, *hybrid methods* use some combination of both approaches.

3.1 Frequency based simulations

The notion of using Monte Carlo based simulations to compute the a posteriori infection distribution comes from Kempe et al [17]. That is, generating random edges and vertex infections, according to the edge weights and the a priori vertex infections, and then approximate the infection probability of vertices by the relative frequency of vertex infections.

All methods described in this section use the above approach. The original method of Kempe et al. can be adapted to the requirements of the Generalized Cascade model. We will refer to this as **Complete Simulation**. In contrast to this, **Edge Simulation** is a hybrid design using heuristics to improve the performance of the simulation.

3.1.1 Previous works

Kempe, Kleinberg and Tardos developed a method based on reachability [17] to compute the Independent Cascade model. They construct an unweighted directed graph G' on the same vertex set as G by drawing the edge (u, v) independently of the other edges with probability $w_{u,v}$. The resulting graph G' can be interpreted as a realization of the possible routes of infection. Those vertices that can be reached from an initially infected vertex also become infected. This way the computation of the iterations one by one can be avoided, and the problem of a single simulation step is reduced to a simple (path) searching problem. Note that this formulation helped to show that $f(A)$, the expected infection of a set A is a *submodular function*, that is $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$ for all $v \in V(G)$ and $S \subset T \subset V(G)$.

The process can be applied to G multiple times, resulting in a series of realizations. The desired distribution can be approximated simply by counting the relative frequency of vertex infections.

3.1.2 Complete Simulation

It is easy to adapt the method of Kempe et al. to the needs of the Generalized Cascade model. In addition to the construction of G', a set $A_0 \subseteq V(G)$ is created, according to the following rule: for all $v \in V(G)$, $v \in A_0$ according to the probability w_v. G' and A_0 is generated multiple times. We will refer to this value as the sample size k. Finally let f_v denote the relative frequency of infection for vertex v.

Algorithm 1 Complete Simulation

Input: Graph G, sample size k
Output: Relative frequency of infection f_v for all $v \in V(G)$

1: $j \leftarrow 0$
2: **for all** $v \in V$: $f_v \leftarrow 0$
3: **while** $j < k$ **do**
4: Generate G'
5: Generate A_0
6: $S \leftarrow \emptyset$
7: $Q \leftarrow V(G')$
8: **for all** $u \in A_0 \cap Q$ **do**
9: **for all** $v \in Q$ **do**
10: **if** there is a path $u, \ldots, v$ in graph G' **then**
11: $S \leftarrow S \cup \{v\}$
12: **end if**
13: **end for**
14: $Q \leftarrow Q \setminus S$
15: **end for**
16: **for all** $v \in S$: $f_v \leftarrow f_v + 1$
17: $j \leftarrow j + 1$
18: **end while**
19: **for all** $v \in V$: $f_v \leftarrow \frac{f_v}{k}$

At line 10, we allow $u = v$. In the main loop, it is indifferent which node in A_0 is the source of the infection. Since a node can be infected only once, any node reachable from a node from A_0 can be left out of further computations. Because of this, the for loops in the algorithm can be interpreted as a single search on G' starting from the nodes in A_0.

Before the main loop, f_v is initialized for all vertices. It is easy to see, that the algorithm computes the relative frequencies for all vertices correctly. Since this method is based on frequency counting, its precision and time complexity depends greatly on the sample size k. We will elaborate on this in Section 4.

3.1.3 Edge Simulation

There is a different way to simulate the generalized model: instead of computing A_0 we can deal directly with the a priori vertex infection probabilities. The notation

is the same as before, w_v denotes the a priori infection probability of vertex v. At line 8, we allow $u = v$.

Algorithm 2 Edge Simulation

Input: Graph G, sample size k
Output: Relative frequency of infection f_v for all $v \in V(G)$

```
1: j ← 0
2: for all v ∈ V: f_v ← 0
3: while j < k do
4:    Generate G'
5:    for all v ∈ V(G) do
6:       s ← 1
7:       for all u ∈ V(G) do
8:          if there is a path u, ..., v in graph G' then
9:             s ← s(1 − w_u)
10:         end if
11:      end for
12:      f_v ← f_v + 1 − s
13:   end for
14:   j ← j + 1
15: end while
16: for all v ∈ V: f_v ← f_v / k
```

In contrast to the previous method, the source of the infection does matter, since the value w_v is not the same for different vertices. We also do not have any restrictions on the structure of G other than being simple, so any vertex can be a part of a loop. This means, that in order to avoid counting a single w_v multiple times, we have to do a search for each node independently. Obviously the above approach increases the time complexity in general, but if the edge probabilities are low enough in G, then G' has many small components, reducing the running time for each search significantly.

It is clear, that the computational time is greater for ES compared to CS. In exchange, we can expect, that the precision of the approximation is less dependent on the sample size. We will elaborate on this in Section 4.

3.2 Neighborhood Bound Heuristic

There are several existing heuristics for the IC model [7, 8, 10, 9] covering a large area of performance requirements. These methods usually exploit one or more properties of the infection process. Chen's DAG and LDAG algorithm for example focuses on the concept, that the edges with high infection probabilities carry the bulk of the infection process. They propose to construct a local directed acyclic graph containing the relevant edges for each node, and then compute the infection process using the DAG.

The main idea of our method is similar: the construction of a small graph containing all of the possible paths of infection inside a given neighborhood for a given vertex. This graph is created in such a way, that the approximation of the a posteriori infection probability of the corresponding vertex can be easily computed. It is important to emphasize, that the goal of this method is the approximation of the GC model as fast as possible, disregarding requirements for precision.

We are also going to rely on the findings in [11, 4]. Based on the examination of economic networks, the authors have found, that the edge infection probabilities are small, typically below 0.2. The above observation greatly limits "the travel distance" of an infection event, since even if we consider a path of length two from the source of the infection, the probability of infection is reduced to 0.04 or below.

Based on the remarks above, we propose the Neighborhood Bound Heuristic (NBH). We will denote the set containing the in-neighbors of vertex v as $N^-(v)$. For all $v \in V(G)$ we are going to construct a weighted, rooted tree T_v with v as the root, and edges pointing towards v. In the first step all vertices $u \in N^-(v)$ are added to T_v, as well as the edges $(u, v) \in E(G)$ for all $u \in N^-(v)$. In the second step we are going to deal with the second neighborhood of v. For all $u \in N^-(v)$ we are going to add the nodes $z \in N^-(u) \setminus \{v\}$ and all of the edges $(z, u) \in E(G)$ to T_v. The subtractions of v is necessary to avoid loops of length two. For all edges in T_v we keep the edge weights from G. Take note, that nothing prohibits the nodes of G (with the exception of v) from appearing multiple times in T_v, this corresponds with our idea of representing all possible infection paths in the second neighborhood of v.

The computation of the a posteriori infection probability of v in T_v is easy. T_v has three levels: the leaves, $N^-(v)$ and the root v. The a posteriori infection probabilities of the leaves are the same as their a priori ones, since they do not have in-neighbors. A node $u \in N^-(v)$ gets infected if one of the leaves connected to it is infected, or becomes infected by itself, meaning $w'_u = 1-(1-w_u)\prod_{z \in N^-(u)}(1-w'_z)$. The computation is executed in the same fashion for v.

The above method is extremely fast, since w'_v can be computed in a single run on the edges of T_v, and if G is sparse, $|E(T_v)|$ is small. The construction of T_v is simple, if we limit the process to the second neighborhood of v in G. If we consider larger neighborhoods, nothing prohibits $|V(T_v)|$ from growing exponentially, and it becomes increasingly difficult to avoid loops. Finally, if we consider the fact, that the edge weights are small, then the loss of precision is still within acceptable bounds.

3.3 Aggregated Linear Effect Model

Our goal in this section is to build up a model that more or less approximates the mechanism of the Generalized Cascade model. We begin with some motivations and define the *Aggregated Linear Effect Model*, shortly ALE model afterward. For a weighted graph G, let the a priori infection of a vertex v be w_v. If one considers only one step of the linear effects at v, it can be defined as $\sum_{u:u \in N(v)} w_{u,v} + w_v$. This is nothing else, but $x + Bx$, where B is the transpose of A, the weighted

incidence matrix of G, and x is the vector of a priori infection probabilities.

In the 2nd, 3rd, ... ith steps we may aggregate the effects of the second, third etc. neighborhoods by adding the B^2x, B^3x, ..., B^ix correction terms to the approximation.

3.3.1 Definition of ALE model.

Let $B = A^T$, where A is the weighted incidence matrix of graph G, and x is the vector, such that $x_v = w_v$, then the a posteriori effect y is defined as

$$y := (I + B + B^2 + \ldots)x,$$

where I is the identity matrix.

Note, that $\sum_{u:u\in N(v)} w_{u,v} + w_v$ is close to the probability of v getting infected independently in one step by itself or by a neighbor. In general, if the weights are small, the error is negligible. Assuming small weights, the infinite series also converges, and we have the more compact form of

$$y := (I + B + B^2 + \ldots)x = (I - A)^{-1}x.$$

The values w_v and $w_{u,v}$ do not have to be probabilities any more, we might consider any appropriate scalar functions (perhaps after scaling in order to maintain convergence). Now the following questions arise:

How good is the ALE model? How to compute (and later on use) it efficiently?

There are two obvious ways to test the first question. One is to consider some real problems, make a network model out of those, and compute the optimal weights in an ALE model such that the model "prediction" are as close to the real values as possible. However, since the IC models already performed well in such case studies, we might consider here an easier way. We just take a weighted G with some a priori infection x, compute the a posteriori infection y by the associated IC model, and then try to find appropriate new weights such that for the a posteriori effect y' by the ALE model we get $\min ||y - y'||$ in a fixed norm.

Once we have the appropriate weights, that is the matrix B, we can compute y easily. Of course not by inverting $I - B$ but solving the equation $(I - B)y = x$ for y by Gaussian elimination.

4 Results

For the purposes of evaluation we have used sparse graphs generated with the forest fire model [18], with $|G(V)| = n = 1000, \ldots, 40000$. This allows us to examine how much the computation time of a given algorithm scales with graph size. Since the performance of most methods described here depends on the size of the infection probabilities, we have used five arrangements of edge weights and a priori distributions.

Setup	A	B	C	D	E
ℓ_1	0.02	0.05	0.1	0.2	0.5
ℓ_2	0.1	0.1	0.2	0.2	0.5

Table 1: Experiment setups.

- Each edge weight is drawn independently from an uniform distribution between $(0, \ell_1)$.
- The expected size of the set of random infectors is $n * \ell_2$. For each vertex, there is an a priori infection probability drawn independently from an uniform distribution between $(0, 0.2)$.
- For all other vertices $w_v = 0$.

Using the above process, for each of the unweighted graphs, we have created five weighted ones with a priori probabilities, resulting in $9 \times 5 = 45$ different benchmark networks. The parameters ℓ_1 and ℓ_2 of each arrangement can be seen in table 1.

We have mentioned in the introduction, that our aim is the analysis of economic networks. We have also mentioned our findings, that infection probabilities in these networks are usually small. Based on this, we consider the probability arrangements A through D to be within our area of interest. Setup E is used to measure the behavior of described methods outside these conditions.

In this section, we are going to evaluate the running time and precision of our methods[1]. While the evaluation of computational time is rather straightforward, we have to make a distinction while measuring performance.

Simulations are Monte Carlo based estimations of the infection process. The goodness of an estimation is governed by the sample size k. From the Law of Large Numbers it follows that if k goes to infinity then the simulationed values converge to the infection probability. Depending on k, the output of the simulations differ from each other, characterized by their deviation. It is important to emphasize, that the deviation only depends on k. It does not depend on the graph size or the experiment setup.

On the other hand, heuristics apply a process similar to the infection model to approximate its output in reasonable time. The goodness of the approximation can be measured by comparing it with a simulation computed with a k large enough to minimize deviation. The difference between these can be described by an error function[2]. The error of these heuristics is highly dependent on the sizes of the edge weights as well as the size of the network.

Based on the above facts, we will evaluate the precision of the simulations and the heuristics somewhat differently.

[1]We have implemented the methods in JAVA, and we have used a computer with an Intel i7-2630QM processor, and 8 gigabytes of memory.

[2]For this purpose we have used the root mean squared error function (RMSE).

4.1 Deviation of the simulations

The most important question of simulation is the relation between the sample size and the accuracy of the simulation. This can be measured as the standard deviation between the a posteriori infections [3]. Increasing k obviously worsens the running time of the simulation, so it is desirable to find a balance between accuracy and complexity.

Let us make some heuristics concerning the expected results of CS. The worst case for the variance is when the characteristic function X_v of the infection of a vertex v follows a Bernoulli distribution. Then the standard deviation of X_v is $\sigma_v = \sqrt{p_v(1-w_v)}$. The standard deviation of the k-element sample is

$$\frac{\sqrt{kw_v(1-w_v)}}{k} = O\left(\frac{1}{\sqrt{k}}\right),$$

that is to get one more correct digit in the outcome one needs one hundred times more iterations.

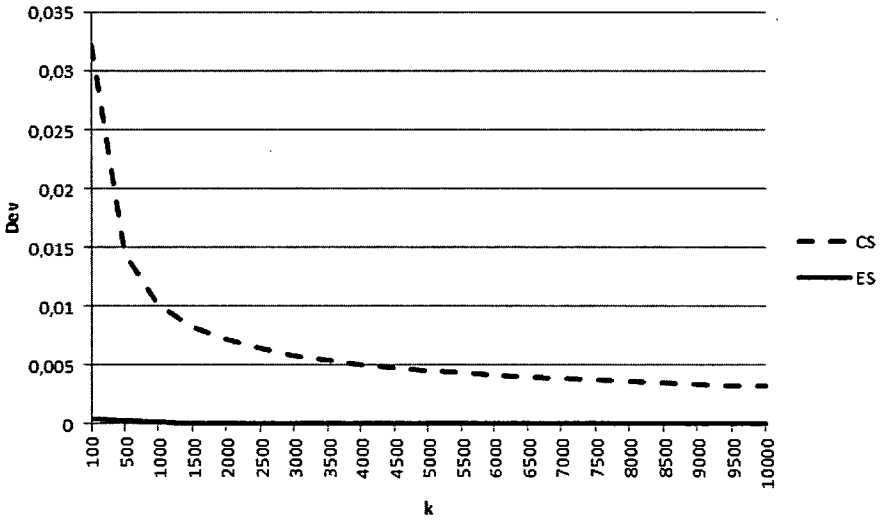


Figure 1: The absolute deviation compared to the sample size. In order to evaluate very large values of k, we have used a small benchmark network.

Of course, $X_v = Y_v + Z_v - (Y_v Z_v)$, where Y_v and Z_v are the characteristic functions of the a priori infection and the infection caused by the network, respectively. We might assume Y_v and Z_v independent of each other, and Y_v follows Bernoulli distribution. Now, if we simulate the edges and use the vertex a priori infection

[3]For testing purposes, we have used Zachary's karate club network [20] with edge weights between 0 and 0.5 and four initially infected nodes. The a priori infection probabilities of these nodes were drawn independently from an uniform distribution between 0 and 1.

probabilities directly, then the approximation of p_v is significantly improved. Indeed, an easy computations gives that

$$Var(X_v) = Var(Y_v) + Var(Z_v) \leq \mathbb{E}[Y_v](1 - \mathbb{E}[Y_v]) + Var(Z_v).$$

However, if we handle Y_v as a constant of value $\mathbb{E}[Y_v]$ then $Var(X_v)$ drops to $Var(Z_v)$. The infection coming for the network is usually much smaller than the a priori infection, and since it is the sum of almost independent variables, the variance of X_v must be even much smaller.

Figure 1 shows the standard deviation compared to the sample size k for both methods. The deviations are averaged for all nodes of the network. It can be seen, that the empirical results for CS roughly correspond to our heuristics. It can also be seen, that even for small values of k ES performs better by magnitudes than CS. In fact, the deviation goes below 10^{-5} for $k = 1500$, while using CS it goes below 10^{-4} only for $k = 100000$.

4.2 Precision of the heuristics

In order to measure the accuracy of NBH and ALE, we have used a very accurate Edge Simulation with $k = 5000$ as a benchmark. It can be seen above that the expected error of this simulation is less than 10^{-5}. Root mean squared error is used to measure the difference between the a posteriori distributions.

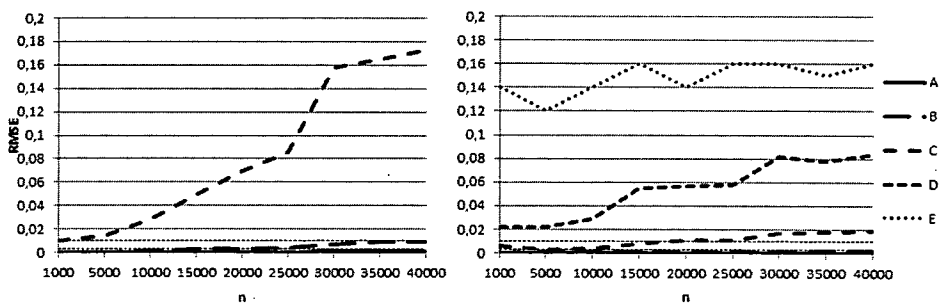


Figure 2: The RMSE of ALE (left) and NBH (right) for all experiment setups. The small dashed lines indicate the standard deviation of CS for $k = 10000$ and $k = 1000$.

On the left hand side of Figure 2, we can see the performance of ALE compared to the benchmark. If the edge weights are small enough (below 0.1), ALE is able to approximate the GC model with reasonable accuracy. For higher weights the performance of ALE gradually worsens to the point, that for the last two setups it is not able to produce output within acceptable bounds. The accuracy of the method also depends on the size on the graph, although for the first two setups, this is barely noticeable. The situation does not so grim if we consider our experiences

with the edge weights of economic networks, which typically fall into the first two category.

If we take a look at the performance of NBH on the right, we can see, that if the edge weights are below 0.2 (Setup D), the error remains below 0.1. It is easy to see, that lower edge infection probabilities produce more accurate approximations, which confirms our expectation, that if these probabilities are low enough, infections do not travel far. Like ALE, NBH is also slightly dependent on the size of the network. It is also clear, that NBH performs better than ALE in general, producing lower error levels for the same arrangements of infection probabilities.

We can compare these result to the deviation of the simulation based methods. The small dashed lines on both parts of Figure 2 indicate the standard deviation of CS with $k = 10000$ (lower) and $k = 1000$ (higher). Take note, that according to Figure 1, the deviation of ES is lower, even for $k = 100$. Based on the above fact, we can say, that the performance of both methods is comparable to the simulations only if the infection probabilities are small enough.

4.3 Computational time

On Figure 3 we can see the computational time of our methods on four probability arrangements. The results for setup A and B were almost the same, so we have left out the former one.

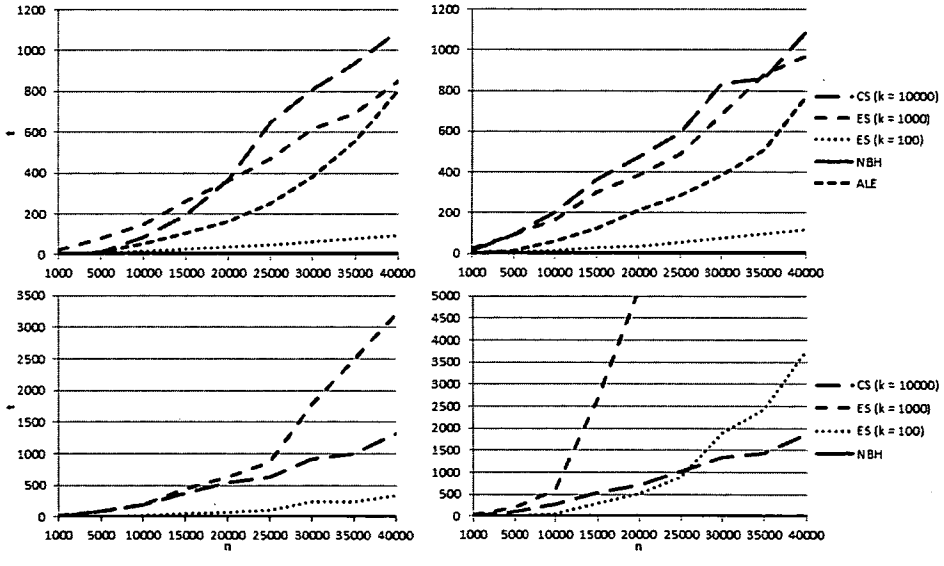


Figure 3: The running time of the algorithms measured in seconds, compared to the size of the graphs. Experiment setup B (upper left), setup C (upper right), setup D (lower left) and setup E (lower right).

The running time of CS scales more or less linearly with graph size, indepen-

dent of probability setups. However, the deviation of CS only decreases below an acceptable level if k is sufficiently high. This means, that even tough the computation of a single G' is fast, a large amount of realizations have to be generated in order to compete with other methods. As a consequence, CS is the slowest of the methods on the first three experiment setups. Its use is only recommended if the probabilities are too high for the other algorithms to tackle.

Unlike CS, the performance of ES is highly dependent on the size of infection probabilities. The computational time gradually worsens with the increase of edge weights, to the point where even $k = 100$ is infeasible. This corresponds with our remarks in Section 3.1.3. Take note though, that the deviation of ES is smaller than CS by magnitudes, meaning an ES with $k = 100$ outperforms a CS with $k = 10000$ in terms of precision. In our area of interest, ES seems to be a reasonable compromise between precision and computational time.

Due to its local nature, NBH is the fastest of the methods. Even on the largest graphs it completes the task in less than two seconds for all experiment setups. As we have seen in the previous section however, this comes at a cost of decreased precision, on all but the smallest probability arrangements. As a consequence, its use is only recommended if the network is large, and the infection probabilities are low.

ALE scales similarly compared to the simulations with a slight increase in performance for the first three probability arrangements, but it is unable to produce meaningful output on the last two setups. If we take the findings of the previous section into account, then we can conclude, that ALE should only be used if the infection probabilities are very low.

5 Conclusions

In this paper we have described four different methods able to compute the Generalized Cascade model. All of these methods are based on the observation, that in realistic networks, the edge infection probabilities are low.

The Edge Simulation is a Monte Carlo based method that greatly reduces the variance of the resulting a posteriori distribution. Tests indicate that the variance can be reduced by two magnitudes with the same sample size. Unfortunately the method requires more computations than other simulations, and its speed depends on the edge infection probabilities.

The Neighborhood Bound Heuristic limits the effect a node has on another one to a distance of two. Its accuracy is surprisingly good in graphs with small weights, but even in this case, care must be taken with its use.

The direct adaptation of the method developed by Kempe et al. can be an acceptable choice if the infection probabilities are high. The computational time of CS is independent of the former, and scales linearly with both the size of the network, and the number of samples.

The Aggregated Linear Effect is a model that tries to approximate the mechanism of the Generalized Cascade model, but is computationally more tractable

only if the edge weights are small enough. It is also important to note, that the inverse infection problem can be solved directly with the use of ALE, avoiding the additional cost of a learning algorithm.

Acknowledgments. The first and third authors were partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TÁMOP-4.2.2.C-11/1/KONV-2012-0013).

The second author was partially supported by the European Union and co-funded by the European Social Fund through project HPC (grant no.: TÁMOP-4.2.2.C-11/1/KONV-2012-0010).

The second author was also supported by the Gyula Juhász Faculty of Education, University of Szeged (project no. CS-004/2012).

References

[1] R. Albert, A. L. Barabási, Statistical mechanics of complex networks. *Reviews of Modern Physics*, **74** (2002) 47–97.

[2] M. Boguña, R. Pastor-Satorras, A. Vespignani, Absence of epidemic threshold in scale-free net-works with degree correlations. *Phys. Rev. Lett.* Vol. **90** No 2. (2003) 028701-1–4 .

[3] B. Bollobás, *Modern Graph Theory,* Springer, New York (1998).

[4] A. Bóta, L. Csizmadia and A. Pluhár, Community detection and its use in Real Graphs. *Proceedings of the 2010 Mini-Conference on Applied Theoretical Computer Science - MATCOS* **10** (2010) 95–99.

[5] A. Bóta, M. Krész and A. Pluhár, Systematic learning of edge probabilities in the Domingos-Richardson model. *Int. J. Complex Systems in Science* Volume **1(2)** (2011) 115–118.

[6] Tianyu Cao, Xindong Wu, Tony Xiaohua Hu and Song Wang, Active Learning of Model Parameters for Influence Maximization. *Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science, eds. Gunopulos et al., Springer Berlin/Heidelberg, (2011) 280–295.

[7] Wei Chen, Yifei Yuan and Li Zhang, Scalable Influence Maximization in Social Networks under the Linear Threshold Model. *Proceeding ICDM '10 Proceedings of the 2010 IEEE International Conference on Data Mining*, IEEE Computer Society (2010) 88–97.

[8] Wei Chen, Chi Wang and Yajun Wang, Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks. *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2010) 1029–1038.

[9] M. Kimura, K. Saito, Tractable models for information diffusion in social networks. *Knowledge Discovery in Databases*, Lecture Notes in Computer Science Springer Berlin / Heidelberg, (2006), 259–271.

[10] W. Chen, Y. Wang, S. Yang, Efficient influence maximization in social networks. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2009) 199–208.

[11] A. Csernenszky, Gy. Kovács, M. Krész, A. Pluhár, T. Tóth, The use of infection models in accounting and crediting. *Challenges for Analysis of the Economy, the Businesses, and Social Progress, Szeged* 2009.

[12] P. Domingos, M. Richardson, Mining the Network Value of Costumers. *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining*, ACM (2001) 57–66.

[13] O. Diekmann, J. A. P. Heesterbeek, Mathematical epidemiology of infectious diseases. Model Building, Analysis and Interpretation. *John Wiley & Sons*, 2000.

[14] M.E.J. Newman, The structure and function of complex networks. *SIAM Review* **45**, (2003) 167-256.

[15] M. Granovetter, Threshold models of collective behavior. *American Journal of Sociology* **83**(6) (1978) 1420–1443.

[16] D. Kempe, J. Kleinberg, E. Tardos, Maximizing the Spread of Influence though a Social Network. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2003) 137–146.

[17] D. Kempe, J. Kleinberg, E. Tardos, Influential Nodes in a Diffusion Model for Social Networks. *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, Springer-Verlag (2005) 1127–1138.

[18] J. Leskovec, J. Kleinberg, C. Faloutsos, Graphs over time: densification laws, shrinking diameters and possible explanations. *Proceedings of the1th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2005) 177–187.

[19] M. Krész and A. Pluhár, Prediction of Economic and Social Events by Infection Processes. To appear in *Encyclopedia of Social Network Analysis and Mining*, Springer (2012).

[20] W. W. Zachary, An information flow model for conflict and fission in small groups. *Journal of Anthropological Research* **33** (1977), 452–473.

Acta Cybernetica 21 (2013) 53–73.

Application Oriented Variable Fixing Methods for the Multiple Depot Vehicle Scheduling Problem

Balázs Dávid* and Miklós Krész†

Abstract

In this article, we present heuristic methods for the vehicle scheduling problem that solve it by reducing the problem size using different variable fixing approaches. These methods are constructed in a way that takes some basic driver requirements into consideration as well. We show the efficiency of the methods on real-life and random data instances too. We also give an improved way of generating random input for the vehicle scheduling problem.

Keywords: vehicle scheduling, variable fixing

1 Introduction

The multitude of problems arising in public transportation forms a complex system. These problems can be categorized into two main groups: vehicle scheduling problems and driver scheduling problems. However, these two sets can not be considered totally independently of each other, as vehicle schedules are needed as the basis of constructing driver schedules. Literature addresses the hierarchy between these tasks either by using an integrated or a sequential approach.

Integrated models for the combined vehicle- and driver scheduling problem have become more and more efficient lately [7, 11], and based on them, useful methods exist for the long-term planning of a transportation company's schedule. However, these solutions are still not fast enough for being considered as part of an interactive decision support system.

A sequential approach of the optimization problem breaks it down into a series of different tasks, which are solved one after the other. Such a system can have many different sub-problems, as the one seen in [4], but usually 3 phases have to be carried out: vehicle scheduling, driver scheduling and driver rostering. Fast methods that give efficient solutions exist for all these sub-problems. The main drawback is that the majority of these methods consider solving a stand-alone problem only, and does not deal with its integration into a larger system. However, breaking down the main problem into different smaller tasks results in a flexible

*University of Szeged, E-mail: davidb@inf.u-szeged.hu
†University of Szeged, E-mail: kresz@jgypk.u-szeged.hu

system, so this approach can be an ideal choice to be used for interactive decision support.

In this paper, we will deal with the vehicle scheduling problem as part of such a system. We give a brief overview of the problem itself, and analyze the requirements for real-life applications. We examine a heuristic by Gintner et al. [12], and develop new algorithms using their idea as a basis. We provide test results for all examined methods on real life and random data as well. We also introduce a new way of creating random data based on a method by Carpaneto et al [9]. This new algorithm is needed because the structure of the instances generated by the method in [9] is different from real-life instances of Hungarian transportation companies. We wanted to test our methods on random data that more closely resembles the structure of real-life Hungarian instances.

2 The Vehicle Scheduling Problem

In this section, we present the vehicle scheduling problem, and give an overview of its important models and methods from literature.

Let V be the set of vehicles, and T the set of trips. For each $t \in T$ trip, let $dt(t)$ and $at(t)$ be its departure and arrival time, and let $sl(t)$ and $el(t)$ denote its starting and ending location respectively. Two trips $i, j \in T$ are compatible if there is enough time between $at(i)$ and $dt(j)$ to cover the distance between $el(i)$ and $sl(j)$. A vehicle schedule can contain compatible trips only. A vehicle task is called a deadhead trip if the vehicle changes its location without executing a trip.

The aim of the vehicle scheduling problem is to assign vehicles to execute the trips of a given timetable. These assignments must satisfy certain conditions:

- Every trip must be executed exactly once.
- Trips assigned to a vehicle must be compatible with one another.
- The cost of the assignment must be minimal.

We examine the possible components of the cost function in the following subsection.

2.1 Costs of a Vehicle Schedule

One of the most important steps is to determine the costs arising in our problem. There are several operational costs that can be taken into consideration for a vehicle:

- **Trip distance cost:** the cost of a vehicle to cover a unit distance (usually 1 km) while executing a trip.
- **Deadhead distance cost:** the cost of a vehicle to cover a unit distance while executing a deadhead trip.

- **Daily cost of a vehicle**: the cost of making a vehicle available for use on the given day.

Trip and deadhead distance costs can easily be determined, as the gas consumption of the vehicles is a known parameter. Daily costs are more difficult to give, as they incorporate a number of different smaller costs (eg. maintenance, upkeep, etc.), but the cost of renting a vehicle of the same type for one day can be a good estimate. Using these operational costs only, the problem can be solved in different ways:

- If only the daily costs are used in the objective, the number of vehicles will be minimized.
- If only the deadhead costs are included, the amount of deadhead trips will be minimized, thus minimizing the location changes.
- To get an overall operational cost, all of the above costs have to be included in the objective function.

As we mentioned earlier, the solution of the vehicle scheduling problem alone is just the basis of a daily schedule of a transportation company. Using the above defined operational costs only does not necessarily give a good solution if we want to use it in an "application oriented" way. The vehicle schedules have to be assigned to drivers, who also have a set of constraints for their driver schedules. Driver rules have many different types, and the regulations also vary from country to country, and from company to company. One of the most important constraints that all of them have in common is the limitation of the maximum continuous driving time of a driver. Because of this, vehicle drivers must be assigned a break after a fixed amount of driving time at most.

If the solution of the vehicle scheduling problem produces too "dense" schedules (which do not have enough gaps between tasks to assign these breaks), then the driver scheduling algorithms have to transform the schedules. This means extra computation time for the driver algorithm, and it also means that the underlying vehicle schedule will also be changed significantly, modifying its cost retroactively. This aspect must also be considered, when solving the vehicle scheduling problem.

2.2 Single and Multiple Depot

In some cases the vehicles can also be classified into depots. A depot of a vehicle can mean its starting geographical location, but it can also represent its vehicle type. In both cases a depot-compatibility vector is given for every trip that shows the types of vehicles able to execute the trip.

Complexity of the vehicle scheduling problem depends on the number of vehicle depots. If all vehicles belong to the same depot, the problem is a single depot vehicle scheduling problems (SDVSP). An SDVSP can be modeled as a minimum-cost flow problem, thus even large-size (several thousand trips) instances are polynomially solvable to an optimum. Such a formulation is given in [5].

However, real life instances usually have 2 or more depots. These types of problems are called multiple depot vehicle scheduling problems (MDVSP). The MDVSP was first formulated by Bodin et al. [6], and its NP completeness was proven by Bertossi et al. [3].

Solving the MDVSP results in a set of vehicle schedules, each assigned to a vehicle from one of the depots. For every pair of trips i and j on any schedule, they have to be compatible, and their depot-compatibility has to match the depot of the vehicle assigned to the schedule. However, the solution itself can be constructed with regards to many different costs and constraints.

2.3 Models and Methods for the MDVSP

Literature discusses three main types of models for the MDVSP: single commodity, set partitioning, and multi-commodity. An overview of the different approaches can be found in [8]. In this paper, we will be dealing with the multi-commodity approach, which uses multi-commodity network flow models. In these models, every depot has its own commodity layer in the network, and only those trips are considered at a given layer, which can be executed from the corresponding depot.

The *connection-based network* gives every possible connection between the trips of the problem. To define the problem, the following additional notations have to be introduced: The set of depots that can execute a trip t is denoted by $g(t)$. Let $T_d \subseteq T$ be the set of trips that can be executed from depot d. Similar to the trips, let every d depot have a starting location $sl(d)$ and ending location $el(d)$. The set of nodes of our network will be the following:

$$N = \{dt(t) \cup at(t) \cup sl(d) \cup el(d) | t \in T, d \in D\}.$$

Let

$$A^d = \{(dt(t), at(t)) | t \in T_d\}$$

be the set of trips that can be served by depot d, and let

$$B^d = \{(at(t), dt(t')) | t, t' \in T_d \text{ are compatible}\}$$

be the possible deadhead trips of depot d.
Let

$$P^d = \{(sl(d), dt(t)), (at(t), el(d)) | t \in T_d\}$$

be all the pull-in and pull-out edges of depot d.
The above sets give us the set of edges of the connection based network:

$$E = A^d \cup B^d \cup P^d \cup \{(el(d), sl(d))\} \text{ for every } d \in D.$$

Based on the sets introduced above, a solution of the MDVSP can be determined by using the network (N, E). We define an integer vector x for every edge of the network. A vector component belonging to an edge e of depot d is denoted by x_e^d. The problem can be formalized in the following way:

$$\sum_{d \in g(t)} x^d_{dt(t),at(t)} = 1, \forall t \in T \tag{1}$$

$$\sum_{e \in n^+} x_e^d - \sum_{e \in n^-} x_e^d = 0, \forall d \in D, \forall n \in N \tag{2}$$

$$x_e^d \in 0, 1, \text{ except for } (el(d), sl(d)) \tag{3}$$

where n^+ is the set of outgoing, and n^- is the set of incoming edges from node n.

According to constraint (1) each trip has to be executed exactly once, while (2) means that every vehicle arriving to a geographical location has to leave that location. Constraints for the number of vehicles in each depot can be set as an upper bound for the circulation edges of the corresponding depot. Any flow satisfying the above conditions can give a feasible solution of the problem. For an optimal solution, we have to minimize

$$\sum_e c_e x_e,$$

where c_e is the cost of edge e.

The main drawback of the connection based model comes from its size. The number of compatible trips is high, even with a small number of trips in the problem, and this results in a large number of possible deadhead trips. This makes the size of the problem so large that it can not be used effectively on real-life data, where the number of trips is a couple of thousands usually.

The time-space network has been introduced to vehicle scheduling by Kliewer et al. [13]. It eliminates the drawback that comes from the size of the connection-based network. This way, it is possible to solve larger-sized real-time MDVSP instances efficiently. As we described earlier, the number of edges connecting compatible trips in the connection based model is high, but only a few of these are actually used in a feasible solution. However, if we left any of these connections from the model, we would lose the optimality of the solution.

The time-space network efficiently reduces the number of edges. The model arranges data in two dimensions: time and space. Space represents the set of geographical locations, while the timelines at each location represent a sequence of events. The arrival and departure times of the tasks are denoted on the corresponding timelines, and give the nodes of the model.

The time-space network is constructed using the above given nodes. Apart from this difference, the set N of nodes of the network can be defined similar to the connection based approach. The definition of A^d is also similar for each depot $d \in D$

and P^d is given with the help of the time-lines associated with the corresponding depot.

The definition of deadhead trips is the other main difference between the two models. The timelines used by the time-space network can be used to aggregate deadhead trips by introducing so-called waiting edges. These edges connect adjacent nodes on the timeline. This method reduces the size of the problem significantly. Waiting edges always connect two adjacent nodes on the appropriate timeline. Denoting the set of waiting edges with W^d for every depot $d \in D$, the set of edges of the time-space network is:

$$E = A^d \cup B^d \cup P^d \cup W^d \cup \{(el(d), sl(d))\} \text{ for every } d \in D.$$

Using these, the IP model of the time-space network can be given in a similar way to the connection based network:

$$\sum_{d \in g(t)} x^d_{dt(t),at(t)} = 1, \forall t \in T \tag{4}$$

$$\sum_{e \in n^+} x^d_e - \sum_{e \in n^-} x^d_e = 0, \forall d \in D, \forall n \in N \tag{5}$$

$$x^d_e \geq 0, \tag{6}$$

$$x^d_e \text{ integer} \tag{7}$$

Considerably bigger instances can be solved to optimality using the time-space network. However, the running time can still be an issue, especially in the case of larger real-life instances. Literature provides a variety of methods for solving large MDVSP. Selected heuristics based on both mathematical programming and combinatorial aspects can be found in [14]. However, these solve the standalone vehicle scheduling problem, and do not consider the "application oriented" structure of the vehicle schedules discussed above. We will present a solution method that takes this aspect into consideration also.

In [10], we present a collection of heuristic methods for the MDVSP found in literature, and also analyze the "application oriented" usefulness of their results. Paper [10] partially studied the idea of variable fixing too, which research later became the basic idea for this paper.

3 Reducing the MDVSP model size

In this section, we will give a heuristic to solve the MDVSP, taking into consideration both operational costs, and the "application oriented" structure of the vehicle schedules. A solution with short running time and well structured schedules is important for an interactive decision support system.

Gintner et al. [12] propose a two-phase heuristic to solve large instances of the MDVSP by decreasing its model size. A number of variables of the model are fixed,

and the resulting new problem is solved to optimality afterwards. Because of these two steps, they call this approach fix-and-optimize. The idea behind their heuristic is to solve a number of simplified models of the original problem, and decide on the variables to be fixed based on their results. This is done by finding series of trips that are common in all solutions. If such trips are found, it is presumed that they are likely to appear in the global optimum in the same way. These trips are called stable chains, and are used as single trips in the model of the MDVSP.

Their method decomposes the original MDVSP into an SDVSP for every depot. This problem is constructed and solved in the following way:

- The capacity of the depot is equal to the sum of all depot-capacities of the MDVSP.
- Only those trips are considered, which can be executed from depot.

After the SDVSP sub-problems are solved for each depot, their solutions are used to create stable chains. If a sequence of trips appears in the same order in all solutions, then they are considered as a chain. Using these stable chains as single trips, a smaller MDVSP model is built that has the following properties:

- The number and capacity of the depots are the same as in the original problem.
- The set of trips of the new problem consists of the trips that are not included in any of the stable chains, and a newly created trip for each stable chain. Their costs are the sum of all the trips these chains represent. The departure time and starting location of the first trip of the chain, and the arrival time and ending location of the last trip of the chain are used for this new trip as starting and ending data. These trips can be executed from any depot.

After this new MDVSP is solved, the trips in the stable chains have to be substituted back instead of the new trips, to acquire the final solution.

We chose to develop a heuristic based on the idea of variable fixing, as it can model the "application oriented" aspect of the problem. This is done by fixing trips in the same chain "that should belong together in the final solution". We can also control the amount of time between two consecutive trips of a chain by not adding a possible trip to a chain if that would leave too little, or too much gap in between.

As the basis of the method, we solved a simplified model of the problem that we call a "quasi-multiple depot" model. Though we use only a single depot in this model, two trips are connected only if they would be connected in the multiple depot case as well. This means that the trips have to be compatible, and they must also share a common depot from which they can be served. The cost of the arc between these two trips in our model is calculated using the cheapest possible cost of all their common depots. The capacity of the depot is the sum of the capacities of all depots in our original problem. Pull-out and pull-in arcs of the depot have the weight of the minimal deadhead trip from and to any of the depot locations of

the original problem. Once this "quasi-multiple depot" model is constructed, it is solved by an MILP solver.

We experimented with three different approaches for finding stable chains in the solution of the above problem:

- Build chains with regards to depot costs.
- Fix trips with the same depot-compatibility in a chain.
- Assign trips of the same bus-line to a chain.

The methods will be presented in the following subsections. We also illustrate on a small example their difference in building chains.

3.1 Building chains using depot costs

This was the first heuristic we developed for solving the MDVSP. Depots are ordered into a list increasingly according to the following cost:

$$\frac{1}{\epsilon} * cost(daily) + cost(km)$$

where $cost(daily)$ is the daily cost of a single vehicle from the depot, $cost(km)$ is the cost of that vehicle to travel 1 km, and $\epsilon > 0$ is a parameter.

For every depot in this order, the algorithm examines all the vehicle schedules in the result of the "quasi-MDVSP". If subsequent trips are found which can be executed from this depot, they are considered together in stable chains. These trips are flagged, and cannot be the part of other stable chains. The description of this algorithm can be seen in Algorithm 1.

This algorithm is only the basis of finding the chains, further constraints can be introduced:

- We can give a limit to the number of trips in the chains.
- The length of the chains can be maximized.
- The minimum/maximum gap in time between two trips of the chain can be given.

Experience shows, that limiting the length of the stable chains with the above constraints results in a solution with better cost, but has an increase in running time. The running time of the heuristic was very fast, but the quality of the solutions was far from what we have expected. Because of this further changes have been experimented with to improve the cost of the solution, with a minimal increase in the running time.

Algorithm 1 Variable fixing using depot costs.

```
1: Determine the order D of depots
2: S ← ∅, V ← ∅, L ← ∅
3: for each d ∈ D do
4:   for all trips j ∉ V the solution do
5:     while j can be executed from d do
6:       L ← j
7:       V ← j
8:       j = nexttrip(j)
9:     end while
10:    if |L| > 1 then
11:      S ← L
12:    end if
13:    L ← ∅
14:  end for
15: end for
16: return S
```

3.2 Building chains based on depot-compatibility

Using our experience gained from the method presented above, we tried to find a way to fix trips that have some property in common instead of using a cost function. In our second method, we tried to construct stable chains based on similar depot-compatibilities of the trips. We first examined those trips from the solution of the "quasi-MDVSP" that can be executed from all d depots. Then all that are compatible with $d-1$ depots, and so on. Two subsequent trips are assigned to the same chain if they have exactly the same depot-compatibility in the solution. This algorithm is described in Algorithm 2. The same additional extra constraints that we have shown at Algorithm 1 can also be introduced here.

3.3 Building chains using trips of the same bus-line

This method of constructing the stable chains is closer to the schedule building practice of transportation companies. A driver usually uses the same vehicle during his shift, and he is carrying out consequent trips of the same bus-line. Some changes might occur in his schedule, but their number remains low. However, when solving the MDVSP using an MILP solver, the resulting vehicle schedules usually have a high number of line changes. Though this can not be modeled in costs directly, it puts some load on the driver itself.

We tried to build stable chains using this as a guideline. We fixed those trips in a chain only that belong to the same bus-line. We also set a maximum time limit of the gap between two such trips. If two subsequent trips belong to the same bus-line, but are far from each other in time, then they are not fixed in the same chain. The description of this method can be seen in Algorithm 3.

Algorithm 2 Variable fixing based on depot compatibilities.

$S \leftarrow \emptyset, V \leftarrow \emptyset, L \leftarrow \emptyset$
for $d = numof(depots)$ **downto** 1 **do**
  for all $j \notin V$ trips compatible with exactly d depots **do**
    $L \leftarrow j$
    $V \leftarrow j$
    $k = nextrip(j)$
    while j and k have the same depot-compatibility **do**
      $L \leftarrow k$
      $V \leftarrow k$
      $j = k$
      $k = nexttrip(j)$
    end while
    if $|L| > 1$ **then**
      $S \leftarrow L$
    end if
    $L \leftarrow \emptyset$
  end for
end for
return S

Algorithm 3 Variable fixing based on same bus-lines.

$S \leftarrow \emptyset, V \leftarrow \emptyset, L \leftarrow \emptyset$
for all $j \notin V$ trips **do**
  $L \leftarrow j$
  $V \leftarrow j$
  $k = nextrip(j)$
  while $line(j) = line(k)$ and $dt(k) - at(j) < limit$ **do**
    $L \leftarrow k$
    $V \leftarrow k$
    $j = k$
    $k = nexttrip(j)$
  end while
  if $|L| > 1$ **then**
    $S \leftarrow L$
  end if
  $L \leftarrow \emptyset$
end for
return S

3.4 An illustrative example of building chains

In this subsection we show the differences between the three variable fixing methods presented above. Let us consider a vehicle scheduling problem with 3 depots, 8 trips and 4 geographical locations (A,B,C,D).

Table 1: Details of the trips.

Trip	From	To	Departure	Arrival	Depots
1	C	B	10	12	1,2,3
2	B	A	12	14	1,2,3
3	B	C	12	14	2,3
4	A	B	14	16	1,2,3
5	B	A	16	18	1,2
6	A	B	18	20	1,2
7	C	D	22	24	2,3
8	D	C	24	26	2

Table 1 gives every detail of the trips, including their start and end geographical locations, their departure and arrival times and the depots that they are compatible with. Furthermore, suppose that trips between the same geographical locations belong to the same bus-line. This gives us the following three lines:

- Line A-B: trips 2,4,5,6.
- Line B-C: trips 1,3.
- Line C-D: trips 7,8.

Let the deadhead distance between any pair of geographical locations, and the pull-in and pull-out distance for all depots be 2 minutes. The depot costs of the problem are the following:

- Depot 1: 100 daily cost and 10/minute distance cost
- Depot 2: 200 daily cost and 20/minute distance cost
- Depot 3: 300 daily cost and 30/minute distance cost

The structure of the above problem can be seen on Figure 1. The horizontal lines represent the geographical locations, while the arrows between them correspond to the trips. All three heuristics solve a quasi-multiple depot problem, which results in the following two vehicle schedules:

- Schedule 1 executes trips 1,2,4,5,6.
- Schedule 2 executes trips 3,7,8.

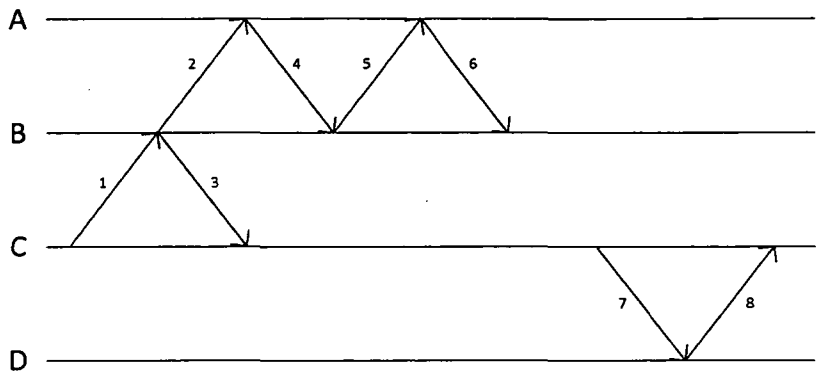


Figure 1: The structure of the problem

The heuristics will try to construct chains based on these schedules. Applying the *method based on depot costs*, the cost function will give the depot order 1,2,3 for an arbitrary $\epsilon > 0$. Using this order, the following chains are constructed:

- Chain 1: trips 1,2,4,5,6.
- Chain 2: trips 3,7,8.

If we build the chains with regards to *depot-compatibility*, first we examine trips that are compatible with all 3 depots, then the trips compatible with 2 depots, and finally the trips that are compatible with 1 depot only. This results in the following chains:

- Chain 1: trips 1,2,4.
- Chain 2: trips 5,6.
- Chain 3: trips 3,7.
- Chain 4: trip 8.

Considering *bus-lines* when building the chains, we have to examine all 3 bus-lines in the schedules. The method constructs the following chains:

- Chain 1: trips 2,4,5,6.
- Chain 2: trip 1.
- Chain 3: trip 3.
- Chain 4: trips 7,8.

4 Random instances

Though we were provided with the real-life instances of the transportation company of Szeged, that gives us only a limited database. As it is difficult to access real-life instances from other companies, the easiest way to acquire more test data that has the properties of the real-life input is to use an algorithm that generates it based on our needs. Many papers from literature that deal with the vehicle scheduling problem present the efficiency of their methods on random instances generated according to an algorithm by Carpaneto et al. [9]. However, our test experience shows that the structure of the data generated by their method was different from real-life instances in many aspects. Because of this, we propose an improved way of generating random data in this section. First, we describe the above method, and then give a new variation for it.

4.1 Generation method by Carpaneto et al.

The input of the algorithm is the n number of trips, and the m number of depots. The number of geographical locations is uniformly chosen from the interval $[\frac{n}{3}, \frac{n}{2}]$, their locations are chosen in a uniform random way on a $60*60$ grid. The deadhead trips between geographical locations p and q correspond to their $d(p,q)$ Euclidean distance.

The properties of every t_i trips is determined using the above. The start and end $sl(t)$ and $el(t)$ geographical locations are chosen uniformly from $[1, f]$. These locations determine the length of the trip, $d(sl(t), el(t))$. Trips can have two types: short trip, or long trip.

There is a 40% chance that a t trip becomes a short trip. Its $dt(t)$ departure time is also chosen randomly:

- with a 15% chance uniformly from [420,480]
- with a 70% chance uniformly from [480,1020]
- with a 15% chance uniformly from [1020,1080]

The $at(t)$ arrival time of a short trip is chosen uniformly from the interval $[dt(t) + d(sl(t), el(t)) + 5, dt(t) + d(sl(t), el(t)) + 40]$.

Long trips are generated with a 60% chance. Their $dt(t)$ departure time is chosen uniformly from $[300, 1200]$, while their $at(t)$ arrival time is chosen uniformly from $[dt(t) + 180, dt(t) + 300]$. Long trips have the same start and end location, which means that a value is assigned to $sl(t) = el(t)$ uniformly from $[1, f]$.

They also presented a possible placement of the depots for $m = 2, 3$. The number of vehicles in each depot is determined uniformly from $[3 + \frac{n}{3m}, 3 + \frac{n}{2m}]$.

4.2 Our new generation method

Our experience showed that the instances generated using the above method were very differently structured from the real-life data we were dealing with. We decided

to modify this method to become closer to those real-life instances. The main difference of this model from our data was that trips had no pre-assigned depot-compatibilities. For this, we introduced an additional input: a p_i probability for every $1 \leq i \leq m$ depot.

The p_i value gives the probability that a trip can be executed from depot i. When the trips are generated, they are assigned a $\mathbf{v} = (v_1, ..., v_m)$ depot-compatibility vector. For every v_i,

$$v_i = \begin{cases} \text{true} & \text{with } p_i \text{ probability} \\ \text{false} & \text{otherwise} \end{cases}$$

If all components of the $\mathbf{v}$ recieve false values, then exactly one of them is set as true. This is also decided using the given probabilities. A trip can be executed only from those depots, whose corresponding components have a true value.

Analyzing the trips of the original generator, we found that the average length of the trips was too high compared to our real-life data, and the trips were scattered geographically. To address this, we introduced some further changes.

The number of generated geographical locations was also very high compared to the number of trips, and two trips rarely followed each other at the same location in a small timeframe. After experimenting, we found the $[\frac{2n}{25}, \frac{3n}{25}]$ interval that gives an acceptable number of locations. However, because of the decreased number of geographical locations, we also had to decrease the area they are generated at. We used a $30 * 30$ grid for this.

To address the problem of the too long average length of the trips, we slightly modified the generation of the trips as well. The ratio of the long and short trips has been exchanged, and we generated short trips with a 60% chance, and long trips with only a 40% chance.

The length of the trips has been decreased. The $at(t)$ arrival time of a short trip is chosen uniformly from the interval $[dt(t)+d(sl(t), el(t)), dt(t)+d(sl(t), el(t))+20]$, while the $at(t)$ arrival time of a long trip is chosen uniformly from $[dt(t)+40, dt(t)+60]$.

Using the modification above, the random generated instances we received resembled more closely to the real-life data we were provided with by the transportation company of Szeged city.

5 Test Results

The different variable fixing approaches discussed earlier were tested on real-life data instances from the city of Szeged, Hungary, as well as on random data generated by an algorithm described in the previous section. We present their results below.

5.1 Real-life instances

Our real-life instances were taken from the transportation company of the city of Szeged, Hungary. The company uses 11 different day-types (called combinations) over its planning period. To get a complete schedule for a normal planning period (which is 2 months in the case of the company), the vehicle scheduling problem has to be solved for all 11 combinations. The daily driver schedules will depend on the combination type of the corresponding day. Of all the combinations 4 have been selected as test cases. The properties of their optimal solutions can be seen in Table 2. The combinations with higher number of trips (*szeged1* and *szeged4*) are workdays of the week, while *szeged2* and *szeged3* are instances taken from a Sunday and Saturday respectively.

The running time given in the table is the running time in seconds needed to find an optimal solution using the SYMPHONY solver on the time-space network model of the problem.

Table 2: Optimal solution of the instances.

Instance	Day type	Running time(s)	Vehicles	Dense schedules
szeged1	Weekday	872	96	4
szeged2	Sunday	431	44	3
szeged3	Saturday	250	55	6
szeged4	Weekday	1179	96	3

As it is visible, the running times of the weekday instances can reach 20 minutes, and solving all the 11 combinations of the company to optimality would take about 8500 seconds. The 2-2,5 hours of running time for calculating the vehicle schedules is not acceptable from the perspective of a decision support system, as there are the additional driver schedules and rosters that still have to be calculated for the whole planning period (which is usually several weeks or months).

We will examine three aspects of the results given by the heuristics:

- The gap in cost of the result from the optimal solution of the MDVSP.
- The ratio of the running time of the heuristic compared to the running time of the IP solver.
- The structure of the schedules.

When vehicle schedules are used as an input in driver scheduling, different driver constraints have to be fulfilled. The most important of these are the maximum consecutive driving time without any rest, and the total length of the schedule given to a driver. We analyze the structure of the vehicle schedules using these aspects. If a schedule violates any of the mentioned constraints, it is labelled as a "dense" schedule.

The results of the variable fixing heuristic of Gintner et al. can be seen in Table 3. Every solution shows a decrease in running time: the average running time of the instances is about 40% of the original, which would mean a running time of 3000-3500 seconds (almost 1 hour) for all the vehicle schedules. The gap from the optimum varies between 0,25%-0,40%.

Table 3: Solution of the variable fixing heuristic of Gintner et al..

Instance	Gap from opt.	Running time ratio	Vehicles	Dense schedules
szeged1	0,27%	57,45%	96	6
szeged2	0,41%	31,55%	44	3
szeged3	0,37%	42,80%	55	6
szeged4	0,25%	35,69%	96	4

The heuristic that uses a depot-cost function for building the chains decreases the running time to around 8,8% of the time needed for the IP solver. This means that a solution is obtained in a couple of minutes (which is at most 4-5 in all test cases). The total running time for all 11 combinations is between 10-15 minutes, which is really good. However, the gap from the optimal solution has risen significantly: in some cases, it was greater than 2,5%. As opposed to the variable fixing heuristic, the greedy method fixes significantly more trips ($\sim$ 66% in comparison with $\sim$ 33%) into stable chains, which greatly reduce the size of the problem. However, the method is less precies because of the fact that more trips are fixed in chains. Limiting the chain construction with the before mentioned alternative constraints (e.g. limit the size/length of the chains, or the types of chosen trips) will lead to a solution with a better cost. On the other hand, less fixed trips also mean a greater problem size, which results in an increase in running time. The results of this method can be seen in Table 4.

Table 4: Solution using chains based on depot costs.

Instance	Gap from opt.	Running time ratio	Vehicles	Dense schedules
szeged1	2,63%	9,29%	100	3
szeged2	1,20%	3,71%	46	5
szeged3	1,12%	10,40%	57	7
szeged4	2,32%	8,40%	98	3

Building the chains based on depot-compatibility shows a more ordered structure than the method discussed above. Though more trips remain single, which comes with a slight increase in running time (in average 11,63% of the IP solution, which is about 15-20 minutes for all the combinations), it is still acceptable. The gap became also significantly smaller, it is at most around 1,25%. This value can

be acceptable, though still seems a bit high. The cost can be improved in the same way as in the previous case, but this will also result in an increase in running time. However, the rate of decrease in the cost will be much smaller with the inclusion of additional constraints. The results of this heuristic can be seen in Table 5.

Table 5: Solution based on depot-compatibility.

Instance	Gap from opt.	Running time ratio	Vehicles	Dense schedules
szeged1	1,14%	12,84%	97	2
szeged2	0,34%	6,50%	44	4
szeged3	0,38%	16,00%	56	8
szeged4	1,26%	19,42%	97	2

Taking into consideration the experience of the previous solution methods and analyzing their difference from the solution of the IP solution, we decided to apply a more structural method for building the chains. Using trips of the same bus-line in a chain again leads to less fixed trips, which means an overall decrease in running time to 24,41% of the original. This results in about 30-35 minutes to solve all the combinations. On the other hand, the gap of the solutions from the optimum is very favourable, not more than 0,60% in any of the instances, but there are much lower ones around 0,20%, or even below. The results of this method are found in Table 6.

Table 6: Solution using chains based on bus-lines.

Instance	Gap from opt.	Running time ratio	Vehicles	Dense schedules
szeged1	0,58%	16,28%	97	3
szeged2	0,03%	21,35%	44	5
szeged3	0,23%	26,00%	55	7
szeged4	0,59%	10,86%	97	3

The solution methods give about the same number of badly structured vehicle schedules as the original IP solutions did. Using additional constraints that limit the length of the chains, the number of the trips in the chains, or the minimal/maximal idle time between two subsequent trips in a chain result in a lower number of "dense" schedules besides the smaller gap from the optimum. However, this would still affect the running time of the methods.

With the use of these "dense" schedules we tried to give a formal way of evaluating the "goodness" of the vehicle schedule structure with regards to driver scheduling. While the number of badly structured schedules stays approximately the same using any of the shown heuristics, the impact it has on a driver scheduling algorithm can be really different depending on which vehicle heuristic is used. We

experimented with several sequential vehicle and driver methods for the problem, among which our most recent research can be found in [2]. Test results show that using any of our proposed variable fixing methods gives a better cost at the end of the driver scheduling phase than using either the optimal MDVSP solution, or the heuristic of Gintner et al.

5.2 Solutions on random data input

As we mentioned earlier, we also tested the algorithms on random data instances. We tried different problem sizes with 50, 250, 500 and 1000 trips respectively. Out of the 4 methods above, the heuristic of Gintner et al. failed to find any chains in all cases, while our heuristic using bus-lines rarely fixed any trips, and it always fixed only less than 5 trips using these inputs. This means that both methods ended up solving the original (or almost exactly the original) MDVSP, and thus their results can not be analyzed properly.

The heuristic of Gintner et al. needs a large number of trips in the input that can be executed from any of the depots. Besides this, these trips have to be close enough to one another so that every solved SDVSP sub-problem schedules them in the same sequence. If the trips that are compatible with every depot are scattered on the timeline of the problem, then none or only some of the trips will be fixed in chains. This scenario is likely to happen in the proposed random instances, which explains the failure of the heuristic in finding chains.

The method based on bus-lines has the same problem on this randomly generated input. Real-life instances have different bus-lines, which roughly mean that there are given p and q geographical locations, between which trips occur back and forth with a given frequency. Random generated instances will not have this kind of order in their timetable, thus this heuristic is likely to fail too.

The results of the other two heuristics can be seen in Table 7. The column marked with (cost) represent the heuristic that uses a cost function, while the column marked with (depot) give results for the heuristic based on depot-compatibility. The heuristic using a cost function arrived at about the same results, as on the real-life instances, while the heuristic based on depot-compatibility also fixed fewer trips than usual. As this method also depends on trips sharing the same depot, it also has a more difficult time finding chains.

Table 7: Solution on random instances.

Instance	Gap (cost)	Gap (depot)
random_50	0%	0%
random_100	0%	0%
random_500	1,57%	$9 * 10^{-6}$%
random_1000	1,54%	0,02%

Test experience on the random instances show that the data generated by our

method is still different from real-life instances in some structural aspects. Heuristics that are based on structural properties appearing in real-life data (Gintner et. al, bus-lines) can not be applied effectively to most of the generated input. This means that we need a method that models the properties of real-life instances more closely. The two main parts that have to be improved in the random generator are the possibility to create entire bus-lines randomly, and to model the depot-compatibility distribution of trips in a better way.

6 Conclusions and Future Work

We examined the vehicle scheduling problem and its existing models and methods in literature, to find one that fits the framework of an interactive decision support system. Such a method is important, as transportation companies do not only need to have an efficient long term planning software, but they must also have a way to assist important decisions and give suggestions in a reasonable time. Running time was an important criterion for such a method, but the value of the solution had to stay close to the optimum as well.

We developed several solution approaches based on the core idea of an efficient heuristic by Gintner et al. Each of our solution approaches became more refined as the previous one, as their results were analyzed and taken into consideration at every step. Our final heuristic comes with both an acceptable running time, and a small gap from the optimal solution. Moreover, our test experience shows that they also work well in a sequential decision support system.

We also examined the commonly used random generator method of Carpaneto et al. However, our experience showed that the distribution of trips generated by their algorithm was very different from real-time instances. We proposed an improved version of this algorithm so that its output has a structure that is closer to real-life. Extensive testing on these instances showed that some of our presented methods also work nicely on random data as well, and produce an acceptable solution for it.

Test results show that the heuristic methods can be slightly improved. In order to do this, more analysis has to be carried out into the structure of the solutions and their differences from the schedules of the original IP solution. Further experiments can also be made with the different parameters and limiting constraints discussed in the paper.

The metric of "dense" schedules that we introduced to measure the effect of our algorithms on future driver schedules turned out to be poorly defined. A different analysis has to be carried out into the interaction of vehicle and driver schedule. We have to give another metric by identifying the exact types of schedules that are expensive to transform in the driver phase.

The random data generating algorithm also has to be improved further. As we have seen at our test cases, the structure of results on random instances differs from real-life cases in important elements. The most important of these is the inclusion of bus-lines, which are crucial to methods that take this structural property into consideration. This requires additional study of the original timetables of our

real-life cases, and tuning the parameters that affect the number and position of geographical locations, and the frequencies of the trips.

Acknowledgments.

This work was partially supported by the Szeged City Bus Company (Tisza Volán, Urban Transport Division) and Gyula Juhász Faculty of Education, University of Szeged (project no. CS-004/2012).

The second author was partially supported by the European Union and co-funded by the European Social Fund through project HPC (grant no.: TÁMOP-4.2.2.C-11/1/KONV-2012-0010).

References

[1] Ahuja, R.K., Magnanti, T.L., Orlin, J.B. Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, 1993.

[2] Árgilán, V., Balogh, J., Békési, J., Dávid, B., Krész, M., Tóth, A. Driver scheduling based on driver-friendly vehicle schedules. in *Proceedings of OR 2011 International Conference on Operations Research*, pages 323-328, Springer-Verlag, 2011.

[3] Bertossi, A.A., Carraresi, P., Gallo, G. On Some Matching Problems Arising in Vehicle Scheduling Models. *Networks* 17, pages 271–281, 1987.

[4] Békési, J., Brodnik, A., Pash, D., Krész, M. An integrated framework for bus logistic management: case studies. in *Logistik Management*, pages 389-411, Physica-Verlag, 2009.

[5] Bodin, L., Golden, B. Classification in vehicle routing and scheduling. *Networks* 11, pages 97–108, 1981.

[6] Bodin, L., Golden, B., Assad, A., Ball, M. Routing and Scheduling of Vehicles and Crews: The State of the Art. *Computers and Operations Research* 10, pages 63–212, 1983.

[7] Borndörfer, R., Löbel, A., Weider, S. A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. *Computer-aided Systems in Public Transport*, pages 3-24, 2008.

[8] Bunte, S., Kliewer, N. An overview on vehicle scheduling models. *Journal of Public Transport* 1(4), pages 299-317, 2009.

[9] Carpaneto, G., Dell'Amico, M., Fischetti, M., Toth, P. A branch and bound algorithm for the multiple depot vehicle sheduling problem. *Networks* 19, pages 531-548, 1989.

[10] Dávid, B. Heuristics for the Multiple-Depot Vehicle Scheduling Problem. in *Proceedings of the 2010 Mini-Conference on Applied Theoretical Computer Science*, pages 23-28, 2011.

[11] Gintner, V., Kliewer, N., and Suhl, L. A Crew Scheduling Approach for Public Transit Enhanced with Aspects from Vehicle Scheduling. *Computer-aided Systems in Public Transport*, pages 25-42, 2008.

[12] Gintner, V., Kliewer, N., and Suhl, L Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice. *OR Spectrum* 27, pages 507-523, 2005.

[13] Kliewer, N., Mellouli, T., Suhl, L. A time-space network based exact optimization model for multi-depot bus shceduling. *European Journal of Operational Research* 175, pages 1616-1627, 2006.

[14] Pepin, A.-S., Desaulniers, G., Hertz A., Huisman, D. Comparison of Heuristic Approaches for the Multiple Depot Vehicle Scheduling Problem. *Journal of Scheduling* 12, pages 17-30, 2009.

Acta Cybernetica **21** (2013) 75–88.

Online Clustering on the Line with Square Cost Variable Sized Clusters

Gabriella Divéki*

Abstract

In the online clustering problems, the classification of points into sets (called clusters) is done in an online fashion. Points arrive one by one at arbitrary locations, to be assigned to clusters at the time of arrival without any information about the further points. A point can be assigned to an existing cluster, or a new cluster can be opened for it. Existing clusters cannot be merged or split. We study one-dimensional variants. The cost of a cluster is the sum of a fixed setup cost scaled to 1 and the square of the diameter of the cluster. The goal is to minimize the sum of costs of the clusters used by the algorithm. In this paper we investigate the problem on the line.

We examine two versions, both maintaining the properties that a point which was assigned to a given cluster must remain assigned to this cluster, and clusters cannot be merged. In the strict variant, the size and the exact location of the cluster must be fixed when it is initialized. In the flexible variant, the algorithm can shift the cluster or expand it, as long as it contains all points assigned to it. We consider the online and the semi-online (the input is sorted according to their coordinates from smallest to largest i.e., from left to right) versions of the above two variants.

We present the first online algorithms for the solution of the problem. We describe algorithms for the strict and the flexible variant both for the online and semi-online versions. We also give lower bounds on the possible competitive ratio in all of the cases.

Keywords: online algorithms, competitive analysis, clustering problems

1 Introduction

In clustering problems, we seek for a partitioning of n demand points into k groups, or clusters, while a given objective function, that depends on the distance between points in the same cluster, is minimized. In the online version, the demand points

*Subotica Tech - College of Applied Sciences, Marka Oreškovića 16, 24000 Subotica, Serbia, E-mail: diveki.gabriella@gmail.com

are presented to the clustering algorithm one by one. The online clustering algorithm maintains a set of clusters, where a cluster is identified by its name and the set of points already assigned to it. Each point must be assigned to a cluster at the time of arrival; the chosen cluster becomes fixed at this time. The clusters cannot be merged or split.

Usually, the quality of an online algorithm is measured by competitive analysis. An online algorithm for a minimization problem is C-competitive if the algorithm cost is never more than C times the optimal offline cost. (For a good introduction to competitive analysis, see [3, 11, 17].) In the case of clustering problems, the costs are based on the number of clusters and their properties, and they depend on the exact specification of the problem.

In this paper we consider the 1-dimensional variant of the 2-dimensional online clustering with variable sized clusters problem which is presented in [22]. In our model points of the 1-dimensional Euclidean space arrive one by one. After the arrival of a point we have to assign it to an existing cluster or to define a new cluster for it without any information about the further points. The clusters are intervals, the cost of each cluster is the sum of the constant setup cost scaled to 1 and the square of the length of the interval. The goal is to minimize the total cost of the clusters.

We consider two variants, both having property that a point assigned to a given cluster must remain in this cluster, and clusters cannot be merged. In the strict variant, the size and the location of the cluster must be fixed when it is initialized. In the flexible variant, the algorithm can shift the cluster or expand it, as long as it contains all the points assigned to it.

In [6] the one-dimensional variant of our problem is examined (with linear cost), where there is no restriction on the length of a cluster, and the cost of a cluster is the sum of a fixed setup cost and its diameter. Both the strict and the flexible model have been investigated and an intermediate model, where the diameter is fixed in advance but the exact location can be modified is also studied. In [6], tight bounds are given on the competitive ratio of any online algorithm belonging to any of these variants. Tight bounds are given of $1+\sqrt{2} \approx 2.414$ on the competitive ratio for the online problem in the strict model, and tight bounds of 2 in the semi-online version. In the intermediate model, the results of the previous model were extended and it is shown that the same bounds are tight for it as well. Using the flexible model, the best competitive ratio dropped to $\Phi = \frac{1+\sqrt{5}}{2} \approx 1.618$. The semi-online version of this model is solved optimally using a trivial algorithm which is discussed as well in [6].

Several results are known on online clustering with fixed unit sized clusters. A study of online partitioning of points into clusters was presented by Charikar et al. [5]. The problem is called online unit covering. A set of n points needs to be covered by balls of unit radius, and the goal is to minimize the number of balls used. The authors designed an algorithm with competitive ratio $O(2^d d \log d)$ and gave a lower bound of $\Omega(\log d / \log\log\log d)$ on the competitive ratio of deterministic online algorithms in d dimensions. This problem is strictly online: the points arrive one

by one, each point has to be assigned to a ball upon arrival, and if it is assigned to a new ball, the exact location of this ball is fixed at this time. The tight bounds on the competitive ratio for $d = 1$ and $d = 2$ are 2 and 4, respectively.

Chan and Zarrabi-Zadeh [4] introduced the unit clustering problem. Here the input and goals are identical to those of unit covering, but the model of online computation is different. This is an online problem as well, but it is more flexible in the sense that the online algorithm is not required to fix the exact position of each ball at the first time the ball is "used". The set of points which is assigned to a ball (cluster) can always be covered by that ball and the ball can be shifted if necessary. The goal is still to minimize the total number of balls used. Unit covering and unit clustering are the same problem when observing in an offline fashion, and the problem is solvable in polynomial time for $d = 1$. In the online model an algorithm for the unit clustering problem has more flexibility because of the optional shifting of a cluster. In [4], the authors showed that standard approaches lead to algorithms of competitive ratio 2 (some of which are valid for unit covering). The lower bound of 2 for unit covering in one dimension is valid even for randomized algorithms. A non-trivial randomized algorithm was presented: a $\frac{15}{8}$-competitive algorithm; also in [21] an $\frac{11}{6}$-competitive randomized algorithm. In [10] an improved deterministic algorithm was given (with competitive ratio $\frac{7}{4}$) and in [8] an algorithm of competitive ratio $\frac{5}{3}$. Currently the best known lower bound is $\frac{8}{5}$ (see [10]).

In [4, 8, 10, 21] the two-dimensional problem is considered using the ℓ_∞ norm instead of the ℓ_2 norm. Thus, "balls" are squares or cubes. The one-dimensional algorithms are used as building blocks in most results in the mentioned papers. This problem has a higher competitive ratio than the one-dimensional case (the best known lower bound is $\frac{13}{6}$ - see [8]). Other variants of the one-dimensional online unit clustering problem were studied in [9].

Our problem is also related to online facility location [7, 12, 13, 14, 15, 19], where the input is a sequence of points and the algorithm has to partition them into clusters and it has to assign a facility to each cluster. On the other hand in facility location the cost of the cluster differs: it is the sum a fixed setup cost and the service cost which is the total distance of the points from a facility assigned to the cluster. In some of the online facility location models it is allowed to merge clusters or to re-assign points.

Our results: We present the first online algorithms for the solution of the problem. We present algorithms for the strict and the flexible variant both for the online and semi-online versions. In this paper, when we refer to a semi-online algorithm we mean an online algorithm for restricted set of inputs in which the points arrive one by one when they are sorted according to their coordinates from smallest to largest (i.e., from left to right).We also give lower bounds on the possible competitive ratio in all of the cases.

We analyze algorithms and give their competitive ratio. We prove that the $GRID_a$ algorithm is 3-competitive in the strict model if we use appropriate size for the cells for the grid. We present the algorithm $SOSM_a$ for the semi-online strict model and prove that it has competitive ratio 2 if the size of the cells are

within appropriate bounds. We also give lower bounds in the strict model for both variants: 2.2208 for the online and 1.6481 for the semi-online variant.

We extend the algorithm to the flexible model and we prove that it is 2-competitive if we use appropriate size for the cells. In this model we show that no online algorithm can have smaller competitive ratio than 1.2993. Also, we give the lower bound for the semi-online flexible model: 1.1991144.

In the rest of the paper for an algorithm A and input I we use $A(I)$ to denote the cost of A on input I.

2 The offline problem

As far as we know the offline clustering with this objective function have not been studied yet. Many papers are published on the offline version where the number of clusters is a given constant k. Usually the cost is the sum of the diameters (see [16] and its references for details) but there are also some results on the models where it depends on the powers of the diameters (see [2]), and even for general cost functions (see [18]). All of these problems are NP-hard. If the number of clusters is not fixed and the cost depends on the diameters then the problem is polynomially solvable for trees see [20] and it has not been studied for more general metric spaces yet.

Lemma 1. *The offline problem can be solved optimally by the dynamic programming algorithm DP using the algorithm for the variation of the k-median problem.*

This is an interesting transition: our offline clustering problem on the line with linear objective function can be solved with a simple greedy algorithm with $O(n \cdot logn)$ time complexity (see [6]), the problem on the line with squared cost can be solved by a standard $O(n^3)$ time dynamic programming algorithm. On the other hand the 2-dimensional case seems to be much harder, we conjecture that it is NP-hard.

The input is n request points $(x_1, ..., x_n)$. The dynamic programming algorithm is shown in Algorithm 1.

Algorithm 1 Algorithm *DP*

- The request points are sorted by their coordinates in ascending order.
- Define the subproblem $F(i,r)$ $(i \geq r)$: the first i request points are divided into r clusters. Then the optimal cost of the clustering problem is $min_r(F(n,r)+r)$.
- The values of $F(i,r)$ can be calculated by the following recursions.

  $F(i,1) = (x_i - x_1)^2$

  $F(i,r) = min_{j=r}^{i}\{F(j-1,r-1) + (x_i - x_j)^2\}$

The dynamic programming algorithm correctly calculates because if the last cluster is $[x_j, x_i]$ then we have to assign the first $j-1$ request points into $r-1$ clusters optimally.

Based on these steps of the dynamic programming algorithm a 2-dimensional array can be filled sorted by the second dimension r in ascending order. Then one can get the optimal solution from this table. As the algorithm DP fills an $n \times n$ table and an element of the table can be computed in $O(n)$ steps, therefore the time complexity of algorithm DP is $O(n^3)$.

3 The strict model

3.1 The online problem

The $GRID$ algorithm which uses a grid in the 1-dimensional space is defined in [9] for the problem of unit covering with rejection. In [22] the $GRID_a$ algorithm was investigated in 2 dimensions for the strict model; we consider its special 1-dimensional case and also the analysis is similar.

Algorithm $GRID_a$ works as follows. Upon arrival of the first point in the interval $I_k = (ka, (k+1)a]$ for every integer $-\infty < k < \infty$, a new cluster is opened in the interval $[ka, (k+1)a]$ and all future points in this interval are assigned to this cluster. The competitive ratio of $GRID_a$ is determined by the following theorem.

Theorem 1. *The competitive ratio of algorithm $GRID_a$ is*

$$\max\{F(\lfloor -2+\sqrt{4+\frac{1}{a^2}}\rfloor), F(\lceil -2+\sqrt{4+\frac{1}{a^2}}\rceil), 2+2a^2\}$$

where $F(k) = \frac{(k+2)(1+a^2)}{1+k^2a^2}$, $k \geq 1$.

Proof. Consider an arbitrary sequence and an optimal solution for it, denoted by OPT. We investigate the clusters of OPT separately. Consider an arbitrary cluster. Let r denote the length of this cluster.

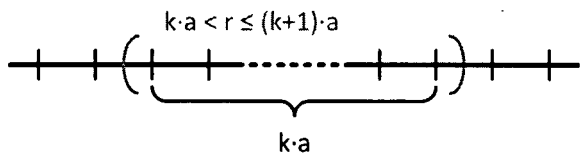


Figure 1: The optimal cluster intersects at most $k+2$ clusters from the grid

Suppose first that $k \cdot a < r \leq (k+1) \cdot a$ for an integer $k \geq 1$. Then this optimal cluster intersects at most $k+2$ clusters from the grid (see Figure 1). Therefore, if we consider only the requests of this optimal cluster then $GRID_a$ has at most

$(1+a^2)(k+2)$ cost. Thus the competitive ratio on this subsequence is at most $\frac{(1+a^2)(k+2)}{r^2+1} < \frac{(1+a^2)(k+2)}{k^2a^2+1} = F(k)$. The derivative of this function is

$$F'(k) = \frac{(1+a^2)\cdot(1-4ka^2-k^2a^2)}{(1+k^2a^2)^2}.$$

$F'(k)$ is 0 at $k^* = -2+\sqrt{4+\frac{1}{a^2}}$. The second derivative of $F(k)$ is

$$F''(k) = \frac{2\cdot(1+a^2)\cdot a^2\cdot(k^3a^2-3k+6k^2a^2-2)}{(k^2a^2+1)^3}$$

while $F''(k^*) < 0$ for every a. Therefore $F'(k)$ is positive before k^*, and it is negative after k^*. This yields that $F(k)$ has maximum at k^*. We have to consider the positive integers, so the maximum is attained either at $k = \lfloor -2+\sqrt{4+\frac{1}{a^2}} \rfloor$ or at $k = \lceil -2+\sqrt{4+\frac{1}{a^2}} \rceil$.

Now suppose that $r \le a$. Then the cluster intersects at most 2 clusters from the grid. Therefore, considering the requests of this cluster $GRID_a$ has at most $2\cdot(1+a^2)$ cost. Thus the competitive ratio on this subsequence is at most $(2+2a^2)/(1+r^2) \le 2+2a^2$.

Now we prove that the analysis is tight. Consider an arbitrary a and let ε be a small positive number. If the request sequence consists of the points $-\varepsilon$ and ε then the optimal solution uses only one cluster and has cost $1+(2\varepsilon)^2$ while the algorithm uses two clusters and has cost $2(1+a^2)$. Since ε can be arbitrarily small we obtain that the competitive ratio is not smaller than $2+2a^2$.

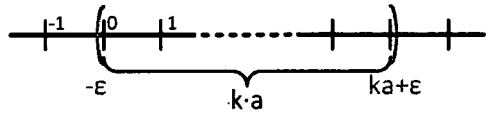


Figure 2: The interval with endpoints $-\varepsilon$ and $ka+\varepsilon$

Now suppose that all the points are requested in the interval with endpoints $-\varepsilon$ and $ka+\varepsilon$ (see Figure 2). If we use only one cluster then the cost is $1+(ka+2\varepsilon)^2$. $GRID_a$ uses $k+2$ cells, thus its cost is $(1+a^2)(k+2)$. This yields that a lower bound on the ratio of the cost of $GRID_a$ and the optimal cost tends to $F(k)$ as ε tends to 0. Therefore we obtained that the competitive ratio of $GRID_a$ is not smaller than $F(k)$ for any positive k, and this shows the tightness of our analysis. □

Corollary 1. *The smallest competitive ratio of $GRID_a$ is obtained if $\frac{1}{2\sqrt{2}} \le a \le \frac{1}{\sqrt{2}}$, then the competitive ratio of the algorithm is* 3.

Proof. First observe that $F(k) = 3$ for $k = 1$ for each value of the parameter a. If $\frac{1}{2\sqrt{2}} \le a \le \frac{1}{\sqrt{2}}$ then $F(k) \le 3$ for all integers $k > 1$ and also $2(1+a^2) \le 3$, thus the algorithm is 3-competitive. If $a > \frac{1}{\sqrt{2}}$ then $2(1+a^2) > 3$. If $a < \frac{1}{2\sqrt{2}}$ then

$F(2) > 3$, therefore if $a \notin [\frac{1}{2\sqrt{2}}, \frac{1}{\sqrt{2}}]$ then the competitive ratio of $GRID_a$ is larger than 3.

□

Remark 1. The competitive ratio of any online algorithm for the strict model is at least 2.2208. The proof of this lower bound in [22] (where the 2-dimensional variant of this problem is studied) uses one dimension so it also applies in our case.

3.2 The semi-online strict model

In the semi-online model the points arrive in ascending order. A possible algorithm to solve this problem is $SOSM_a$.

Algorithm 2 Algorithm $SOSM_a$

1. Let p be the new point.
2. If the algorithm has a cluster which contains p, then assign p to that cluster.
3. Else, open a new cluster $[p, p + a]$ and assign p to the new cluster.

Theorem 2. *The competitive ratio of algorithm $SOSM_a$ is*

$$\max\{F(\lfloor -1 + \sqrt{1 + \frac{1}{a^2}} \rfloor), F(\lceil -1 + \sqrt{1 + \frac{1}{a^2}} \rceil), 1 + a^2\}$$

where $F(k) = \frac{(k+1)(1+a^2)}{1+k^2a^2}$, $k \geq 1$.

Proof. Consider and arbitrary sequence and an optimal solution for it, denoted by OPT. We investigate the clusters of OPT separately. Consider an arbitrary cluster. Let r denote the length of this cluster.

Suppose first that $k \cdot a < r \leq (k+1) \cdot a$ for an integer $k \geq 1$. Then this optimal cluster intersects at most $k+2$ clusters (see Figure 1). We have to consider only the clusters which left endpoint is bigger than or equal to the left endpoint of this optimal cluster. The eventual other cluster which is hanging into this optimal cluster is considered with the optimal cluster on the left. Therefore, if we consider only these clusters then $SOSM_a$ has at most $(1 + a^2)(k + 1)$ cost. Thus the competitive ratio on this subsequence is at most $\frac{(1+a^2)(k+1)}{r^2+1} < \frac{(1+a^2)(k+1)}{k^2a^2+1} = F(k)$. The derivative of this function is

$$F'(k) = \frac{(1 + a^2) \cdot (1 - 2ka^2 - k^2a^2)}{(1 + k^2a^2)^2}.$$

$F'(k)$ is 0 at $k^* = -1 + \sqrt{1 + \frac{1}{a^2}}$. The second derivative of $F(k)$ is

$$F''(k) = \frac{2 \cdot (1+a^2) \cdot a^2 \cdot (k^3a^2 - 3k + 3k^2a^2 - 1)}{(k^2a^2+1)^3}$$

while $F''(k^*) < 0$ for every a (the calculations have been made in MATLAB). This yields that $F(k)$ has maximum at k^*. We have to consider the positive integers, so the maximum is attained at $k = \lfloor -1 + \sqrt{1+\frac{1}{a^2}} \rfloor$ or at $k = \lceil -1 + \sqrt{1+\frac{1}{a^2}} \rceil$.

Now suppose that $r \leq a$. Then the cluster intersects at most 2 $SOSM_a$ clusters, but we have to consider only the cluster which left endpoint is bigger than or equal to the left endpoint of this optimal cluster. Therefore, considering the requests of this cluster $SOSM_a$ has at most $1 + a^2$ cost. Thus the competitive ratio on this subsequence is at most $(1+a^2)/(1+r^2) \leq 1 + a^2$.

Now we prove that the analysis is tight. Consider an arbitrary a and let ε be a small positive number. If the request sequence consists of one point then the optimal solution has cost 1 and the algorithm uses one cluster and has cost $1 + a^2$. We obtain that the competitive ratio is not smaller than $1 + a^2$.

Now suppose that all the points are requested in the interval with endpoints $-\varepsilon$ and $ka + \varepsilon$. If we use only one cluster then the cost is $1 + (ka + 2\varepsilon)^2$. $SOSM_a$ uses $k + 1$ cells, thus its cost is $(1 + a^2)(k + 1)$. This yields that a lower bound on the ratio of the cost of $SOSM_a$ and the optimal cost tends to $F(k)$ as ε tends to 0. Therefore we obtained that the competitive ratio of $SOSM_a$ is not smaller than $F(k)$ for any positive k, and this shows the tightness of our analysis. □

Corollary 2. *The smallest competitive ratio of $SOSM_a$ is obtained if $\frac{1}{\sqrt{5}} \leq a \leq 1$, then the competitive ratio of the algorithm is 2.*

Proof: First observe that $F(k) = 2$ for $k = 1$ for each value of the parameter a. If $\frac{1}{\sqrt{5}} \leq a \leq 1$ then $F(k) \leq 2$ for all integers $k > 1$ and also $1 + a^2 \leq 2$, thus the algorithm is 2-competitive. If $a > 1$ then $1 + a^2 > 2$. If $a < \frac{1}{\sqrt{5}}$ then $F(2) > 2$, Therefore if $a \notin [\frac{1}{\sqrt{5}}, 1]$ then the competitive ratio of $SOSM_a$ is larger than 2.

Theorem 3. *The competitive ratio of any semi-online algorithm for the strict model is at least 1.6481.*

Proof. Let the first request point be $p_1 = 0$ and let a_1 be the length of the cluster which is opened by the algorithm. Let the second request point be $p_2 = a_1 + \varepsilon$. Then the online algorithm opens a new cluster with length $a_2 \geq 0$.

- If $a_1 + \varepsilon \leq 0.83035$ then
  - if $a_2 \leq 0.30817$ then another request point arrives: $p_3 = a_1 + a_2 + 2\varepsilon$.

$$\frac{A(I)}{OPT(I)} \geq \frac{3 + a_1^2 + a_2^2 + a_3^2}{1 + (a_1 + a_2 + 2\varepsilon)^2} \to \frac{3 + a_1^2 + a_2^2 + a_3^2}{1 + (a_1 + a_2)^2} \geq \frac{3 + a_1^2 + a_2^2}{1 + (a_1 + a_2)^2}$$

$$\geq \frac{3 + 0.83035^2 + 0.30817^2}{1 + (0.83035 + 0.30817)^2} > 1.6481$$

The inequality is valid because the ratio is decreasing both in a_1 and a_2; $\varepsilon \to 0$, $a_1 \leq 0.83035$, $0 \leq a_2 \leq 0.30817$ and $a_3 \geq 0$.

- if $a_2 > 0.30817$ then the request sequence stops and we have:

$$\frac{A(I)}{OPT(I)} \geq \frac{2 + a_1^2 + a_2^2}{1 + (a_1 + \varepsilon)^2} \to \frac{2 + a_1^2 + a_2^2}{1 + a_1^2}$$

$$\geq \frac{2 + 0.83035^2 + 0.30817^2}{1 + 0.83035^2} > 1.6481$$

The inequality is valid because the ratio is decreasing both in a_1 and a_2; $\varepsilon \to 0$, $a_1 \leq 0.83035$ and $a_2 > 0.30817$.

- If $a_1 + \varepsilon > 0.83035$ then

  - if $a_2 \leq 0.77894$ then another request point arrives: $p_3 = a_1 + a_2 + 2\varepsilon$. The optimal solution may use 2 clusters ($[p_1, p_1]$ and $[p_2, p_3]$) and the estimation follows:

$$\frac{A(I)}{OPT(I)} \geq \frac{3 + a_1^2 + a_2^2 + a_3^2}{2 + (a_2 + \varepsilon)^2} \to \frac{3 + a_1^2 + a_2^2 + a_3^2}{2 + a_2^2}$$

$$\geq \frac{3 + a_1^2 + a_2^2}{2 + a_2^2} \geq \frac{3 + 0.83035^2 + 0.77894^2}{2 + 0.77894^2} > 1.6481$$

The inequality is valid because the ratio is decreasing both in a_1 and a_2; $\varepsilon \to 0$, $a_1 \leq 0.83035$, $0 \leq a_2 \leq 0.77894$ and $a_3 \geq 0$.

  - if $a_2 > 0.77894$ then the request sequence stops. The optimal solution may use 2 clusters ($[p_1, p_1]$ and $[p_2, p_2]$) and we have:

$$\frac{A(I)}{OPT(I)} \geq \frac{2 + a_1^2 + a_2^2}{2} \geq \frac{2 + 0.83035^2 + 0.77894^2}{2} > 1.6481$$

The inequality is valid because the ratio is decreasing both in a_1 and a_2; $a_1 \leq 0.83035$ and $a_2 > 0.77894$.

□

4 The flexible model

4.1 The online problem

In the case of 1 dimension with the linear cost the ECC (extend closed cluster) algorithm (see [6]) has competitive ratio $\frac{1+\sqrt{5}}{2} \approx 1.618$. In [22] it is extended to 2-dimensions and it is shown that the extended algorithm ECC is not constant competitive. If we consider the 1-dimension variant with the square cost we obtain that it is neither constant competitive. The proof of that claim is the same as in [22]. As the proof shows an algorithm should limit the size of the clusters. The following extension of the $GRID_a$ algorithm satisfies this property.

Algorithm 3 Algorithm $FGRID_a$

1. Let p be the new point.
2. If the algorithm has a cluster whose current associated interval contains p, then assign p to that cluster, and do not modify the associated interval of the cluster.
3. Else, consider the cell from the grid which contains p.
   a) If this cell does not have a cluster, then open a new cluster and assign p to the new cluster. In this case the current cluster consists of a single point p.
   b) Otherwise, extend the cluster contained in the interval to cover p.

Theorem 4. *The competitive ratio of algorithm $FGRID_a$ is 2 if $\frac{1}{\sqrt{5}} \leq a \leq 1$.*

Proof. Consider an arbitrary sequence and an optimal solution for it, denoted by OPT. We investigate the clusters of OPT separately. Consider an arbitrary cluster. Let r denote the length of the side of this cluster.

Suppose that $k \cdot a < r \leq (k+1) \cdot a$ for an integer $k \geq 1$. Then the optimal cluster intersects at most $k+2$ cells of the grid. The cells which are not at endpoints of the optimal cluster might be completely covered by $FGRID_a$. Consider now the end cells, denote by A_1 and A_2 the square costs of the intervals covered by the optimal cluster in these end cells and let $A = A_1 + A_2$. At these end cells of the optimal cluster we have two possibilities. If the cell has no intersection with other optimal clusters, then OPT and $FGRID_a$ cover the same parts of the cell. If the cell intersects at least one other optimal cluster, then it might be completely covered by $FGRID_a$ but then its online cost is divided between at least two optimal clusters and we have to consider only the half of this cost here which is $\frac{1}{2} \cdot (1+a^2)$. Therefore we obtained that assigning a total cost $2 \cdot \frac{1}{2}(1+a^2) + A$ from the online cost to these end cells we cover the full online cost by the costs assigned to the optimal clusters. Thus we assigned at most $(1+a^2) \cdot k + 2 \cdot \frac{1}{2}(1+a^2) + A = (k+1)(1+a^2) + A$

cost from $FGRID_a(I)$ to this optimal cluster. The cost of the optimal cluster is at least $1 + k^2a^2 + A$. If we consider the ratio of these costs we obtain that it is

$$\frac{(k+1)(1+a^2)+A}{k^2a^2+1+A} \leq \frac{(k+1)(1+a^2)}{k^2a^2+1}.$$

If $k = 1$ then this ratio is 2 for each a. For $k > 1$ and $a > 1$ this ratio is smaller than 2.

Now suppose that $r < a$. Then the optimal cluster intersects at most 2 cells from the grid. At these cells again we have two possibilities. If the cell has no intersection with other optimal clusters, then OPT and $FGRID_a$ cover the same parts of the cell. If the cell intersects at least one other optimal cluster, then it might be completely covered by $FGRID_a$ but then its cost is divided between at least two optimal clusters and we have to consider only the half of this cost here which is $\frac{1}{2}(1+a^2)$. Therefore we obtained that assigning a total cost $2 \cdot \frac{1}{2}(1+a^2) + r^2 = 1 + a^2 + r^2$ from $FGRID_a(I)$ to this cluster we cover $FGRID_a(I)$ by the costs assigned to the optimal clusters. The cost of the optimal cluster is at least $1 + r^2$. If we consider the ratio of these costs we obtain that it is

$$\frac{1+a^2+r^2}{1+r^2} \leq 1 + a^2.$$

On the other hand $1 + a^2 \leq 2$ if $a \leq 1$.

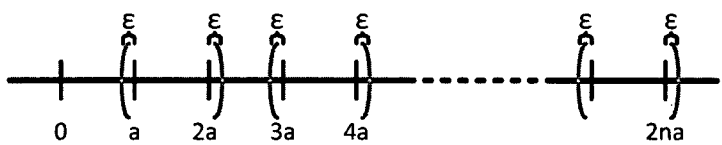


Figure 3: The competitive ratio of $FGRID_a$: the n intervals with endpoints $(2i-1)a - \varepsilon$ and $2ia + \varepsilon$, $i = 1, ..., n$

Now we prove the tightness. Fix an a and let $\varepsilon > 0$ be a small positive number. Consider the input I_n where all the points in the n intervals with endpoints $(2i-1)a - \varepsilon$ and $2ia + \varepsilon$, $i = 1, ..., n$ are requested (see Figure 3).

Then a solution can use each such interval as a cluster therefore the cost of OPT is at most $n \cdot (1 + (a+\varepsilon)^2)$. Now investigate the behavior of $FGRID_a$. It covers completely the grid cells with endpoints ia and $(i+1)a$, $i = 1, ..., 2n-1$ and with ε^2 cost the 2 end cells.

Therefore we obtained that $FGRID_a(I_n) \geq (1+a^2)(2n-1) + 2(1+\varepsilon^2)$. The ratio $FGRID_a(I_n)/OPT(I_n)$ tends to 2 as ε tends to 0 and n tends to ∞, thus we proved that the algorithm is not better than 2-competitive. □

Theorem 5. *The competitive ratio of any online algorithm for the flexible model is at least 1.2993.*

Proof. Suppose that there exists an online algorithm with smaller competitive ratio than 1.2993, denote it by A. Consider the following input sequence. The first two points are $p_1 = 0$ and $p_2 = 0.878$. Now distinguish the following cases.

- If A assigns these points to different clusters then three more points arrive: $p_3 = 0.329$, $p_4 = 0.439$ and $p_5 = 0.549$. The optimal algorithm uses only one cluster and its cost is $1 + 0.878^2 = 1.770884$. The cost of the online algorithm is at least $2 + 0.329^2 + 0.439^2 = 2.300962$ (it is the case when A extends both existing clusters "inward": one to the nearest new point and the other to the second new point – see Figure 4), thus the ratio is at least $2.300962/1.770884 > 1.2993$, which is a contradiction.

- If A assigns the points to one cluster then two more points arrive $p_3 = -0.355$ and $p_4 = 1.233$. Then the optimal algorithm uses two clusters, both of them have size 0.355, thus the optimal cost is $2 \cdot (1 + 0.355^2) = 2.25205$. The cost of A is at least $2 + (0.878 + 0.355)^2 = 3.520289$, thus the ratio is at least $3.520289/2.25205 \approx 1.563149$, which is a contradiction.

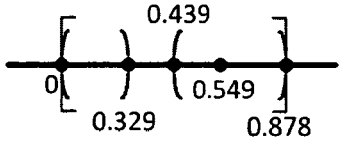


Figure 4: Lower bound in the flexible model: the cost of the online and offline algorithms

We obtained contradiction in both cases, thus we proved the theorem.

□

4.2 The semi-online flexible model

Theorem 6. *The competitive ratio of any semi-online algorithm for the flexible model is at least 1.1991144.*

Proof. Suppose that there exists a semi-online algorithm with smaller competitive ratio than 1.1991144, denote it by A. Let the first two request points be $p_1 = 0$ and $p_2 = 0.81725$.

- If the semi-online algorithm A puts them into one cluster then another request point arrives $p_3 = 1.29147$. The cost of the semi-online algorithm is at least $1 + 1.29147^2 = 2.6678947609$ (it is the case when the algorithm extends the existing cluster to the new point) while the optimal offline algorithm uses two clusters with p_1 in the first and p_2 and p_3 in the second cluster. Its cost is $2 + 0.47422^2 = 2.2248846084$, so we obtain:

$$\frac{A(I)}{OPT(I)} \geq \frac{2.6678947609}{2.2248846084} \approx 1.199116 > 1.1991144$$

- If the semi-online algorithm A puts p_1 and p_2 into two clusters, the sequence stops. The offline algorithm puts them into one cluster, so the competitive ratio is:

$$\frac{A(I)}{OPT(I)} \geq \frac{2}{1+0.81725^2} \approx 1.199114409 > 1.1991144$$

In both cases we have contradiction so the claim of the theorem holds.

□

Remark 2. We note that a similar modification to the algorithm $SOSM_a$ like in the online case (modification of $GRID_a$ that led to the algorithm $FGRID_a$) does not result in a better competitive ratio than 2 (like in the online case with algorithm $FGRID_a$).

References

[1] Anagnostopoulos, A., Bent, R., Upfal, E., and Van Hentenryck, P. A simple and deterministic competitive algorithm for online facility location. *Information and Computation*, **194(2)**, 175–202, 2004.

[2] Bilo, V., Caragiannis, I., Kaklamanis, C., and Kanellopoulos, P. Geometric Clustering to Minimize the Sum of Cluster Sizes. *ESA '05, LNCS 3669*, pp. 460–471, 2005.

[3] Borodin, A. and El-Yaniv, R. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[4] Chan, T. M. and Zarrabi-Zadeh, H. A randomized algorithm for onine unit clustering. *Theory of Computing Systems*, **45(3)**, 486–496, 2009.

[5] Charikar, M., Chekuri, C., Feder, T., and Motwani, R. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, **33(6)**, 1417–1440, 2004.

[6] Csirik, J., Epstein, L., Imreh, Cs., and Levin, A. Online Clustering with Variable Sized Clusters. *Algorithmica*, DOI: 10.1007/s00453-011-9586-2, 2011

[7] Divéki, G. and Imreh, Cs. Online facility location with facility movements. *Central European Journal on Operations Research*, **19(2)**, 191–200, 2011.

[8] Ehmsen, M. R. and Larsen, K. S. Better bounds on online unit clustering. *In Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT2010)*, pages 371–382, 2010.

[9] Epstein, L., Levin, A., and van Stee, R. Online unit clustering: Variations on a theme. *Theoretical Computer Science*, **407(1-3)**, 85–96, 2008.

[10] Epstein, L. and van Stee, R. On the online unit clustering problem. *ACM Transactions on Algorithms*, **7(1)**, Article 7 (18 pages), 2010.

[11] Fiat, A., Woeginger, G. J., editors. *Online algorithms: The State of the Art, LNCS 1442.* Springer-Verlag Berlin, 1998.

[12] Fotakis, D. Incremental Algorithms for Facility Location and k-Median. *Theoretical Computer Science*, **361**, 275–313, 2006.

[13] Fotakis, D. A Primal-Dual Algorithm for Online Non-Uniform Facility Location. *Journal of Discrete Algorithms*, **5**, 141–148, 2006.

[14] Fotakis, D. Memoryless Facility Location in One Pass. *Proceedings of STACS '06, LNCS 3884*, 608–620, 2006.

[15] Fotakis, D. On the Competitive Ratio for Online Facility Location *Algorithmica*, **50(1)**, 1–57, 2008.

[16] Gibson, M., Kanade, G., Krohn, E., Pirwani, I. A., and Varadarajan, K. On Metric Clustering to Minimize the Sum of Radii. *Algorithmica*, **57**, 484–498, 2010.

[17] Imreh, Cs. Competitive analysis. In *Algorithms of Informatics Volume 1*, ed. Antal Iványi, mondAt, Budapest 2007, 395–428.

[18] Levin, A. A generalized minimum cost k-clustering. *ACMTrans. Algorithms*, **5(4)**, Article 36, 2009.

[19] Meyerson, A. Online facility location. *In Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS2001)*, pages 426–431, 2001.

[20] Shah, R. and Farach-Colton, M. Undiscretized dynamic programming: faster algorithms for facility location and related problems on trees. *In Proc. of the* 13^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pp. 108–115, 2002.

[21] Zarrabi-Zadeh, H. and Chan, T. M. An improved algorithm for online unit clustering. *Algorithmica*, **54(4)**, 490–500, 2009.

[22] Divéki, G. and Imreh, Cs. An online 2-dimensional clustering problem with variable sized clusters *Submitted to OPTE, 2011.*

Acta Cybernetica **21** (2013) 89–103.

Spectrum Skeletonization: A New Method for Acoustic Signal Feature Extraction

Tibor Dobján* and Gábor Németh†

Abstract

Vibration Analysis Tests (VAT) and Acoustic Emission tests (AE) are used in several industrial applications. Many of them perform analysis in the frequency domain. Peaks in the power density spectrum hold relevant information about acoustic events. In this paper we propose a novel method for feature extraction of vibration samples by analyzing the shape of their auto power spectrum density function. The approach uses skeletonization techniques in order to find the hierarchical structure of the spectral peaks. The proposed method can be applied as a preprocessing step for spectrum analysis of vibration signals.

Keywords: spectrum analysis, skeletonization, spectrum segmentation, feature extraction

1 Introduction

Acoustic events play an important role in several fields of industries [10, 25, 27, 30, 32]. Nowadays, vibration analysis tests are one of the applied methods for non-destructive testing. Vibration analysis tests are used to diagnose rotating machines [31] and for detection of hit and leakage [8]. With acoustic emission burst analysis we can draw conclusions about existing cracks and crack propagation [26].

These methods investigate acoustic samples by extracting some features of the vibration signal. There are three major methods for vibration signal feature extraction:

1. In time domain some statistical parameters (e.g., root mean square, mean, variance, skewness, kurtosis) can be computed [17, 27]. There are also time synchronous average signal (TSA) based methods, filter based methods, and stochastic methods [8, 28].

*College of Dunaújváros, Hungarian Acoustic and Industrial Diagnostic Testing Laboratory, E-mail: dobjan.tibor@email.duf.hu

†University of Szeged, Institute of Informatics, E-mail: gnemeth@inf.u-szeged.hu

2. Frequency domain analysis is used to investigate the frequency components of periodic signal samples [17,32,36]. Applications of frequency-domain analysis use Fast Fourier Transformation (FFT) [14].

3. Time-frequency domain analysis can be used for non-periodic and non-stationary signal samples, like bursts [17, 36]. Applications of time-frequency domain analysis use continuous wavelet transforms (CWT) [23].

In this paper we propose a novel method for feature extraction from the auto power spectrum density (APSD) function. Our approach produces an automatic partitioning of a given APSD function along the frequency axis, where each range contains a dominant frequency component. Our method ensures that the bounds of frequency ranges, referred to as *separators* in the following, are defined within the local minima of the APSD function: by this way no relevant peak in the power spectrum density is split to disjunct frequency ranges. The peaks of the APSD function are detected by the skeletonization method [12, 29]. Skeleton is a shape descriptor that summarizes the general form of objects. Since skeletons provide also structural properties of the power spectrum, each separator can be located between two skeletal branches.

The rest of this paper is organized as follows: Section 2 gives a short description about our used data. Section 3 details how to determine the relevant frequency ranges of the APSD function. Section 4 contains discussion about some possible improvements and connecting problems of our basic methods. The main conclusions of this document are summarized in Section 5.

2 Collection of recorded signals

Two datasets of acoustic events are examined in this paper. First, we have collected some acoustic emission event from Gleeble-measurements, however, the count of recorded signals was not large enough for a reliable evaluation, hence we also created a dataset of noises from knocks. Each event was registered manually.

2.1 Gleeble-measurements

In the Gleeble-laboratory of College of Dunaújváros we carried out a measurement by Gleeble 3800 thermo-mechanical physical simulator, which is able to preform tensile tests and hot deformation on steel specimens.

In the test we put a standard tensile test specimen into the jaws of Gleeble (see Figure 1). We placed two acoustic emission sensors at the end of the specimen behind the jaws. The specimen was then heated according to the heating profile (see Figure 2). It was heated up at 10 °C/s to 900 °C, and then held at temperature for 120 seconds. The specimen was subsequently controlled cool at 10 °C/s to approximately 400 °C and free cooled the rest of the way to room temperature.

During the test, we listened to acoustic emission events in the 1MHz frequency range (F_s = 2MHz). The measurement systems stored the whole time-series during

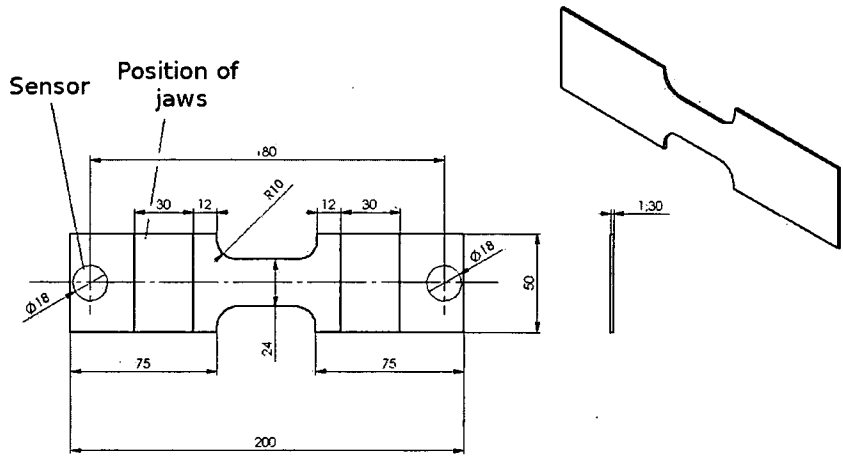


Figure 1: Drawing of a test specimen to be put into the jaws of Gleeble

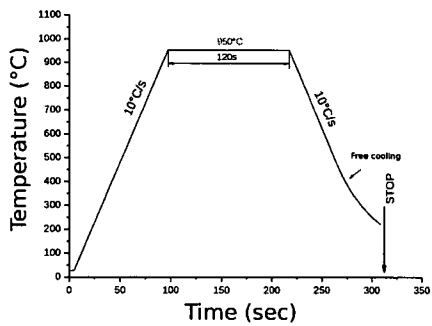
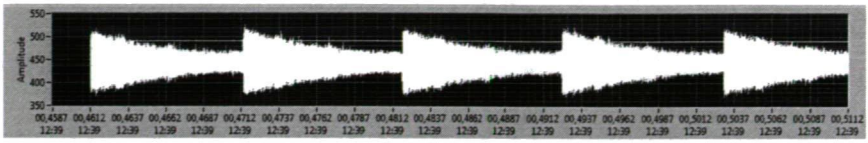


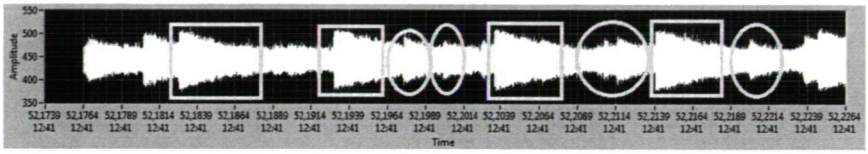
Figure 2: Thermal profile of the test specimen

the test. We found that there is an artificial periodicity in the most of events (see Figure 3(a)).

We found real acoustic events in the beginning of the cooling period (see Figure 3(b)). These acoustic events come from the observed physical phenomenon that can be used in the industrial applications. Unfortunately, we have only a few recorded signals of the artificial periodic noise. To show that our method produces different spectrum segmentation for different source of noises, we created a database of acoustic events.



(a)



(b)

Figure 3: Time series of artificial periodic noise (a). Acoustic events on the cooling period (b). Signal segments marked by squares are periodic artificial, while the segments marked by ellipses are real acoustic events.

2.2 Knocking noises

The acoustic events were recorded by a general headset microphone to a single channel WAV file with 44100Hz sampling frequency (F_s = 44100Hz). As for gathering well-identified short time acoustic signal, we have dropped 6 objects having different shapes and materials into a china and a plastic dishes: toothpick (wood), match (wood), staple (metal), safety pin (metal), screw (metal), and nail (metal). Knocking these objects in the dishes indicates 12 strict classes of noises, but as for considering the material of objects, there are four classes taken into the account. Each of the 12 types of noises contains 100 recorded signals.

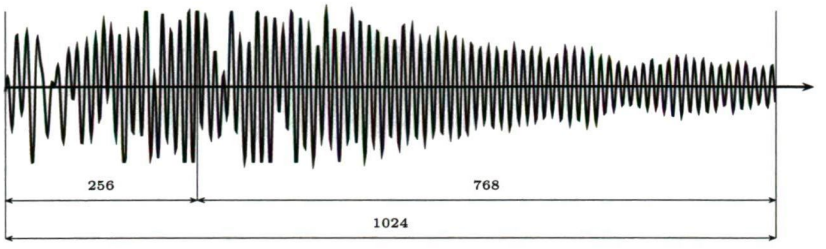
The recorded signals have various lengths, hence we had to cut the investigated signals for the same size (i.e., 1024 samples (see Figure 4)). These parts are cutted out around the first location where the signals reach their maximum amplitudes (see Figure 4). One may think that there is no difference between recorded signals, however there were various distances between the dish and the microphone, moreover the dropping height of the objects also differed during the recording.

3 Feature extraction from vibration signals

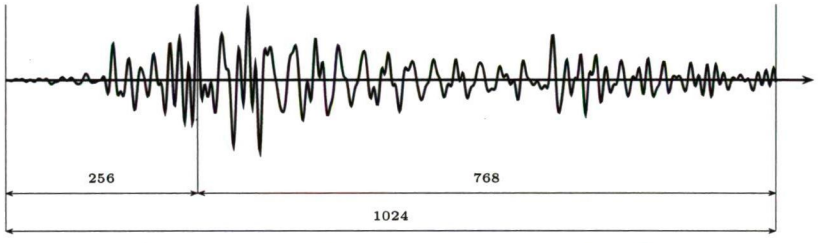
In this section, we will explain our proposed method for feature extraction.

Let $x(n)$ be a vibration signal containing n sampled data and the i^{th} value of $x(n)$ is denoted by x_i $(i = 1, \ldots, n)$.

Let us denote f_{max} the maximal frequency that is $\frac{F_s}{2}$, and let Δf the frequency resolution, where $\Delta f = \frac{f_{max}}{\frac{n}{2}}$. Furthermore, we denote the set of investigated frequencies by $\mathcal{F} = \{\Delta f, 2\Delta f, \ldots, f_{max}\}$.



vibration signal of knocking safety-pin in china



vibration signal of knocking match in plastic dish

Figure 4: A various vibration signals using 1024 samples. The sampled data begins $1/4n$ before and ends $3/4n$ after location of the maximum peak amplitude (where n is the number of samples).

To compute the frequency spectrum we use Fast Fourier Transform (FFT). The APSD function of vibration signal $x(n)$ is computed by Equations (1) and (2) [14].

$$apsd = \log\left(\frac{1}{n}|FFT(x(n))|\right) \tag{1}$$

$$\begin{aligned} APSD(f) &= \min(apsd) + apsd(i), \\ f &= \Delta f \cdot i, \quad (i = 1, \ldots, n) \end{aligned} \tag{2}$$

We use only the positive half of the APSD function. Figure 5 shows an example for an APSD function.

Next, the APSD function of signal $x(n)$ is mapped into a quadratic binary image I_{APSD} :

$$I_{\text{APSD}}(u,v) = \begin{cases} 1 & \text{if} \quad v \leq \left\lfloor \dfrac{n}{2} \dfrac{APSD(u \cdot \Delta f)}{\max\limits_{i \in \mathcal{F}}(APSD(i))} \right\rfloor \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

In I_{APSD} the background is white and black points represent the quantized values below the APSD function.

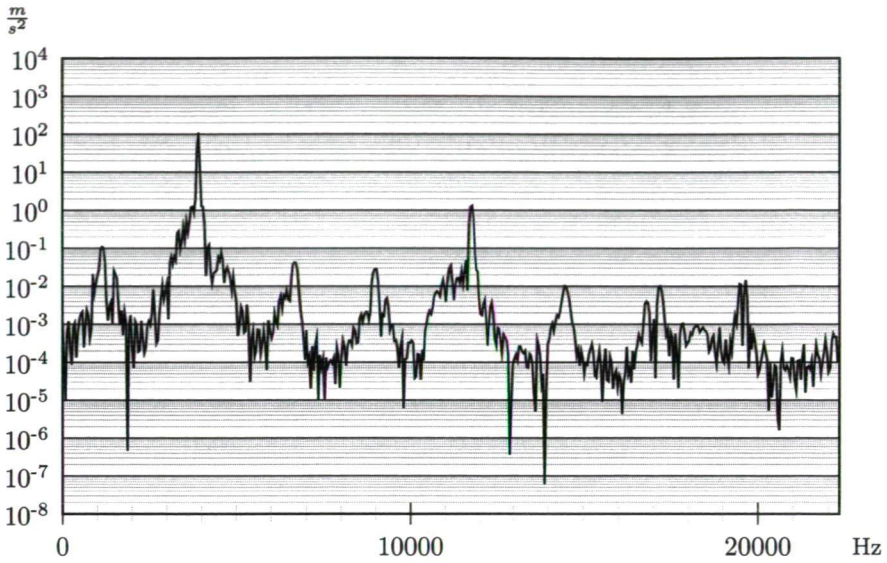
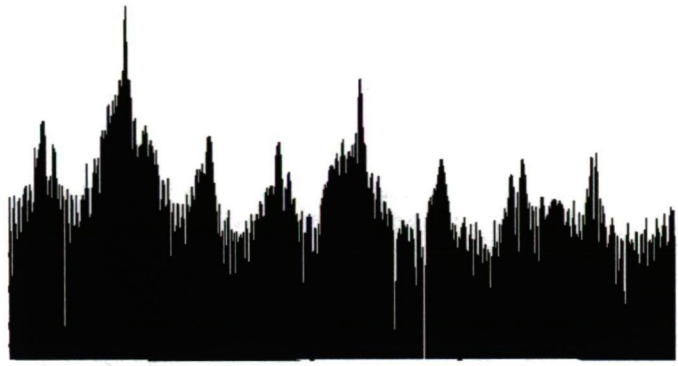


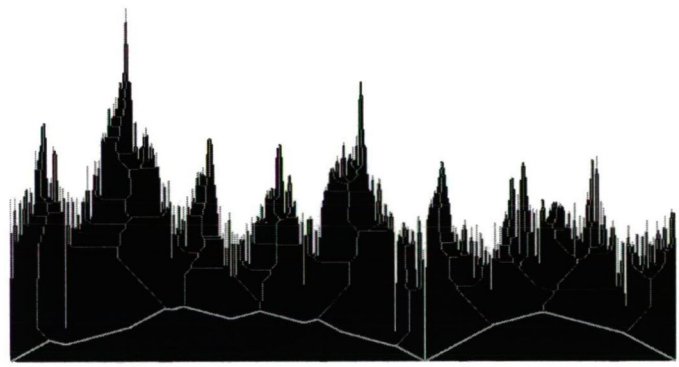
Figure 5: APSD function of a vibration signal

After we created the APSD function image I_{APSD}, the dominant peaks and frequency ranges have to be determined. First we extract the centerline of the APSD function image by a topology preserving skeletonization algorithm [12, 29]. Here we used a thinning algorithm [3, 6, 12, 13, 18, 19, 20] to extract the centerline, since it is the most efficient skeletonization technique for digital pictures. Skeletal branches are growing into dominant peaks of function. Furthermore, the centerline indicates the hierarchical structure of peaks as well.

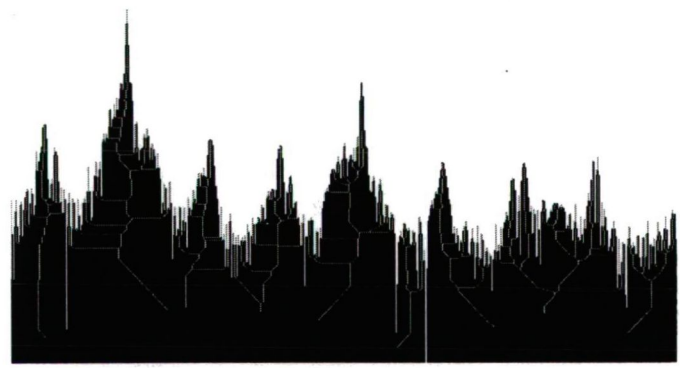
As a next step, the lowest connected skeletal curve segments (i.e., that are depicted with thick gray curve in Figure 6(b)) are removed. This curve segment is characterized as the set of skeletal points having the maximum y coordinate in each column in image I_{APSD} (if the $(0, 0)$ point of I_{APSD} is its upper-left corner, and y coordinates increasing vertically down). This provides that the skeletal branches growing into the peaks of APSD function will be disconnected at their roots. Removal of the lowest curve segments disconnects only the skeletal branches into isolated *skeletal trees*, but does not affect the hierarchy of the side branches in the trees (see Figure 6(b) and (c)). The *width of a tree* can be determined by measuring the horizontal distance between the endpoints of the most left and most right branch of the tree. It is not hard to see that, the ranges (i.e., the widths) of the skeletal trees indicate disjunct intervals in the whole frequency range, however the union of these disjunct intervals does not form to whole frequency range. Our aim is to partition the whole frequency range into intervals by separators such that the union of the intervals matches the length of the whole range, with no peaks split by any of the separators.



(a) Spectrum image



(b) Spectrum skeleton (centerline)



(c) Spectrum skeletal trees

Figure 6: Spectrum is mapped to a binary image (a), centerline (b), and the skeletal trees (c) of the spectrum. Note that in (b) the lower skeletal branches (that ensures the connectedness) of the skeleton is removed. Hence only the skeletal trees are left.

Let $\mathcal{T}(1)$ and $\mathcal{T}(2)$ two adjacent skeletal trees. Furthermore, let us denote the most left and most right upper endpoint of tree $\mathcal{T}$ by $\mathcal{T}_l$ and $\mathcal{T}_r$, respectively. We propose that, the separator line between $\mathcal{T}(1)$ and $\mathcal{T}(2)$ is chosen at the frequency, where the APSD function reaches its minimum between $\mathcal{T}(1)_r$ and $\mathcal{T}(2)_l$ (see Figure 7). Our method is sketched in Algorithm 1.

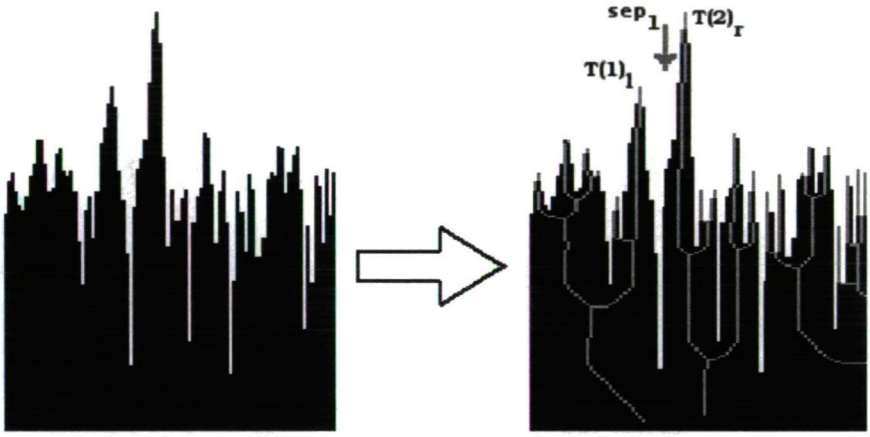


Figure 7: A part of the spectrum image is enlarged (left), and the location between the skeletal trees (right). The separator is drawn where the APSD function reaches its minimum between skeletal trees $\mathcal{T}(1)$ and $\mathcal{T}(2)$.

Algorithm 1 Partition the APSD function to dominant frequency ranges

Func PartitionAPSD(APSD($\mathcal{F}$))

1: Map the APSD($\mathcal{F}$) function to the image I. *Getting a binary picture about the APSD function*
2: Extracting centerlines of image I.
3: **for** $i = 1$ **to** $n/2$ **do** *Split centerlines into skeletal trees.*
4: Delete (i.e., change it to white) the skeletal points with maximum y coordinate in column i.
5: **end for**
6: Let c be the count of skeletal tree. *Labeling is used.*
7: Let the trees are labeled from 1 to c from left to right (i.e., the left most tree has Label 1, and the right most tree has Label c).
8: **for** $i = 1$ **to** $c - 1$ **do** *Finding separators between two trees.*
9: Find $sep_i \in [\mathcal{T}(i)_r, \mathcal{T}(i+1)_l]$, where $APSD([\mathcal{T}(i)_r, \mathcal{T}(i+1)_l])$ reaches its global minimum.
10: **end for**
11: **return** $\{sep_1, \ldots, sep_{c-1}\}$

In the case of spectrum skeletons, the challenge is to match the appropriate branches to each other. In acoustic vibration signals, numerous of extra peaks may appear in the spectrum because of various acoustic noises. Thinning algorithms may produce many skeletal branches that are grown from the extremities of boundary points. Several points in a spectrum-image may fulfill the applied geometric constraints of the thinning algorithm, which means each peak may be represented by a skeletal side branch. The count and the position of the endpoints is strongly dependent on the variance of the spectrum. Hence, instead of the endpoints we focus of usage of skeletal components and the hierarchic structure of skeletal trees, which indicate the dominant spectrum segments.

In spectrum segmentation, conventional methods partition the frequency range equidistantly (see Figure 8(a)) [11,33]. Equidistant segmentation does not consider the shape of the spectrum which means that some peaks of the spectrum may be split during the segmentation. According to our concept, dominant peaks hold relevant information about the source of the event.

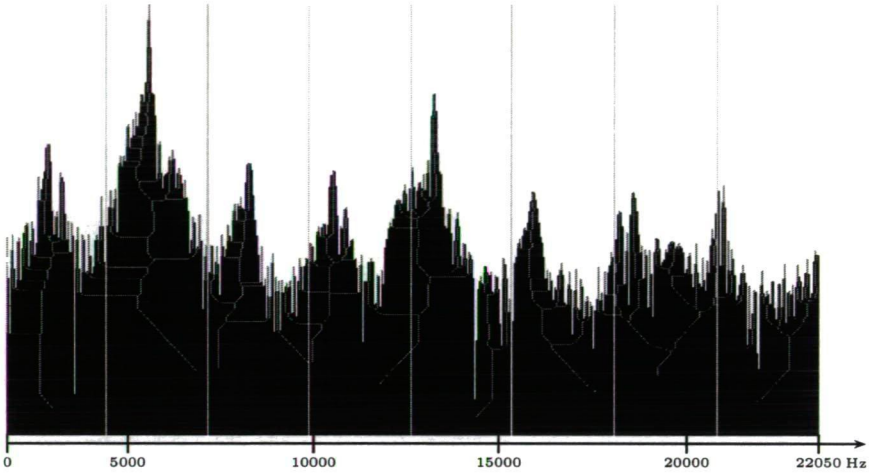
In our approach, where the frequency ranges are partitioned dynamically (see Figure 8(b)), each relevant peak is represented by a skeletal tree. The width of a frequency range is determined by the difference of local minima around the given skeletal tree. The APSD functions of recorded signals are always differ from each other because of the background noise of the signal. Hence there is no guarantee that skeletons will indicate the same number of branches and the same number of frequency domain partitions for each APSD function from the class of similar acoustic events. One may think that this noise produce several trees and small frequency ranges. Each noisy peak is represented by a skeletal side branch, but this does not dominant affect to the tree widths. With the help the dynamic partitioning can be described a given APSD function.

4 Further works

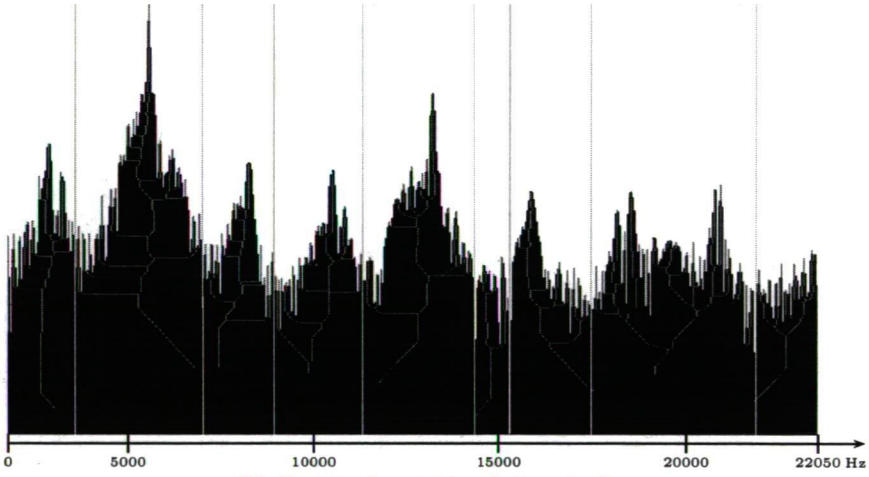
The proposed method holds some new ideas that point to a new way of spectrum segmentation. These could be a component of a complex acoustic event recognition system. The quality of event registration is important since it is the input of the segmentation algorithm. It may influence the results of feature extraction. Using skeletal structures we could represent the APSD spectra by graphs. We assume that graph (or sub-graph) isomorphism also can be applied in identification. For identification and clustering of acoustic events we use machine learning methods. What follows is an explanation of their planned use.

4.1 Auto event registration

Our feature extraction method is based on Fourier transformation. The Fourier transformation is not suitable for analysis transient acoustic events because it is suitable only for stationary time series. To get the best approximation, we should select the beginning and the end time of the event correctly, with following rules:



(a) Spectrum is partitioned into ranges of same size



(b) Spectum is partitioned dinamically

Figure 8: Regular (equidistant) partition of the frequency range (a). Dynamic partitioning using skeletal trees (b). Frequency axis can be computed by product the position of the separator by Δf.

- The selected piece of time series should contain only the event, with the least background noise. (There is always background noise during the event.)
- It should contain the entirety of the event

There are more solutions on the literature for this problem like Auto regression – Akaike Information Criterion (AR-AIC) [21] or Sequential Probability Ratio Test (SPRT) [8]. The frequently used approaches are the threshold-based algorithms.

4.2 Using graph theory

Since skeletonization methods produce reduced structures which represent the dominant peaks in the APSD functions, they are able to build a graphic structure. In future work we would like to build graphs from skeletal branches, and sub-graph isomorphism may help us to find out differences in structure. Since the area below the APSD function does not contain any cavity, this area can be represented by a tree graph derived from centerlines. This concept leads to comparison of graph (or sub-graph) structures for identification. We can assume that similar acoustic events have similar skeletal structures that can be identified by corresponding graph branches. Graph correspondences constructed by centerlines gave a solution for several applications [1, 7, 15, 24, 35].

4.3 Machine learning

In order to recognize the source of the signal, some machine learning algorithms are applied. Machine learning classifies signals based on some features (e.g., maximum amplitude, variance of the signal, or partial RMS rates) [4,5,9,22]. Machine learning is composed of two phases: in the learning phase, a set of sample data is assigned by class labels and their considered features are extracted; while in the second phase some unclassified data must be assigned to a class only based on their features. We investigated of usage the following classifiers in the future work: Linear classifiers determine a hyperplane in the feature space that separates data into two classes correctly. Support vector machines (SVM) [2, 16, 34] are kernel based classifiers that allow non-linear class borders for better accuracy. K-means classifiers are capable of distinguish more than two classes as well. Both of the linear and SVM classifiers are binary classifiers (i.e., they separate data into two disjunct sets), however there exists some extension of them to distinguish more than two classes. One of the concepts is *one-against-all*, while the other one is the *one-against-one* comparison [34].

5 Conclusion and open questions

In this paper we presented a novel method for spectrum segmentation based on skeletons. The proposed algorithm can be applied as a preprocessing step in spectrum description and analysis.

The APSD function computed from acoustic vibration signals are mapped to a binary image, where the background is represented by white pixels, and the area below the APSD function is formed by black pixels. The centerline of the APSD function image is extracted by a topology preserving thinning algorithm. The dominant peaks in the APSD function image are represented by skeletal branches which hold some structural information about the set of local maxima of APSD function. The extracted centerline is used for spectrum segmentation. Removal of the lowest segments of centerline splits the "skeleton" into skeletal trees. These skeletal trees indicate the main frequency components in the APSD function, however their union does not form the whole frequency range. Hence, we found the border of the frequency ranges in the minimum of function value in the frequency intervals between two skeletal trees. This fulfills our requirement to the effect that no peak in the spectrum is split into disjunct frequency segments. According to our concept, peaks in the APSD function hold relevant information about the origin of the acoustic signal.

We are working on some open questions in our further works:

- Thinning methods work on binary images, and quantization the APDS values yield some distortions in our data. Voronoi skeleton [12, 29] works with the original values, hence it can be a more appropriate solution for segmentation based on skeletons.
- In pattern recognition and shape analysis skeletons are converted to graphs, then it can be used for the formal description of the shape. In the further works we are focusing to the graph representation to describe similarity of APSD functions.

The proposed feature extraction by skeletons opens a new window for vibration signal processing and spectrum segmentation.

References

[1] Bai, X. and Latecki, L. J. Path similarity skeleton graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(7):1282 –1292, july 2008.

[2] Ben-Hur, A. and Weston, J. A user's guide to support vector machines. In Carugo, Oliviero, Eisenhaber, Frank, and Walker, John M., editors, *Data Mining Techniques for the Life Sciences*, volume 609 of *Methods in Molecular Biology*, pages 223–239. Humana Press, 2010.

[3] Bertrand, G. and Couprie, M. Two-dimensional parallel thinning algorithms based on critical kernels. *Journal Mathematical Imaging and Vision*, 31(1):35–56, May 2008.

[4] Bishop, C. M. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[5] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[6] Couprie, M. Note on fifteen 2d parallel thinning algorithms. In *Internal Report, Université de Marne-laVallée*, 2006.

[7] Demirci, M. F. and Osmanlioglu, Y. Many-to-many matching under the l_1 norm. In *ICIAP*, pages 787–796, 2009.

[8] Dobján, T., Pletl, Sz., Deák, T., Doszpod, L., and Pór, G. Identification of the place and materials of knocking objects in flow induced vibration. *Acta Cybernetica*, pages 53–67, 2010.

[9] Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern Classification.* Wiley, New York, 2. edition, 2001.

[10] Eltabach, M., Vervaeke, T., Sieg-Zieba, S., Padioleau, E., and Berlingen, S. Features extraction using vibration signals for condition monitoring of lifting cranes. In *Proc. of International conference (Surveillance 6), Compiegne*, 2011.

[11] Ghaderi, H. and Kabiri, P. Automobile independent fault detection based on acoustic emission using FFT. In *Singapore International NDT Conference & Exhibition (SINCE 2011)*, 2011.

[12] Gonzalez, R. C. and Woods, R. E. *Digital Image Processing (3rd Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[13] Guo, Z. and Hall, R. W. Parallel thinning with two-subiteration algorithms. *Commun. ACM*, 32(3):359–373, March 1989.

[14] Heinzel, G., Rüdiger, A., and Schilling, R. Spectrum and spectral density estimation by the Discrete Fourier Transform (DFT), including a comprehensive list of window functions and some new at-top windows, 2002.

[15] Hiransakolwong, N., Vu, K., Hua, K. A., and Lang, S.-D. Many-to-many skeletal-graphs matching approach to shape recognition. In *Proc. International Conference: Sciences of Electronic, Technologies of Information and Telecommunications*, 2004.

[16] Hsu, C.-W., Chang, C.-C., and Lin, C.-J. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003.

[17] Jossa, I., Marschner, U., and Fischer, W.-J. *Signal-based feature extraction and SOM based dimension reduction in a vibration monitoring microystem.*, pages 283–288. London: Springer, 2001.

[18] Kardos, P., Németh, G., and Palágyi, K. An order-independent sequential thinning algorithm. In Wiederhold, P. and Barneva, R. P., editors, *IWCIA*, volume 5852 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2009.

[19] Kardos, P. and Palágyi, K. Order-independent sequential thinning in arbitrary dimensions. In *Proc. IASTED International Conference on Signal and Image Processing and Applications, SIPA 2011*, pages 129–134, Crete, Greece, 2011.

[20] Kong, T. Yung and Rosenfeld, A., editors. *Topological Algorithms for Digital Image Processing.* Elsevier Science Inc., New York, NY, USA, 1996.

[21] Küperkoch, L. *Automated recognition, phase arrival time estimation and location of local and regional earthquakes.* PhD thesis, Ruhr-Universität, Bochum, Germany, 2010.

[22] MacKay, D. J. C. *Information Theory, Inference & Learning Algorithms.* Cambridge University Press, New York, NY, USA, 2002.

[23] Mallat, S. *A Wavelet Tour of Signal Processing, 2nd ed.* Academic, San Diego, CA, 1999.

[24] Palágyi, K., Tschirren, J., Hoffman, E. A., and Sonka, M. Quantitative analysis of pulmonary airway tree structures. *Computers in Biology and Medicine*, 36(9):974–996, 2006.

[25] Peng, Z., Chu, F., and He, Y. Vibration signal analysis and feature extraction based on reassigned wavelet scalogram. *Journal of Sound and Vibration*, 253(5):1087 – 1100, 2002.

[26] Pór, G., Doszpod, L., and Dobján, T. Developing an acoustic emission measuring system based on modular high speed data acquisition devices. In *30th European Conference on Acoustic Emission Testing and 7th International Conference in Acoustic Emission*, September 2012.

[27] Samanta, B. and Al-Balushi, K.R. Artificial neural network based fault diagnostics of rolling element bearings using time-domain features. *Mechanical Systems and Signal Processing*, 17(2):317 – 328, 2003.

[28] Schoonewelle, H., Hagen, T. H. J. J. Van Der, and Hoogenboom, J. E. A comparison of three time-domain anomaly detection methods. *Annals of Nuclear Energy*, 23(2):159–170, 1996.

[29] Siddiqi, K. and Pizer, S. *Medial Representations: Mathematics, Algorithms and Applications.* Springer Publishing Company, Incorporated, 1st edition, 2008.

[30] Su, H., Chong, K. T., and Ravi Kumar, R. Vibration signal analysis for electrical fault detection of induction machine using neural networks. *Neural Computing and Applications*, 20(2):183–194, March 2011.

[31] Szántó, J. and Nagy, I. Integration of fault diagnostic technologies into a complex condition monitoring system and its practical results. In *World Congress on Maintenance (WCM-2008)*, pages 24–26, November 2008.

[32] Tang, J., Chai, T.Y., Zhao, L.J., and Wen, Y. Feature extraction and selection based on vibration spectrum with application to mill load modeling. In *Advanced Control of Industrial Processes (ADCONIP), 2011 International Symposium on*, pages 266–271, may 2011.

[33] Tsunodaa, T., Kato, T., Hirata, K., Sekido, Y., Sendai, K., Segawa, M., Yamatoku, S., Morioka, T., Sanoa, K., and Tsuneokaa, O. Studies on the loose part evaluation technique. *Progress in Nuclear Energy*, 15:569–576, 1985.

[34] Weston, J. and Watkins, C. Support vector machines for multi-class pattern recognition. In *ESANN*, pages 219–224, 1999.

[35] Xu, Y., Wang, B., Liu, W., and Bai, X. Skeleton graph matching based on critical points using path similarity. In Zha, Hongbin, Taniguchi, Rin-ichiro, and Maybank, Stephen, editors, *Computer Vision – ACCV 2009*, volume 5996 of *Lecture Notes in Computer Science*, pages 456–465. Springer Berlin / Heidelberg, 2010.

[36] Yang, H., Mathew, J., and Ma, L. Vibration feature extraction techniques for fault diagnosis of rotating machinery : a literature survey. In *Asia-Pacific Vibration Conference*, pages 801–807, Gold Coast, Australia, 2003.

Acta Cybernetica **21** (2013) 105–122.

The Vagueness Measure: A New Interpretation and an Application to Image Thresholding

József Dombi* and Gergely Gulyás†

Abstract

Here, we introduce a new interpretation of the vagueness measure (which appeared in an earlier work) and an application for this approach. If the vagueness measure is computed for the distribution function of a given population, the value obtained gives a similar characteristic as the standard deviation of the population. Based on this property, a new global thresholding algorithm was developed that generalizes the idea of Otsu's optimality criterion by the means of continuous-valued logic. The performance of this method is compared with other commonly used algorithms to validate the usefulness of the proposed approach. Although the purpose of this algorithm is to threshold a grayscale image (which can be a useful step in the segmentation process of biological and medical images), it can be generalized for other tasks that require the separation of two or more populations, characterized by real values.

Keywords: Image segmentation; Global thresholding; Fuzziness measure; Pliant system; Vagueness functions; Vagueness measure; Ignorance Functions; Weak Ignorance Functions

1 Introduction

In the field of fuzzy sets, the fuzziness measures are used to reflect the uncertainty of the membership functions. The vagueness measure [9] was derived from fuzziness measures and it is a part of the Pliant system [11] which is a coherent continuous-valued logical system. A vagueness measure can be constructed by means of vagueness functions, which are closely related to entropy functions like Shannon's entropy function (as we showed earlier). In this paper we present a new interpretation of the vagueness measure. When applied to a distribution function, the result value gives a similar characteristic as the standard deviation of the given population.

In the field of image processing, segmentation is one of the most important tasks. The goal of segmentation is to divide the image into disjoint regions so that these

*6720, Szeged, Árpád tér 2, Hungary, University of Szeged, E-mail: dombi@inf.u-szeged.hu
†University of Szeged, E-mail: gulyasg@inf.u-szeged.hu

classes represent different objects. A frequently used technique for segmentation is the global thresholding, whose method is very useful in biological and medical image processing. Generally, the task requires to choose an appropriate threshold value for a given property and this value divides the pixels of the image into two classes based on the property: the pixels whose property value are below of the threshold constitute the first class, while the others belong to the second class. In many cases the property is the grayscale value of the pixels, but it can be other feature value which were computed for each pixels. Many algorithms have been introduced in the literature that can choose a good threshold value [15, 18, 23, 24, 26, 29, 33]. Some of these methods are based on fuzzy set theory and are able to handle any ambiguity of data [3, 4, 6, 7, 13, 14, 27, 28].

In this paper we will present the vagueness measure in a new context because we apply it on distribution functions. We will show that this interpretation is related to the standard deviation. Based on this property, we give a new application of the vagueness measure in a global thresholding algorithm which generalizes the idea of Otsu's method. The proposed approach will be compared with other well-known algorithms in order to show the effectiveness of the algorithm.

2 Preliminaries

Let f be a 2-dimensional discrete image function which is defined in such a way that

$$f\colon \{0,\dots,M-1\}\times\{0,\dots,N-1\}\to\{0,\dots,L-1\},$$

where L denotes the maximum intensity level. Let h denote the histogram (function) of the image which gives the number of occurrences at a given gray level:

$$h(q)=|\{(x,y): f(x,y)=q \text{ and } (x,y) \text{ is a pixel}\}|,$$

where $|.|$ is the cardinality of a given set. The normalized histogram p gives the probability of occurrence of a given gray level:

$$p(q)=P(\text{intensity}=q)=\frac{h(q)}{\sum_{i=0}^{L-1}h(i)}$$

For a given threshold value t, we can define the a priori probabilities of the background and object:

$$p_B^{(t)}=\sum_{q=0}^{t}p(q) \qquad p_O^{(t)}=\sum_{q=t+1}^{L-1}p(q)$$

As we mentioned earlier, which part of the intensities belongs to the object depends on the image, but for simplicity we will take the object to be the brighter part of the image. Let c denote the cumulative distribution function such that

$$c(q) = P(\text{intensity} \leq q) = \sum_{i=0}^{q} p(i).$$

Consider the cumulative distribution functions of the background and object for a given threshold t:

$$c_B^{(t)}(q) = \frac{\sum_{i=0}^{q} h(i)}{\sum_{i=0}^{t} h(i)} \qquad q = 0, \ldots, t$$

$$c_O^{(t)}(q) = \frac{\sum_{i=t+1}^{q} h(i)}{\sum_{i=t+1}^{L-1} h(i)} \qquad q = t+1, \ldots, L-1$$

We can assign a binary image to a given grayscale image f and a given threshold intensity t in the following way:

$$f_t(x,y) = \begin{cases} 1 & \text{if } f(x,y) \leq t \\ 0 & \text{otherwise} \end{cases}$$

In general, a *global thresholding algorithm* is one that determines a single threshold value and thresholds the entire image with that value.

3 Vagueness functions and vagueness measure

In Fuzzy Theory [31, 32] the membership function (denoted by μ) can be regarded as the approximation of the characteristic function belonging to crisp sets. Then, a fuzziness measure expresses the distance between the characteristic function and a given membership function, and reflects the uncertainty of the membership function. In the following sections we introduce the vagueness functions and the vagueness measure [9] which belong to the Pliant system (which will be also introduced). The vagueness measure can be interpreted as a fuzziness measure in the Pliant system but we give a new interpretation of the vagueness measure on distribution functions (when it does not play the role of a fuzziness measure).

3.1 Pliant system

The Pliant system [8, 10, 11] is a subclass of continuous valued logic where the fuzzy logical operators like negation, conjunction and disjunction, and the fuzziness measure constitute a coherent system. In the Pliant logic each operator is defined by one generator function. In fuzzy logic, several different generator functions are used. One of the main features of the Pliant concept that consists of infinitely many negations [8]. In the following we shortly introduce the definition of the Pliant system and the necessary notions from fuzzy logic.

Definition 1. *We say that $n(x)$ is a negation if n: $[0,1] \to [0,1]$ satisfies the following conditions:*

C1:	$n : [0,1] \to [0,1]$ *is continuous*	*(Continuity)*
C2:	$n(0) = 1,\ n(1) = 0$	*(Boundary conditions)*
C3:	$n(x) < n(y)$ *for* $x > y$	*(Monotonicity)*
C4:	$n(n(x)) = x$	*(Involution)*

Using the general representation theorem we have for the strict t-norm (conjunctive operator) and strict t-conorm (disjunctive operator) the following definitions.

Definition 2. *Let $f_c(x) : [0,1] \to [0,\infty]$ be continuous and strictly decreasing monotone generator function. Then*

$$c(x,y) = f_c^{-1}(f_c(x) + f_c(y)). \tag{1}$$

is a strict t-norm (conjunctive operator).

Definition 3. *Let $f_d(x) : [0,1] \to [0,\infty]$ be continuous and strictly increasing monotone generator function. Then*

$$d(x,y) = f_d^{-1}(f_d(x) + f_d(y)). \tag{2}$$

is a strict t-conorm (disjunctive operator).

Note: In Pliant logic, c stands for the conjunctive operator and d stands for the disjunctive operator. Those familiar with fuzzy logic theory will find that the terminology used here is slightly different from that used in standard texts [1, 2, 5, 12, 17, 21].

Definition 4. *If $f_c(x)$ and $f_d(x)$ are related in an inverse manner, i.e.*

$$f_c(x) f_d(x) = 1, \tag{3}$$

then we will call the generated connectives a Pliant system. It has been shown that only this system of strict t-norms and t-conorms is equipped with infinitely many negations [8].

Definition 5. *The general form of the multiplicative Pliant system is*

$$o_\alpha(x,y) = f^{-1}\left((f^\alpha(x) + f^\alpha(y))^{1/\alpha}\right) \tag{4}$$

where $f : [0,1] \to [0,\infty]$ is a continuous and strictly decreasing function which is the generator function of the strict t-norm operator.

$$\begin{array}{llll} \textit{If} & \alpha > 0, & \textit{than} & o_\alpha(x,y) \quad \textit{conjunctive operator (t-norm).} \\ \textit{If} & \alpha < 0, & \textit{than} & o_\alpha(x,y) \quad \textit{disjunctive operator (t-conorm).} \end{array} \tag{5}$$

The corresponding negation is:

$$\eta_\nu(x) = f^{-1}\left(f(\nu_0)\frac{f(\nu)}{f(x)}\right) \qquad or \qquad \eta_{\nu_*}(x) = f^{-1}\left(\frac{f^2(\nu_*)}{f(x)}\right) \tag{6}$$

We have to note that η_ν and η_{ν_*} are well defined and satisfy the conditions of Definition 1 (see in [8]), thus the negations can be interpreted when $f(x) = 0$. The parameter ν, like ν_* is the neutral value of the negation and can be interpreted as the strictness of the negation.

Because the generator function is determined up to a multiplicative constant, we can arrange it such that $f(\nu_0) = 1$ and so $\eta_\nu(x) = f^{-1}(f(\nu)/f(x))$. If $f(\nu_0) = f(\nu) = 1$, then we get the following.

Definition 6. *Let $f(x) : [0,1] \to [0,\infty]$ be continuous and strictly decreasing monotone function. Then*

$$\eta(x) = f^{-1}\left(\frac{1}{f(x)}\right) \tag{7}$$

is the standard Pliant negation function.

In the Pliant system the conjunctive and disjunctive operators exist in weighted forms which are defined by the following formulas:

$$c(\underline{\mathbf{w}}, \underline{\mathbf{x}}) = f^{-1}\left(\sum_{i=1}^{n} w_i f(x_i)\right) \quad \text{and} \quad d(\underline{\mathbf{w}}, \underline{\mathbf{x}}) = f^{-1}\left(\frac{1}{\sum_{i=1}^{n} \frac{w_i}{f(x_i)}}\right). \tag{8}$$

Thus, we can define a bivariate conjunction operator where the weights are equal, and what we call mean conjunction operator.

Definition 7. *Let $\bar{c}(x,y)$ denote the mean conjunction operator in bivariate case:*

$$\bar{c}(x,y) = f^{-1}\left(\frac{1}{2}(f(x) + f(y))\right) \tag{9}$$

where $f\colon [0,1] \to [0,\infty)$ is a continuous, strictly decreasing generator function such that $f(1) = 0$ ([8]).

3.2 Vagueness functions and measure

The idea and construction of a vagueness measure [9] can be derived from the fuzziness measure. Now we will introduce the vagueness measure as a part of the Pliant system.

Definition 8. *The vagueness function $v : [0,1] \to [0,1]$ in the Pliant system is*

$$v(x) = \bar{c}(x, \eta(x)) = f^{-1}\left(\frac{1}{2}\left(f(x) + \frac{1}{f(x)}\right)\right) \tag{10}$$

Definition 9. *The normalized vagueness function is*

$$v^*(x) = \frac{1}{v(\nu_0)} v(x), \tag{11}$$

where ν_0 is the fix point of the negation η such that $\eta(\nu_0) = \nu_0$.

Now we will give a list of properties for the vagueness function:

(P1):	$v(x) = 0$ iff $x \in \{0,1\}$	(Sharpness: no vagueness)
(P2):	$v(x)/v(\nu_0) = 1$ iff $x = \nu_0$	(Maximality: maximal vagueness)
(P3):	$v(x_1) < v(x_2)$ if	(Monotonicity)
	$x_1 < x_2$ and $x_1 \leq \nu_0$ or	
	$x_1 > x_2$ and $x_1 \geq \nu_0$	
(P4):	$v(x) = v(\eta(x))$	(Symmetry)

We can get a very simple form of the vagueness function if we use the generator function of the Dombi operator [8], namely $f(x) = \left(\frac{1-x}{x}\right)^{\alpha}$. First, we get the following formula:

$$v_{\alpha}(x) = \frac{1}{1 + \left(\frac{1}{2}\left(\frac{1-x}{x}\right)^{\alpha} + \frac{1}{2}\left(\frac{x}{1-x}\right)^{\alpha}\right)^{\frac{1}{\alpha}}}. \tag{12}$$

In the Dombi operator case the negation η in the Pliant system is the standard negation $\eta(x) = 1 - x$ (based on Definition 6) and therefore $\nu_0 = \frac{1}{2}$ and $v_{\alpha}(\nu_0) = \frac{1}{2}$, and we get the following.

Definition 10. *The normalized vagueness function in the Dombi case is*

$$v^*_{\alpha}(x) = \frac{1}{v_{\alpha}(\nu_0)} v_{\alpha}(x) = 2v_{\alpha}(x) = \frac{2}{1 + \left(\frac{1}{2}\left(\frac{1-x}{x}\right)^{\alpha} + \frac{1}{2}\left(\frac{x}{1-x}\right)^{\alpha}\right)^{\frac{1}{\alpha}}}. \tag{13}$$

Let $\alpha = 1$. Then the vagueness function and the normalized vagueness function have the following simple forms:

$$v_1(x) = 2x(1-x) \qquad \text{or} \qquad v^*_1(x) = 4x(1-x) \tag{14}$$

While we got the idea of the vagueness measure from the fuzziness measures, it can be interpreted in a more general way than a measure of fuzziness. Thus, we give a definition which works on a finite set of values from $[0,1]$.

Definition 11. *Let $\underline{x} = \{x_1, x_2, \ldots, x_n\}$ where $x_i \in [0,1]$, and let v be a vagueness function. Then*

$$\mathcal{V}(\underline{x}) = \frac{1}{n} \sum_{i=1}^{n} v(x_i) \tag{15}$$

is a vagueness measure defined by v.

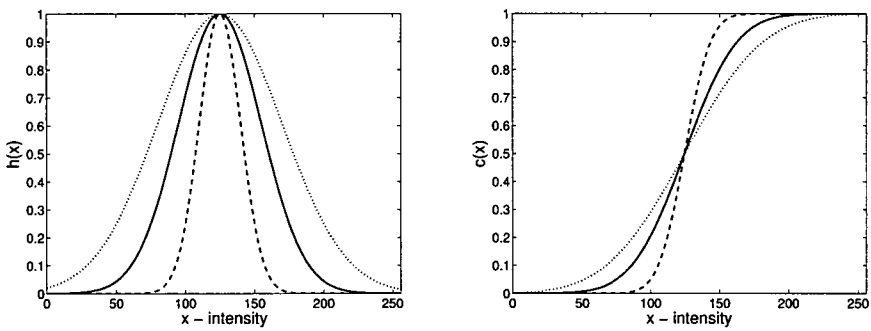
Definition 12. *Let $\mathcal{V}^*_{\alpha}$ denote the vagueness measure defined by v^*_{α}, where v^*_{α} is the normalized vagueness function in the Dombi case.*

3.3 Relationship between the vagueness measure and the standard deviation

As we showed earlier in [9], the vagueness measure can be interpreted as a fuzziness measure. In this subsection we provide a small example which shows that if the

vagueness measure is applied to a distribution function, it is closely related to the variance of the given population.

In our example, three histograms were generated using three Gaussians with the same mean and different standard deviations (the discrete histogram functions were generated by a simple sampling process). The cumulative distribution functions were also calculated, all functions can be seen in Figure 1. The dashed, the solid and the dotted line histograms were created using 15, 30 and 45 as standard deviation values, respectively. The corresponding vagueness values are 16.92, 33.85 and 49.96 (we used $\mathcal{V}_1^*$ as the vagueness measure), which are related to the former standard deviations. A short explanation is the following. The less the standard deviation, the sharper the distribution function is (see in in Figure 1), and more values are closer to 0 or 1, which leads a smaller vagueness measure value. Table 1 contains more vagueness measure values calculated for the example distribution functions using different α values as measure parameters.



(a) Histograms (b) Distribution functions

Figure 1: Example histograms and cumulative distribution functions. The dashed, the solid and the dotted line histograms were created using Gaussians with the standard deviation values 15, 30 and 45, respectively.

3.4 Relationship between the vagueness and weak ignorance functions

We have to mention the similarity between the vagueness functions and weak ignorance functions which were proposed in [25] and have almost the same properties, although they have different constructions. Weak ignorance functions can be defined based on ignorance functions [4].

Definition 13. *A continuous mapping* $G\colon [0,1]^2 \to [0,1]$ *is an ignorance function such that:*

α \ σ	15.000	30.000	45.000
0.125	32.658	65.218	90.927
0.250	25.777	51.540	74.609
0.500	20.551	41.100	60.360
1.000	16.923	33.846	49.958
2.000	14.658	29.317	43.350
4.000	13.371	26.743	39.570
8.000	12.684	25.369	37.546

Table 1: Relationship between the vagueness measure and the standard deviation. The table contains the vagueness measure values computed for the example distribution functions (with different standard deviation values, appeared in the first row) where the measure has different α values (appeared in the first column).

(G1) $G(x,y) = G(y,x)\ \forall x,y \in [0,1]$
(G2) $G(x,y) = 0$ *iff* $x = 1$ *or* $y = 1$
(G3) $G(0.5, 0.5) = 1$
(G4) G *is decreasing in* $[0.5,1]^2$
(G5) G *is increasing in* $[0,0.5]^2$

Theorem 1. *Let* $G\colon [0,1]^2 \to [0,1]$ *be an ignorance function. The continuous function* $g\colon [0,1] \to [0,1]$ *given by*

$$g(x) = G(x, 1-x)$$

is a weak ignorance function which satisfies:

(g1) $g(x) = g(1-x)\ \forall x \in [0,1]$
(g2) $g(x,y) = 0$ *iff* $x \in \{0,1\}$
(g3) $g(0.5) = 1$

The similarities between the vagueness and weak ignorance functions are obvious. The $(g2)$ property is the same as the $(P1)$ property. In the case of the $(g1)$ property, it can be seen that the vagueness function is more general because it expresses the $(P4)$ symmetry using a general negation, while the standard strong negation $N(x) = 1 - x$ appears in the $(g1)$ property. The $(P2)$ property states that a normalized vagueness function gets its maximum $(= 1)$ at the fixpoint of its negation. If we suppose again that in $(g1)$ the strong negation appears with the fixpoint 0.5, then $(g3)$ tells us that at this fixpoint g has its maximum value. It is the same as in $(P2)$ if we restrict it to the strong negation. The monotonicity is not required for weak ignorance functions explicitly.

4 Thresholding based on vagueness measure

Now we present a new global thresholding algorithm which uses the vagueness measure and generalizes the idea of Otsu's method [22]. Although this algorithm works on a grayscale image it can be generalized for the case of real values because it only needs the distribution functions of the populations. First, consider the two vectors of the cumulative distribution function values of the background and object for a given threshold t:

$$\underline{\mathbf{C}}_B^{(t)} = \{c_B^{(t)}(0), c_B^{(t)}(1), \ldots, c_B^{(t)}(t)\}$$
$$\underline{\mathbf{C}}_O^{(t)} = \{c_O^{(t)}(t+1), c_O^{(t)}(t+2), \ldots, c_O^{(t)}(L-1)\}$$

Similarly to Otsu's thresholding criterion [22] which minimizes the intra-class variance, we construct a new method which minimizes the joint vagueness measure of the background and the object in order to yield two well separated population of intensities.

Definition 14. *Let $\underline{\mathbf{C}}_B^{(t)}$ and $\underline{\mathbf{C}}_O^{(t)}$ the vectors of the cumulative distribution function values of the background and object for a given threshold t. Then the joint vagueness measure belonging to these vectors and the threshold t is defined by the following formula:*

$$p_B^{(t)} \cdot \mathcal{V}_{\alpha_1}^* \left(\underline{\mathbf{C}}_B^{(t)}\right) + p_O^{(t)} \cdot \mathcal{V}_{\alpha_2}^* \left(\underline{\mathbf{C}}_O^{(t)}\right). \tag{16}$$

With the notations above we can express the new thresholding criterion in the terms of the following optimization problem:

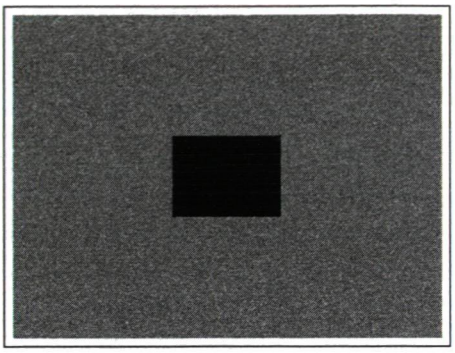
$$t^* = argmin_t \left(p_B^{(t)} \cdot \mathcal{V}_{\alpha_1}^* \left(\underline{\mathbf{C}}_B^{(t)}\right) + p_O^{(t)} \cdot \mathcal{V}_{\alpha_2}^* \left(\underline{\mathbf{C}}_O^{(t)}\right)\right), \tag{17}$$

where t^* is the best threshold corresponding to the minimum joint vagueness measure. The apriori probabilities of the background and the object balance the effect of the vagueness measures if the two population do not have the same size. The pseudo-code of the algorithm can be seen below (Algorithm 1: Pliant Thresholding Algorithm (PTA)). The method requires the histogram h of the image and the parameters of the vagueness measures (α_1, α_2) as the input. The first step can be carried out by using only the histogram. In the second step the calculated values must be used and the vagueness values. The algorithm chooses the threshold t^* which belongs to the lowest joint vagueness measure.

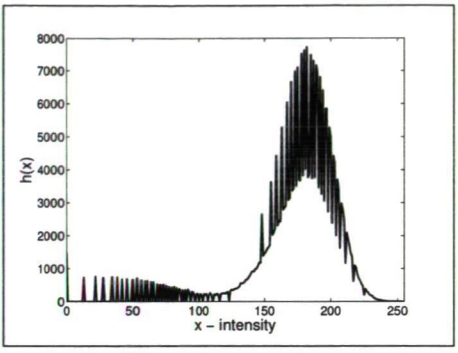
An example image and the running results of the PTA can be seen in Figure 2. Figure 3 contains an example where the PTA was applied on the same image using different values of α_1, α_2.

5 Experimental results

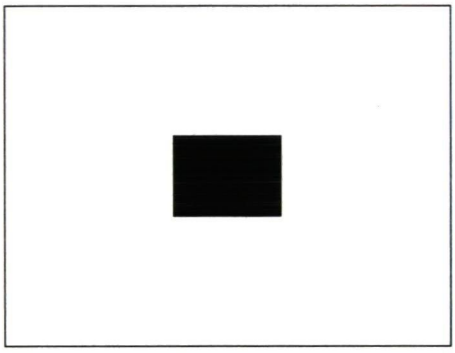
In order to validate the performance of the PTA, it was applied to a set of synthetic images and to a set of standard images (which are widely used in the literature).



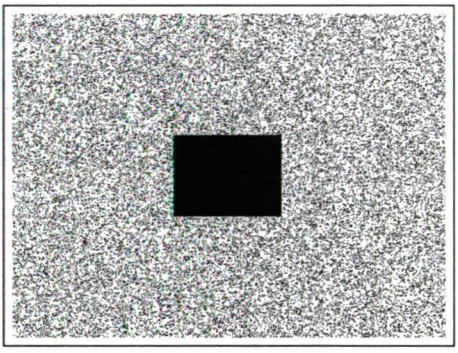
(a) Original image



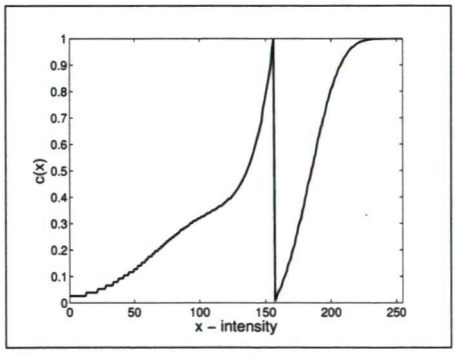
(b) Histogram of the image



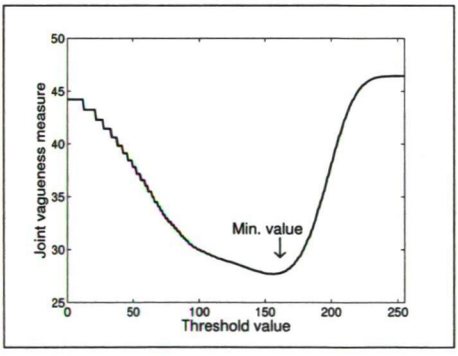
(c) Optimal segmentation



(d) The thresholded image using the value 156



(e) The distribution functions belonging to the threshold value 156



(f) Joint vagueness measure as the function of the threshold value

Figure 2: An example image and the results of the PTA. The algorithm chose the threshold value 156.

Algorithm 1 Pliant Thresholding Algorithm (PTA)

Require: histogram h, parameters of the two vagueness measures α_1, α_2

(a) compute $p_B^{(t)}, p_O^{(t)}, c_B^{(t)}, c_O^{(t)}$ for all $t \in [0, L-1]$

(b) compute joint vagueness measures for all $t \in [0, L-1]$

(i) compute value $\mathcal{V}^*_{\alpha_1}\left(\underline{\mathbf{C}}_B^{(t)}\right)$

(ii) compute value $\mathcal{V}^*_{\alpha_2}\left(\underline{\mathbf{C}}_O^{(t)}\right)$

(iii) compute the joint vagueness measure

(c) take t^* as the best threshold corresponding to the minimum joint vagueness measure

Figure 3: Applying of the PTA using different sets of (α_1, α_2): $(0.0625, 16.0)$, $(0.125, 8.00)$, $(0.25, 4.00)$, $(0.50, 2.00)$, $(1.00, 1.00)$, $(2.00, 0.50)$, $(4.00, 0.25)$, $(8.00, 0.125)$, $(16.00, 0.0625)$, from left to right, top to bottom.

We compared the PTA (using the parameters $\alpha_1 = \alpha_2 = \frac{1}{2}$) with some other well-known and commonly used thresholding methods, namely Otsu's method [22], Huang-Wang algorithm [13], Kittler's method [16], Li's method [20], and Area algorithm [3] (with $\phi_1(x) = \phi_2(x) = x$). There are many approaches available for comparing thresholding algorithms in the literature [19, 24, 26, 33]. Here we have applied the misclassification error [30] as the performance criterion which is the following.

Definition 15. *Let B_O and F_O denote the known background and object pixel sets belonging to the optimal segmentation, and let B_T and F_T be the background and object area pixels in the result image thresholding by t, respectively. Then the misclassification error which belongs to a given threshold t is computed by the following formula:*

$$ME = 1 - \frac{|B_O \cap B_T| + |F_O \cap F_T|}{|B_O| + |F_O|}. \tag{18}$$

5.1 Comparison on the sets of synthetic images

Now, we introduce the creating process of the synthetic images which were used in the first evaluation. The first (basic) image set was created as follows. An image consists of a square on the center of the image which is the object and the other surrounding pixels compose the background (we do not need more complicated shapes because the algorithm works on the image's histogram). The object and the background were generated by two Gaussians (with different means but with not necessarily different standard deviations). The means were picked up from the set $\{0, 30, 50, 120, 150, 200\}$ while for the deviations we used the set $\{10, 20, 30\}$ (not all of the possible selections were generated). So we got a synthetic set with 250 images. We applied different distortions on the synthetic images in order to yield more test cases: we used additive Gaussian noise (with different standard deviations) and impulsive salt-and-pepper noise (with different amount of corrupted pixels). Each distortion with 4 different parameters were used on the original images so the total number of the images was $250 + 250 \cdot 2 \cdot 4 = 2250$. Some examples can be seen in Figure 4.

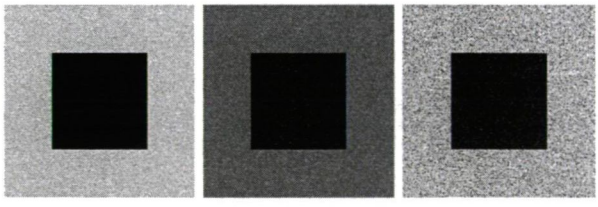


Figure 4: Synthetic image examples (from the left to the right): standard image without any distorsions, image with Gaussian noise, and image contaminated by salt-and-pepper noise.

The summarized results can be found in Table 2. The rows belong to different synthetic sets: the first contains the misclassification errors on the original images while the next four show the results on the images with additive Gaussian noise, and the last four give the values for the salt-and-pepper noise contaminated images. As it can be seen the PTA and the Area algorithm have almost the same error values (small differences occur in the the cases of the salt-and-pepper noise contaminated images) and both perform better than the other studied methods.

Type	Otsu	Huang-W.	Kittler	Li	Area	PTA ($\alpha = 0.5$)
B	0.049	0.049	0.071	0.053	0.045	0.045
G2	0.049	0.049	0.080	0.054	0.045	0.045
G5	0.051	0.055	0.084	0.056	0.048	0.048
G10	0.058	0.068	0.113	0.065	0.059	0.059
G15	0.070	0.093	0.164	0.079	0.076	0.076
SP1	0.054	0.052	0.077	0.058	0.049	0.048
SP3	0.071	0.061	0.078	0.072	0.058	0.056
SP5	0.099	0.072	0.089	0.090	0.069	0.065
SP10	0.124	0.098	0.109	0.136	0.096	0.108

Table 2: Misclassification errors on synthetic images. Every row belongs to a set of synthetic images: B means the original synthetic images, G denotes the images with additive Gaussian noise where the number is the standard deviation σ, and SP refers to the salt-pepper noise contaminated images where the number gives the percentage of the modified pixels.

5.2 Comparison on the set of standard images

The second evaluation of the algorithm was investigated by using 20 classical and bimodal test images which are taken from a collection of Carnegie Mellon University (http://www.cs.cmu.edu/~cil/v-images.html). The images can be seen in Figure 5 while the ground-truth images appear in Figure 6 where the ground-truth thresholds were set manually.

The missclassification error values are presented in Table 3 below. These values are the average measures calculated for the 20 test images. In this case, the PTA gives a slightly worse result than Otsu, but Li's algorithm and the Area perform very similarly. We should mention here that the measure ME suffers from the subjectivity of the human expert or observer who sets the ground-truth thresholds, so these differencies could be regardarded as insignificant. However we have chosen the ground-truth threshold instead of creating a gold standard mask because we compare global thresholding algorithms, and we think in this case that the ground-truth threshold value gives the technical maximum for these algorithms at least from ME aspect.

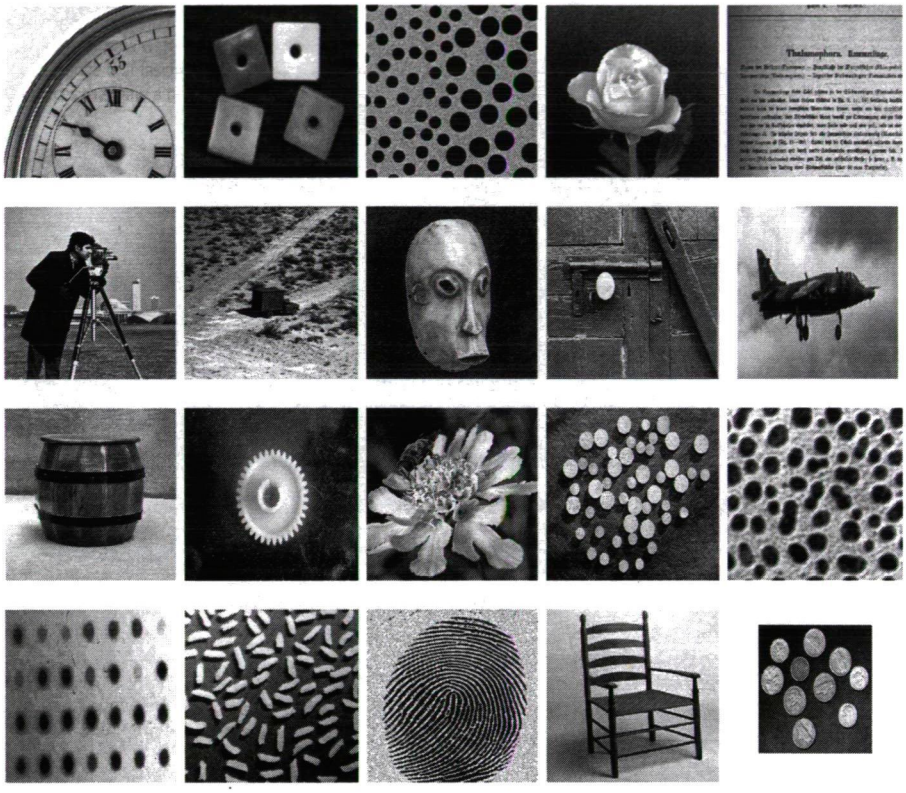


Figure 5: Original images

Images	Otsu	Huang-W.	Kittler	Li	Area	PTA ($\alpha = 0.5$)
Standard	0.085	0.135	0.185	0.101	0.106	0.095

Table 3: Misclassification errors on standard images.

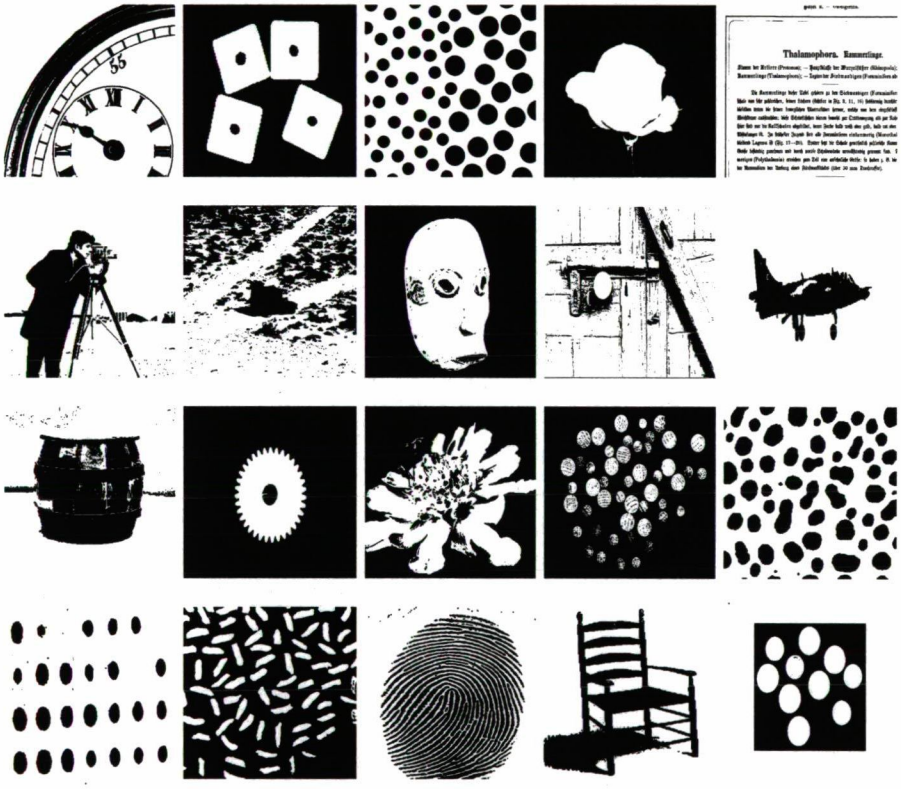


Figure 6: Ground-truth images

6 Conclusions

In this study we provided a new interpretation of the vagueness measure which assigns a value to a distribution function that is similar to the population's standard deviation. Then we gave an application using this property (in the form of a global thresholding algorithm). The proposed method performs as well as or better than the classical and state-of-the-arts methods. One advantage of this approach is that the algorithm can be parameterized, hence it can be adapted to a given problem. In the future, we would like to try other conjunction operators taken from the Pliant system (e.g. a weighted conjunction) that have more parameters. It is an interesting question of how the parameters should be optimized such that it fits the given problem. Anyway, it would be useful to know what classes of problems the vagueness measure can be applied to.

7 Acknowledgments

This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TÁMOP-4.2.2.C-11/1/KONV-2012-0013).

References

[1] Alsina, C., Schweizer, B., and Frank, M.J. *Associative functions: triangular norms and copulas.* Word Scientific Publishing, 2006.

[2] Beliakov, G., Pradera, A., and Calvo, T. *Aggregation Functions: A Guide for Practitioners.* Studies in Fuzziness and Soft Computing. Vol. 221., Springer, 2007.

[3] Bustince, H., Barrenechea, E., and Pagola, M. Image thresholding using restricted equivalence functions and maximizing the measures of similarity. *Fuzzy Sets and Systems*, 158(5):496–516, 2007.

[4] Bustince, H., Pagola, M., Barrenechea, E., Fernandez, J., Melo-Pinto, P., Couto, P., Tizhoosh, HR, and Montero, J. Ignorance functions. An application to the calculation of the threshold in prostate ultrasound images. *Fuzzy Sets and Systems*, 161(1):20–36, 2010.

[5] Calvo, T., Mayor, G., and Mesiar, R., editors. *Aggregation Operators.*, volume 97 of *New Trends and Applications Studies in Fuzziness and Soft Computing.* Physica-Verlag, Heidelberg, 2002.

[6] Chaira, T. and Ray, AK. Segmentation using fuzzy divergence. *Pattern Recognition Letters*, 24(12):1837–1844, 2003.

[7] Chaira, T. and Ray, AK. Threshold selection using fuzzy set theory. *Pattern Recognition Letters*, 25(8):865–874, 2004.

[8] Dombi, J. Demorgan systems with an infinitely many negations in the strict monotone operator case. *Information Sciences*, 181:1440–1453, 2011.

[9] Dombi, J. Fuzziness measure in the pliant system: The vagueness measure. *Acta Technica Jaurinensis*, 4(1), 2011.

[10] Dombi, J. On certain class of aggregation operator. *Information Sciences, Under review process*, 2011.

[11] Dombi, J. Pliant operator system. *Recent Advances in Intelligent Engineering Systems, Studies in Computational Intelligence, Springer*, 378:31–58, 2012, under print.

[12] Grabisch, M., Marichal, J.-L., Mesiar, R., and Pap, E. *Aggregation Functions.* Encyclopedia of Mathematics and Its Applications 127, Cambridge University Press, Cambridge, 2009.

[13] Huang, L.K. and Wang, M.J.J. Image thresholding by minimizing the measures of fuzziness. *Pattern Recognition*, 28(1):41–51, 1995.

[14] Jawahar, CV, Biswas, PK, and Ray, AK. Investigations on fuzzy thresholding based on fuzzy clustering. *Pattern Recognition*, 30(10):1605–1613, 1997.

[15] Kapur, JN, Sahoo, PK, and Wong, AKC. A new method for gray-level picture thresholding using the entropy of the histogram. *Computer Vision, Graphics, and Image Processing*, 29(3):273–285, 1985.

[16] Kittler, J. and Illingworth, J. Minimum error thresholding. *Pattern Recognition*, 19(1):41–47, 1986.

[17] Klement, E.P., Mesiar, R., and Pap, E. *Triangular norms.* Dordrecht:Kluwer, 2000.

[18] Lee, S.U., Yoon Chung, S., and Park, R.H. A comparative performance study of several global thresholding techniques for segmentation. *Computer Vision, Graphics, and Image Processing*, 52(2):171–190, 1990.

[19] Levine, M.D. and Nazif, A.M. Dynamic measurement of computer generated image segmentations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):155–164, 2009.

[20] Li, CH and Tam, PKS. An iterative algorithm for minimum cross entropy thresholding. *Pattern Recognition Letters*, 19(8):771–776, 1998.

[21] Mesiar, R., Kolesárová, A., Calvo, T., and Komornková, M. *A Review of Aggregation Functions.* Fuzzy Sets and Their Extension: Representation, Aggregation and Models. Studies in Fuzziness and Soft Computing, Vol. 220, 2008.

[22] Otsu, N. A threshold selection method from gray-level histograms. *Automatica*, 11:285–296, 1975.

[23] Ridler, TW and Calvard, S. Picture thresholding using an iterative selection method. *Systems, Man and Cybernetics, IEEE Transactions on*, 8(8):630–632, 2007.

[24] Sahoo, PK, Soltani, S., and Wong, AKC. A survey of thresholding techniques. *Computer Vision, Graphics, and Image Processing*, 41(2):233–260, 1988.

[25] Sanz, Josean, Fernndez, Alberto, Sola, Humberto Bustince, and Herrera, Francisco. A genetic tuning to improve the performance of fuzzy rule-based classification systems with interval-valued fuzzy sets: Degree of ignorance and lateral position. *Int. J. Approx. Reasoning*, pages 751–766, 2011.

[26] Sezgin, M. and Sankur, B. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–168, 2004.

[27] Tizhoosh, H.R. Image thresholding using type II fuzzy sets. *Pattern Recognition*, 38(12):2363–2372, 2005.

[28] Wang, Q., Chi, Z., and Zhao, R. Image thresholding by maximizing the index of nonfuzziness of the 2-D grayscale histogram. *Computer Vision and Image Understanding*, 85(2):100–116, 2002.

[29] Weszka, J.S. A survey of threshold selection techniques. *Computer Graphics and Image Processing*, 7(2):259–265, 1978.

[30] Yasnoff, W.A., Mui, J.K., and Bacus, J.W. Error measures for scene segmentation. *Pattern Recognition*, 9(4):217–231, 1977.

[31] Zadeh, L.A. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[32] Zadeh, L.A. *Fuzzy sets and their applications to cognitive and decision processes.* Academic Press, New York, 1975.

[33] Zhang, Y.J. A survey on evaluation methods for image segmentation. *Pattern Recognition*, 29(8):1335–1346, 1996.

Acta Cybernetica 21 (2013) 123–134.

Elimination of the Background of Electron Microscope Images by Using FPGA

Ádám Fazekas*, Hiroshi Daimon†, Hiroyuki Matsuda†, and László Tóth*

Abstract

The purpose of our development is to design an FPGA based hardware acceleration system that is able to be used for analyzing photoemission electron microscope (PEEM) images or improving their quality. Even though a usual PEEM has an energy filter unit, which is able to eliminate certain disturbing signals, a post processing computation can also be useful to improve the image quality. Here we propose an FPGA based hardware acceleration system for the computation of a certain image background component. It has uniquely designed hardware modules that perform the computations in parallel, resulting in less calculation time. The system shown here is a prototype which was only used for testing and experimental purposes.

Keywords: photoelectron spectra, Shirley background, field-programmable gate array, hardware acceleration

1 Introduction

Due to the technological advancement there is an increasing demand for observing processes which take place in micro- and nano-scale ranges. For this purpose among others, photoemission electron microscopy (PEEM) provides a solution. It gives photoelectron spectra from individual small areas and has a wide range of applications in many branches of science, such as physical, chemical and biological research, nanotechnology, semiconductor design and manufacturing. However the images that they produce could contain certain disturbing signals which completely obscure the useful information, i.e. a photoelectron peak contains large background originated from higher energy peaks. There are several ways to eliminate this background component and one of them is to apply post-process computations. This method can be a computationally intensive task because the background should be calculated for each pixel, so it is appropriate to use hardware acceleration to

*University of Debrecen, Faculty of Informatics, Department of Informatics Systems and Networks, Kassai út 26 Debrecen, 4028 Hungary. E-mail: adam.j.fazekas@gmail.com

†Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, NARA 630-0192 JAPAN

reduce the execution time. Field-programmable gate arrays (FPGA) [2] provide an excellent opportunity for the design of such systems. The FPGA architecture offers massive parallel capabilities and uniquely customizable hardware components designed to carry out the given task in the most efficient way. In this paper we propose a prototype of hardware acceleration system for computing a background component on FPGA platform. The system was implemented and tested on an Altera DE2 Development and Education FPGA board [1]. The design performs the calculations in parallel with specialized hardware components; therefore, it takes less time to complete than it would take with an ordinary computer.

2 Physical background

2.1 About the Photoelectron Emission Microscope

This work is related to a new display-type ellipsoidal mesh analyzer (DELMA) [6, 14], with a new type 1π sr wide acceptance angle spherical aberration corrected electrostatic lens (WAAEL) [5, 9, 7, 8, 15]. This special photoemission electron microscope is able to be used for simultaneous angular and energy distribution measurements, electron spectroscopy and spectrography, diffraction and holographic measurements. Furthermore, due to the extremely large acceptance angle it can be used for stereo photoemission electron microscopy (Stereo-PEEM) to obtain three-dimensional atomic and electronic structures of microscopic-materials.

2.2 The examined background component

There are several solutions for the correction of chromatic and spherical aberration where i.e. one of the unique is operating by applying a time dependent electric field [11]. The objective lens that was applied in our case corrects the spherical aberration only by applying a quasi-ellipsoidal shape mesh lens inside [5]. One of the advantages of this type of lens is that the sample area is field free. Furthermore, it has no pass energy limit; therefore, it can be applied in wide energy ranges. However, due to the wide acceptance angle it requires careful design and construction. The quality of the measured images can be improved if we can distinguish the background and then subtract it from the image. This can be achieved by taking images at many pass-energies, where each pixel of the image behaves in a way that is known, but differently from the background components (Fig. 1) [15]. In this paper we deal with the elimination of the Shirley-background component [12] by using an FPGA processor. As a first step we have examined this method for image processing purposes and tested it on a low cost FPGA device.

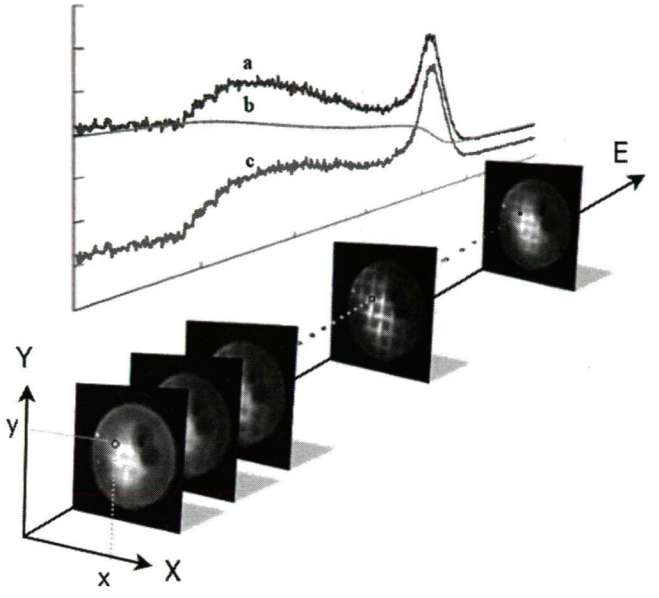


Figure 1: The spectral image sequence (bottom) and the corresponding intensity distribution for the calculations of the background subtracted images [15, 6]. Curve 'a' is the original intensity distribution among the energy axes (E) at given (x,y) coordinates on the images, where an elastic peak and a plasmon-loss peak are seen. Curve 'b' is the calculated Shirley-background. Curve 'c' is the intensity after the subtraction of the background.

3 The applied Hardware

3.1 A brief description of Field Programmable Gate Arrays

Field programmable gate arrays (FPGA) are integrated circuits that do not have specific functionality; therefore, one must program the device to make it able to perform the required task. The main components of an FPGA are the logic blocks that contain logic elements, programmable interconnect and input/output ports. In addition, almost every FPGA has further special components like embedded memory or multiplier circuits. By configuring or reconfiguring the logic elements and the programming interconnect between them the functionality is given to the device to perform the desired task. This flexibility allows the designer to implement various hardware designs using a hardware description language like Verilog or VHDL. The benefits of FPGAs are not only the flexibility but also the paralleling capabilities. The implemented hardware modules can be run parallel independently, which greatly improves the system efficiency. For these reasons FPGAs often provide higher performance than ordinary processors and digital signal processing devices [2].

3.2 The applied FPGA device

The background computing system was implemented on an Altera DE2 Development and Education FPGA board [1]. This device has a Cyclon II FPGA processor, some basic hardware peripherals such as external memories and several input output ports like RS-232. Although this device was not developed for high performance computations, it was perfectly suitable for testing and experimental purposes for the prototype system. For the design we used the Altera Quartus II web edition development software, and implemented the design in the Verilog hardware description language.

4 Implementation

4.1 Algorithm

To determine the background component (Fig. 2) we used the iterative Shirley method [12] and modified the algorithm structure for a feasible hardware design. The method was implemented in both hardware and software platforms for comparison reasons.

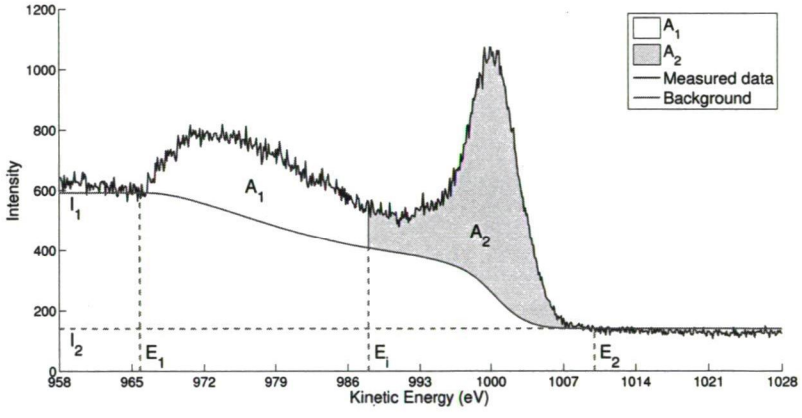


Figure 2: The Shirley background and parameters for its computation.

Our design divides the Shirley algorithm into two phases. The first phase computes the area between the points of the background (S_i) and the points of a spectrum (data) with rectangle approximation; where E_1 and E_2 are the two energy indexes, the background computation takes place between them, and its values are set by the user. ΔE is the energy difference (step size) between two consecutive points in the spectrum. Here is the pseudocode for the area computation ($A_{max} = A_1 + A_2$):

Algorithm 1 Area computation

1: **for** $k := E_2$ **downto** E_1 **do**
2: $A_{max} := A_{max} + (data(k) - S_i(k)) * \Delta E$
3: $A_2(k) := A_{max}$
4: **end for**

This part also computes the value of A_2 for every point because it is produced during the computation of A_{max}. The second phase computes the points of the Shirley background for the next iteration, where I_1 and I_2 are the intensities at E_1 and E_2 energies. Pseudocode for computing the Shirley background in the 'i'-th iteration:

Algorithm 2 Shirley computation

1: **for** $j := E_1$ **to** E_2 **do**
2: $S_i(j) := I_2 + (I_1 - I_2) * (A_2(j)/A_{max})$
3: **end for**

This separation is important since we could define unique arithmetical circuits for both parts that made the computation more efficient. In the following, the implementation will be explained in more details.

4.2 System architecture

The background computing system consists of two main components, an FPGA based computer unit and a Java application running on a personal computer. The Java application provides the measured data and the computation parameters for the FPGA through serial communication using the RS-232 communication standard. Also, this application gives user interface for the system, where the computation parameters such as E_1, E_2 and the step size ΔE can be set. The FPGA stores the incoming data in the external memory. After the transfer, the central controller unit starts serving the background computer hardware modules which operate in parallel. The computed background intensities are stored in the external memory too. When all the background intensities have been calculated, the central controller unit sends the results back to the Java application through the serial port. Figure 3 shows the schematic diagram of the system. There are subsystems for the different tasks such as the memory controller for the memory operations, the I/O controller for communication, and Shirley modules for computing the Shirley background. Each subsystem is controlled by the system controller which has the role of managing the serving strategy for the Shirley modules.

4.3 Shirley background computation with hardware modules

The values of the Shirley background are determined by specialized hardware modules. All of the Shirley modules have their own memory and arithmetical modules

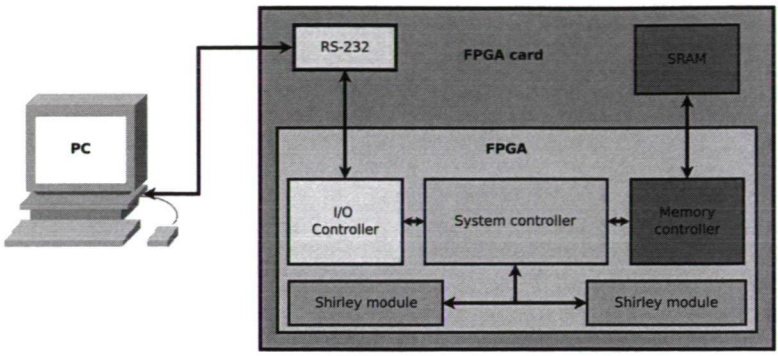


Figure 3: The system architecture.

so they could operate independently in parallel. The modules implement the iterative Shirley method for the background calculation. The current system operates with the IEEE-754 standard single precision floating point and the Q32.16 fixed point number formats [3]. We used real numbers to keep the computation precision however, considering only image processing purposes even integer arithmetic would be enough. In this case we can get much shorter running time and simpler circuit, but the results would not reflect the measured data precisely. The primary data format used for storing data in the memory and transferring it between the PC and the FPGA system is IEEE-754 standard single precision floating point. The arithmetical modules are optimized with a pipeline technique [10] that results in increased throughput and reduced running time. The Shirley modules consist of three major components; the arithmetical, the memory controller and the Shirley controller units. The memory controller unit performs the read and write operations on the dedicated memory which contains the required data for the computation of one spectrum. The controller unit manages the computation procedure implemented as a finite state machine [4]. The iterative Shirley method was divided into two parts as mentioned earlier. The area between the measured data and the flat background intensities is computed at the beginning of the process and the new approximations of the background intensities are calculated afterwards. The area processor unit (Fig. 4) determines the summarized area, A_{max}, in the first step of each iteration, while the A_2 values are saved for every point in a dedicated memory, so they will be simply read out when they are needed for the calculation. This part uses Q32.16 fixed point arithmetic for fast summation, thus conversion is required at the ingress and egress part of this circuit. After this, the Shirley processor unit (Fig. 4) determines the intensities of the points of the next background approximation. The Shirley computer module uses the IEEE-754 standard single precision floating point number format, because the division can be implemented more efficiently by this than the fixed point case. The reason behind the usage of different number formats is that the summations consume much more time and resources with floating point than with fixed point numbers, even if we consider the

time for conversion. Furthermore, the division with fixed-point number format has similar disadvantages compared to the case of floating-point. The whole iteration forms one pipeline circuit; therefore, the computation of the background intensities performed rapidly.

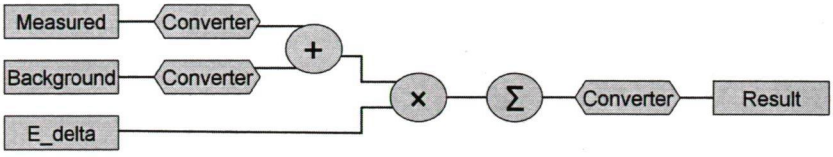


Figure 4: The block diagram of the area processor unit.

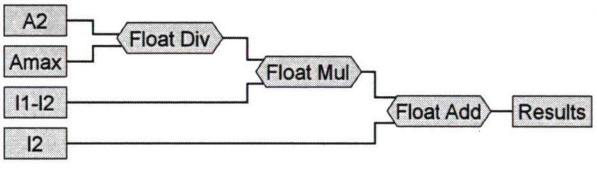


Figure 5: The block diagram of the Shirley processor unit.

The iteration process ends when the difference between two consecutive background approximations no longer exceeds a predefined constant value.

4.4 The final results

We have designed a hardware accelerated computation system that can distinguish certain components of electron microscope images originated from different physical processes, i.e. the disturbing backgrounds. The prototype of the system has been completed and it is able to determine and subtract certain background components. The magnified test images of a mesh sample (SUS316, #100) were taken by the DELMA at the beam-line BL07LSU of Spring-8 [6, 13] (Fig. 6). The sample in the presented work was irradiated by 1 keV energy and 250 μm diameter electron beam with 14° inclination to the sample surface. The system performance was tested on low magnification images with forced quality degradation by taking the images with fully opened apertures. The magnified (12×) images were intensified and converted into visible light with a microchannel plate (MCP) - phosphor screen combination and recorded as 300 × 300 pixel size images by a PCO camera. Figure 6 shows the results of the background removal by FPGA processor. Significant improvements can be seen at the electron beam illuminated image center after the subtraction of this background component (Fig. 6).

In the center of the image notable contrast and signal-to-noise ratio improvement can be seen (Fig. 7). This is important since the center region can be used in higher magnification cases.

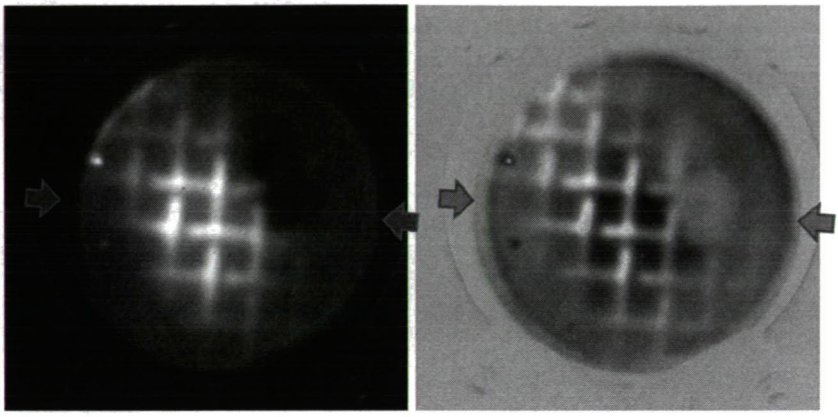


Figure 6: The original (left) [6] and background-subtracted images by FPGA processor (right). The arrows indicate the centerlines of the sample region where the intensity curves of figure 7 were measured.

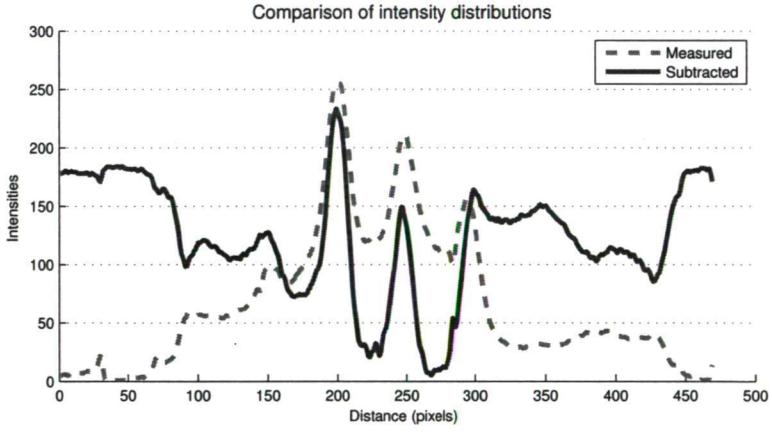


Figure 7: The intensity distribution of the images in the horizontal centerlines of the sample region marked by the blue (original) and red (background-subtracted) arrows in Fig. 6.

We have also achieved remarkable results in the relative reduction of running time. Using specialized hardware components the background computation is done efficiently in parallel. Our present system realizes two parallelized computation threads where the limitation comes only from the utilized development board's properties. Theoretically, the parallel threads are limited only by the physical resources of the FPGA and the applied serving strategy. For example, by a board with more advanced FPGA, we could implement even more than a thousand parallel Shirley modules. In this case large amounts of measured data must be sent to

the device; therefore, the bottleneck of the prototype system is the communication between the PC and the FPGA board. However, this process can be omitted by integrating the FPGA based system into the measuring device. The current FPGA prototype system has significantly shorter computation time than an ordinary computer has even though that the applied FPGA has only 50 MHz clock frequency while the PC runs on 3.6 GHz. Furthermore, the FPGA has much lower power consumption. The following table summarizes the results:

Table 1: Running times of the FPGA system with one Shirley module, with two Shirley modules in parallel, and the PC for a hundred spectra.

Spectrum length (number of points)	Running Time (ms)		
	FPGA[1]		PC[2]oftware
	1 Shirley-module	2 Shirley-module	
70	3,660	2,267	6,317
175	8,070	5,101	19,038
350	15,420	9,828	30,496
700	30,102	19,278	44,389

Table 1 shows the running times of the algorithm on different platforms. Column one holds information about the length of the spectra. The second column shows that the computation time of hundred spectra were measured. The third column shows the running time of the background computation on the FPGA[1] with one Shirley module. In this case there is no parallel computing of different spectra, just the hardware implementation of the previously described algorithm, with a pipelined design. The fourth column shows the running time of the background computations for a hundred spectra with two parallel Shirley modules on the FPGA. In those cases there are no external factors that would affect the length of the computation so they are identical for every run. The fifth column shows the running time of the software implementation of the same algorithm on a personal computer[2]. There are several factors that affects the computation time such as the scheduling of the operating system or available resources at a given moment etc. So the running time, in this case, is the average of several computations. The results on the FPGA tend to be better than on the PC (Fig. 8) even though that the PC has a much higher clock frequency ($50MHz << 3.6GHz$) and the point is to offload the efforts from the PC and embed the background correction system into the measuring device. The performance can be increased further if more Shirley modules can be placed on the FPGA.

[1]Altera DE2 board with Cyclone II FPGA clocked with 50 MHz.

[2]PC configuration: Motherboard: Gigabyte 890XA-UD3, Processor: AMD Phenom(tm) II X4 965 (4 CPUs) 3.4GHz, Memory: 4096MB RAM, Operation System: Windows 7.

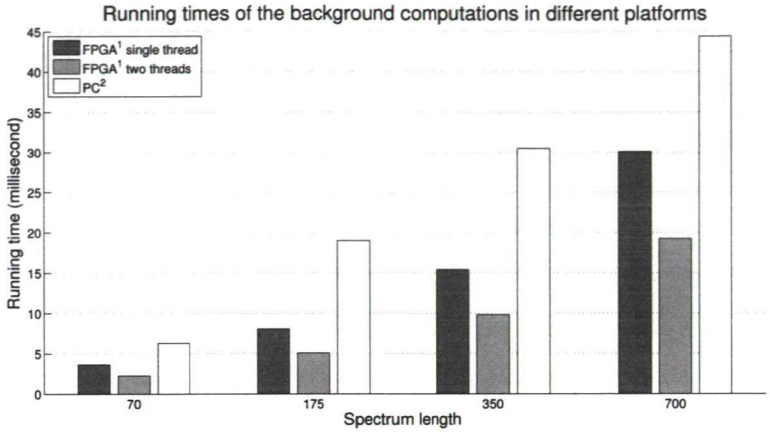


Figure 8: The proportions of the difference between the running times of different platforms remain the same if the spectrum length (energy resolution of the image sequence) is increasing.

5 Conclusions

The prototype of the background computer system provided remarkable results and important experiences which will be useful at the design of a new high performance hardware acceleration system. During the development of the prototype we realized that relevant performance enhancement requires a high-end FPGA platform which has the necessary resources to determine the background values in real-time. That device could be used as an embedded unit related to the measuring instrument so it is no longer necessary to use communication protocols between the PC and the hardware. The running time could be easily reduced in the future if the system performs the computations in more than two threads of the Shirley modules, which is limited mainly by the applied hardware resources and not by the realization of the method, therefore using a higher performance FPGA, more parallel computation modules could be executed simultaneously.

6 Acknowledgement

We would like to thank to the Japan Synchrotron Radiation Research Institute (JASRI) for getting possibility of using the BL07LSU beam-line at the Super Photon Ring - 8 GeV [13] synchrotron facility and to the Altera Company for providing the DE2 development and education board in the framework of Altera University Program.

[1] Altera DE2 board with Cyclone II FPGA clocked with 50 MHz.

[2] PC configuration: Motherboard: Gigabyte 890XA-UD3, Processor: AMD Phenom(tm) II X4 965 (4 CPUs) 3.4GHz, Memory: 4096MB RAM, Operation System: Windows 7.

References

[1] Altera Corporation. DE2 Development and Education Board User Manual. ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf, 2006.

[2] Altera Corporation. FPGAs. http://www.altera.com/products/fpga.html, 2011.

[3] Brown, S. and Vranesic, Z. *Other Number Representations*, pages 282–288. McGraw-Hill Higher Education, 2002.

[4] Brown, S. and Vranesic, Z. *Synchronous Sequential Circuits*, pages 447–527. McGraw-Hill Higher Education, 2002.

[5] Daimon, H., Matsuda, H., and Tóth, L. Stereo-peem for three-dimensional atomic and electronic structures of microscopic materials. *Surface Science*, 601(20):4748–4758, 2007.

[6] Goto, K., Matsuda, H., Hashimoto, M., Nojiri, H., Sakai, C., Matsui, F., Daimon, H., Tóth, L., and Matsushita, T. Development of display-type ellipsoidal mesh analyzer. *e-Journal of Surface Science and Nanotechnology*, 9:311–314, 2011.

[7] Matsuda, H. and Daimon, H. Approach for simultaneous measurement of two-dimensional angular distribution of charged particles. ii. deceleration and focusing of wide-angle beams using a curved mesh lens. *Physical Review E*, 74(036501), 2006.

[8] Matsuda, H., Daimon, H., Tóth, L., and Matsui, F. Approach for simultaneous measurement of two-dimensional angular distribution of charged particles. iii. fine focusing of wide-angle beams in multiple lens systems. *Physical Review E*, 75(046402), 2007.

[9] Matsuda, H., H., Daimon, Kato, M., and Kudo, M. Approach for simultaneous measurement of two-dimensional angular distribution of charged particles: Spherical aberration correction using an ellipsoidal mesh. *Physical Review E*, 71(066503), 2005.

[10] Popa, M. A flexible and general solution for reconfiguring pipeline computing systems. *International Conference on Computer Systems and Technologies*, I, 2004.

[11] Schnhense, G. and Spiecker, H. Correction of chromatic and spherical aberration in electron microscopy utilizing the time structure of pulsed excitation sources. *Journal of Vacuum Science & Technology B*, 20(1523373):2526–2534, 2002.

[12] Shirley, D. A. High-resolution x-ray photoemission spectrum of the valence bands of gold. *Physical Review B*, 5(12):47094714, 1972.

[13] SPring-8 (Super Photon Ring - 8 GeV). Japan synchrotron radiation research institute (jasri) 1-1-1, kouto, sayo-cho, sayo-gun, hyogo 679-5198 japan @ONLINE, 2011.

[14] Tóth, L., Goto, K., H., Matsuda, Matsui, F., and Daimon, H. New 1 π sr acceptance angle display-type ellipsoidal mesh analyzer for electron energy and two-dimensional angular distribution as well as imaging analysis. *Nuclear Instruments and Methods in Physics Research Section A*, 648:58–59, 2011.

[15] Tóth, L., Matsuda, H., Matsui, F., Goto, K., and Daimon, H. Details of 1 π sr wide acceptance angle electrostatic lens for electron energyand two-dimensional angular distribution analysis combined with real space imaging. *Nuclear Instruments and Methods in Physics Research Section A*, 661:98–105, 2011.

Acta Cybernetica **21** (2013) 135–147.

A Probabilistic Quality Model for C# – an Industrial Case Study*

Péter Hegedűs†

Abstract

Both for software developers and managers it is crucial to have clues about different aspects of the quality of their systems. Maintainability is probably the most attractive, observed and evaluated quality characteristic of all. The importance of maintainability lies in its very obvious and direct connection with the costs of altering the behavior of the software.

In this paper we present an existing approach and its adaptation to the C# language for estimating the maintainability of the source code. We used our model to assess the maintainability of the C# components of a large international company. We analyzed almost a million lines of code and evaluated the results with the help of IT professionals of our industrial partner.

The application of our method and model was successful as the opinions of the developers showed a 0.92 correlation with the maintainability values produced by our C# maintainability model.

Keywords: software maintainability, ISO/IEC 9126, C# quality model, industrial case study

1 Introduction

Both for software developers and managers it is crucial to have clues about different aspects of the quality of their systems. The information can mainly be used for making decisions, backing up intuition, estimating future costs and assessing risks. The ISO/IEC 9126 standard [13] defines six high-level product quality characteristics which are widely accepted both by industrial experts and academic researchers. These characteristics are: *functionality*, *reliability*, *usability*, *efficiency*, *maintainability* and *portability*. The characteristics are affected by low-level quality properties that can be *internal* (measured by looking inside the product, e.g. by analyzing the source code) or *external* (measured by execution of the product, e.g. by performing testing).

*The publication is supported by the European Union and co-funded by the European Social Fund, TÁMOP-4.2.2/B-10/1-2010-0012.

†University of Szeged, E-mail: hpeter@inf.u-szeged.hu

Maintainability is probably the most attractive, observed and evaluated quality characteristic of all. The importance of maintainability lies in its very obvious and direct connection with the costs of altering the behavior of the software. Although, the quality of source code unquestionably affects maintainability, the standard does not provide a consensual set of source code measures as internal quality properties. The standard also does not specify the way how the aggregation of quality attributes should be performed. These are not deficiencies of the standard, but it offers a kind of freedom to adapt the model to specific needs.

We have introduced a practical quality model in one of our previous works [3] that differs from the other models (e.g. [4, 5, 6, 12, 17]) in many ways:

- It uses a large number of other systems as benchmark for the qualification.
- The approach takes into account different opinions of many experts, and the algorithm integrates the ambiguity originating from different points of view in a natural way.
- The method uses probabilistic distributions instead of average metric values, therefore providing a more meaningful result, not just a single number.

Although the introduced model proved to be useful and accepted by the scientific community, real industrial settings and evaluations are required to show that our solution is useful and applicable in real environments too. Additionally, the first published model is only a prototype for the Java language and weighted by a small number of researchers and practitioners. The goal of the current work is to develop a method and model for estimating the maintainability of the C# systems of a large international company. To achieve this goal, the following tasks were completed:

- Together with the industrial partner, we have introduced a new maintainability model for systems written in the C# language.
- A benchmark from the C# systems of the company has been created (almost a million C# code lines has been analyzed).
- A method and tool has been developed for qualifying the smaller components of the company's software using the benchmark - producing a relative measure for maintainability of the components (we were able to rank the components of the company).
- A new weighting has been created involving the developers and managers.
- According to the method and model, a large number of components have been evaluated.

The results were discussed after the evaluation and compared to the developers' opinions. The industrial application of our method and model was successful, as the opinions of the developers highly correlated with the maintainability values

produced by our C# maintainability model. This result shows that our probabilistic quality model is applicable in industry, as the industrial partner accepted the provided results and found our approach and tool very useful.

The rest of the paper is organized as follows. First, we give a short overview about the related work in Section 2. Then, in Section 3 we introduce our approach for estimating maintainability and describe the details of the case study setup. In Section 4 we present and evaluate the results of the case study. Section 5 collects the possible threats to the validity of our work. Finally, we conclude the paper in Section 6.

2 Related Work

A large number of works deal with software maintainability analysis using the source code of the software. Kanellopoulos et al. [14] apply data mining and clustering techniques to comprehend an object-oriented system and evaluate its maintainability. They successfully explore some quality attributes of the *JBoss* open source system. To understand whether some technologies consistently outperform others, Sartori et al. [19] use some crude indicators, such as the density of bugs and the time required to fix bugs. They find that there is a connection between these indicators and the programming language of the software. We also use source code metrics and static analysis but unlike these works we are particularly interested in the ISO/IEC 9126 based qualification of an industrial software. Moreover, we focus on the C# language and its specific quality attributes to be able to assess the maintainability of programs.

There are many other existing practical quality models. Some of them [2, 4, 7, 12, 16] adapt the ISO/IEC 9126 standard. These models are based on low level source code metrics and a method for aggregating these values to higher levels. There are other models also, e.g. Bansiya and Davis [4] present a hierarchical model for assessing the quality of object-oriented design. Their model focuses on design level metrics. Plösch et al. introduce a quality model [18] based on a technical topic classification. Their approach is more technical than the ISO/IEC 9126 and makes it easier to assign metrics - provided by static code analysis tools - to quality attributes. Letouzey presents a quality model and analysis model [15] which is used to estimate the quality and the technical debt of an application source code. He uses the concept of technical debt for expressing the quality of the software (larger debt indicates worse maintainability) based on different indicators calculated from the source code. Although we also introduce a new quality model, our primary purpose is to empirically validate its results with the help of an industrial case study rather than showing only a formal approach or results on small examples. We focus on the quality of C# systems while the mentioned quality models are either specific to Java or very general (applicable to any OO system).

There are many papers presenting industrial case studies of maintainability analysis of C# systems. But while we are interested in the ISO/IEC 9126 based qualification, they focus on different quality properties. Gatrell et al. [10] perform a study analyzing the refactorings in a commercial C# software comprising 270

versions. In another work Gatrell and Counsell focus on the fault-proneness [9] of the code, they document a study of change in commercial, proprietary C# software and attempt to determine whether a relationship exists between class changes and faults. The work of Goldschmidt et al. [11] concentrates particularly on the maintainability of the applied persistency techniques in C# systems.

3 Approach

3.1 Probabilistic Software Quality Model

Our probabilistic software quality model [3] is based on the quality characteristics defined by the ISO/IEC 9126 standard.

The computation of the high level quality characteristics is based on a directed acyclic graph whose nodes correspond to quality properties that can either be internal (low-level) or external (high-level). Internal quality properties characterize the software product from an internal (developer) view and are usually estimated by using source code metrics. External quality properties characterize the software product from an external (end user) view and are usually aggregated somehow by using internal and other external quality properties. The nodes representing internal quality properties are called *sensor nodes* as they measure internal quality directly. The other nodes are called *aggregate nodes* as they acquire their measures through aggregation. The edges of the graph represent dependencies between an internal and an external or two external properties. The aim is to evaluate all the external quality properties by performing an aggregation along the edges of the graph, called Attribute Dependency Graph (ADG).

Goodness is the term that we use to express how good or bad an attribute is. We assume that the goodness of each sensor node is not known precisely; hence it is represented by a random variable with a probability density function, the *goodness function.* For constructing goodness functions we make use of the metric histogram over the source code elements, as it characterizes the whole system from the aspect of one metric. As the notion of goodness is relative, we expect it to be measured by means of comparison with other histograms. Applying a distance function between two histograms, we get one goodness value for the subject histogram that is relative to the other histogram. In order to obtain a proper goodness function, we repeat this comparison with histograms of many different systems independently. In each case we get a goodness value which can basically be regarded as sample of a random variable from the range $[-\infty, \infty]$. Interpolation of the empirical density function leads us to the goodness function of the sensor node.

Besides constructing goodness functions for sensor nodes, we need a way to aggregate them along the edges of the ADG. We created an online survey where we asked many experts (both industrial and academic people) for their opinion about the weights between quality attributes. The number assigned to an edge is considered to be the amount of contribution of source goodness to target goodness. Consequently, the votes form a multi-dimensional random variable for each

aggregate node. Taking into account every possible combination of goodness values and weights, and also the probabilities of their outcome, we defined a formula for computing goodness functions for every aggregate node. It is the multi-dimensional extension of the classical linear combination. The aggregation method ensures that the goodness functions of aggregate nodes are really expressing the probabilities of their goodness from the aspect represented by the node.

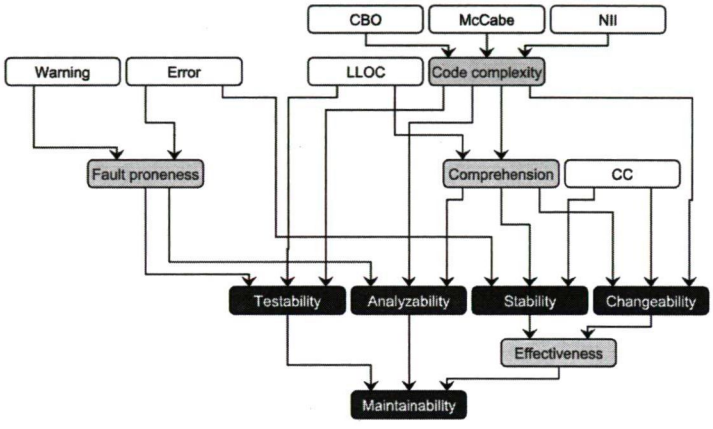


Figure 1: Java maintainability model

In our previous work [3] we have introduced a prototype ADG (see Figure 1) for Java language. To be able to perform the construction of goodness functions in practice, we have built a source code metric repository database, where we uploaded source code metrics of more than 100 open source and industrial software systems. To get an absolute measure for software maintainability, we calculated all the maintainability values of the systems in the repository. The resulting distribution of the values can be seen on Figure 2. As the values given by the quality model are unbounded we can assign to each system a rank value according to this distribution. The rank is a real value between 0 and 1 that objectively reflects the absolute maintainability value of a system based on the used repository database. Note that the rank is exactly the proportion of the systems in the repository that have a worse maintainability value than the subject system.

3.2 Adaptation of the Approach to C# Language

To adapt the previously described approach for qualifying the C# components of our industrial partner we have introduced a new ADG. As a result of a collective work the ADG shown in Figure 3 has been developed. It is much larger than the Java prototype ADG and contains some C# specific rule violations also. We chose FxCop [1] as a rule checker and built the number of different rule violations into the model as sensor nodes. The reason behind choosing FxCop is that it is a widely accepted rule checker in the C# world and our industrial partners already used this

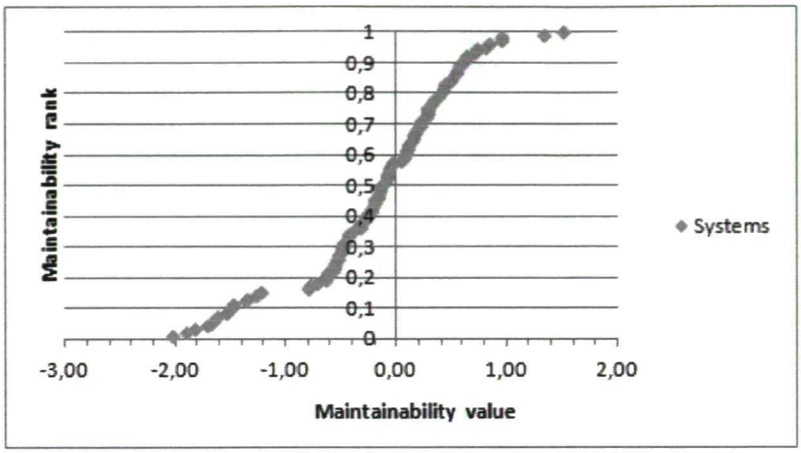


Figure 2: Distribution of the maintainability of benchmark systems

checker at the time of model construction. For calculating the source code metrics we used the Columbus toolset [8]. The sensor nodes included in the model can be seen in Table 1. We note that the differences in the sensor nodes compared to the Java model are rather general improvements on the model than specialties of C#. Only FxCop rule violations were added due to the adaption of the C# language.

Table 1: Sensor nodes in the model

DIT	Depth of inheritance tree	*NLE*	Nesting level
NOI	Number of outgoing invocations	*IR*	Interoperbility Rules
CBO	Coupling between object classes	*NR*	Naming Rules
McCabe	McCabe's cyclomatic complexity	*CC*	Clone coverage
LCOM5	Lack of cohesion on methods	*PR*	Performance Rules
DR, UR	Design Rules and Usage Rules[1]	*SR*	Security Rules
NII	Number of incoming invocations	*LLOC (class)*	Logical code lines of classes
LLOC (method)	Logical code lines of methods		

The model contains the following intermediate aggregated nodes:

- *Unit Complexity* – the class level complexity of the system.
- *System Complexity* – the complexity of the system as a whole.
- *Code Complexity* – the general complexity of the source code.

[1] All the rule sensor nodes refer to the number of FxCop rule violations of that group found in the system.

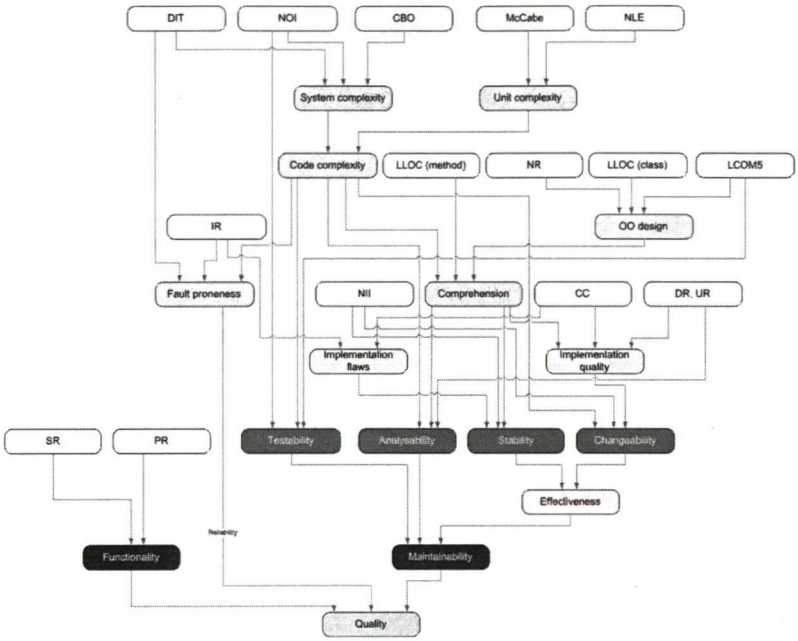


Figure 3: The created C# maintainability model

- *OO design* – the fulfillment of OO design principles.
- *Fault-proneness* – the fault proneness of the code due to dangerous program constructs.
- *Comprehension* – how easy it is to comprehend the code of the system.
- *Implementation flaws* – the implementation problems in the system.
- *Implementation quality* – the low level quality of the code implementation.
- *Effectiveness* – the effectiveness of the code change.

The ISO/IEC 9126 quality characteristics[2] in the model are the following:

- *Testability* – the capability of the software product to enable modified software to be validated.
- *Analyzability* – the capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified.
- *Changeability* – the capability of the software product to enable a specified modification to be implemented, where implementation includes coding, designing and documenting changes.

[2]The model does not include all the characteristics defined by the standard yet.

- *Stability* – the capability of the software product to avoid unexpected effects from modifications of the software.
- *Functionality* – the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions. The functions satisfy the formulated or supposed conditions.
- *Maintainability* – the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.
- *Quality* – the overall quality of the software system.

4 Results and Evaluation

To make the maintainability model work we had to build a benchmark database from different C# systems. Since our approach of creating benchmark database from large number of open source systems was not applicable, in this case we needed a new idea. As our industrial partner owns a huge amount of C# code itself and they were only interested in the code maintainability of their components compared to each other we decided to build a benchmark from their over 300 components[3]. This way we were able to give a relative maintainability value for each component (estimate the component's maintainability in comparison to other components). Moreover, we could define a ranking based on the relative maintainability between the components. Some basic properties of our partner's source code can be seen in Table 2.

Table 2: Basic properties of the industrial partner's software

Property	Value
Total number of logical lines of code	711 944
Total number of classes in the system	4 942
Total number of methods in the system	48 787
Number of components in the system	315

As a final step before the qualification of the components a weighting was introduced on the created C# ADG. 7 IT professionals from our industrial partner and 5 academical co-workers voted on the importance of each dependency between quality attributes. We assigned the distribution of the votes to each edge in the ADG as weights to be able to perform the aggregation algorithm (for details see Section 3.1).

With the help of the adapted model, benchmark and votes we calculated the maintainability values of each component. The results of 10 selected components

[3]Here we refer to the source code of a self-compilable *dll* or *exe* as a component.

can be seen in Table 4. The detailed results of the best out of these 10 components is shown in Figure 4. On the left hand side we present the goodness values of low level quality properties (i.e. sensor nodes) of the system. Although our model works with goodness functions we can simply derive a single value by taking the average of the samples. 0 means the worst, while 1 means the best achievable result. On the right side the goodness values of high level quality attributes (i.e. aggregate nodes) are shown. The method level code lines (*LLOC*) got the worst qualification from the sensor nodes. From ISO characteristics *Functionality* got the best score according to our model.

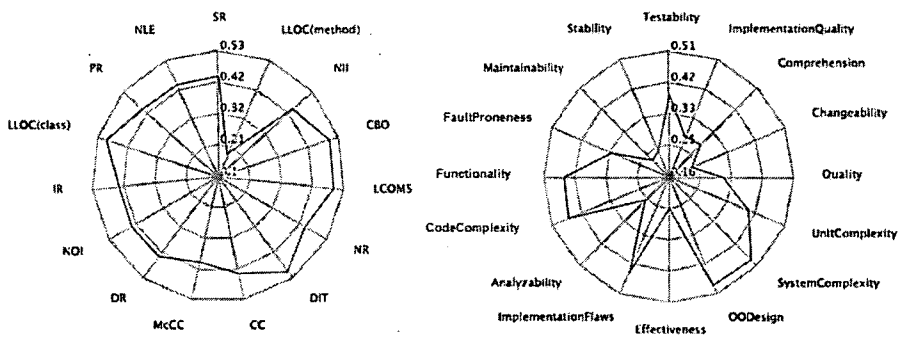


Figure 4: Detailed results of a C# component

The maintainability values of the 10 presented components were manually evaluated by 7 IT professionals of the company. We chose components that each of the IT professionals knew well. So they were able to subjectively assess the maintainability of these components. Every IT professional scored the maintainability of the 10 components on a scale from 0 to 10. 0 means the worst possible maintainability, 10 the best. After collecting all the votes we calculated the average of these votes and divided the value by 10 to convert the value to the [0,1] interval. In this way we could compare the maintainability values provided by our model with the average votes of the IT professionals. Table 4 shows the maintainability values together with the normalized average subjective votes. Although the average human votes are higher than the estimated values the Pearson correlation analysis gave **0.923**, a very high correlation between the two data sets. Since our model does not calculate an absolute maintainability measure, the values cannot be compared directly. However, the correlation analysis showed that our model was able to assess the maintainability of the components relatively to each other which was our goal.

Due to the small sample size and low variance in the values we performed a test where we randomly excluded two out of the ten cases and recalculated the Pearson's and Spearman's rank correlation values. We wanted to rule out that the

high correlation is caused by a few dominant values. Table 3 shows the recalculated correlation values for ten cases.

Table 3: The recalculated correlation values

Pearson's	0.879	0.925	0.879	0.924	0.948
Spearman's	0.896	0.896	0.896	0.854	0.970
Pearson's	0.925	0.924	0.923	0.901	0.777
Spearman's	0.812	0.854	0.916	0.896	0.766

The last column presents the case when the largest and smallest value is removed. Even in this case the correlation values remain fairly high. But in general, we can say that there is not much variance in the original and recalculated values meaning that the high correlation is not caused by some dominant values. Moreover, Spearman's rank correlation is even more stable than Pearson's correlation. It is good because it relies on the ordering of the values and the ordering of the qualifications is the most important in our case study.

Additionally to the subjective voting, we have discussed the results of all these components with the IT professionals in detail. Every extreme sensor values have been justified and for every component we reached a consensual acceptance of the goodness values of high level quality characteristics. Moreover, all the IT professionals agreed with the ranking of the components suggested by the maintainability values.

Table 4: The maintainability values and the average IT professional votes

Maintainability	0.311	0.261	0.261	0.261	0.26
Avg. expert vote	0.56	0.48	0.473	0.53	0.47
Maintainability	0.26	0.221	0.221	0.216	0.178
Avg. expert vote	0.49	0.4	0.44	0.45	0.3

5 Threats to Validity

An obvious threat to the validity of the presented results is the fact that the calculated maintainability values are relative. This means that the presented values reflect the maintainability of the components in comparison to the code base of our industrial partner. What follows from this is that the best component has a maintainability value of 1 and the worst has 0. We cannot place the maintainability of the components on an absolute scale based on these values. However, our primary goal was not to give an absolute measure for maintainability but to be able to rank

and compare the components of our partner according to their maintainability. For this it is enough to have a relative maintainability value because the ranking of the components would be the same using absolute measures. Moreover, it is very easy to adapt our approach to get an absolute measure for maintainability. Only the benchmark database should be extended with many other third party C# systems.

Another major threat is a relatively small number of experts taking part in model declaration and dependency weighting. As we try to estimate a subjective concept a very large number of human opinions would be required to get an objective estimation. However, our experiences show that with the votes of more than 10 persons a very good estimation can be reached. Additionally, it is also very easy to add more expert votes to our model and get more reliable data.

Since it would have been a tiresome task to manually validate the results of all the 315 components we selected 10 of them. So our conclusions about the validity of our maintainability estimations are limited to the evaluated components. Nevertheless, we think that due to the nature of the selection we examined a representative sample of the results and our model indeed approximates the opinions of the IT professionals well.

6 Conclusions and Future Work

In this paper we presented an approach and a concrete model for estimating the maintainability of C# systems. Our primary purpose was to prove the applicability of our previous approach in an industrial environment. Therefore, we adapted our Java specific quality model to assessing the maintainability of C# systems.

This work has been performed in collaboration with the co-workers of our industrial partner, an international software development company. The adaptation of our previous approach included the following steps: 1) introduction of a new maintainability model for C# systems, 2) creation of a benchmark from the C# components of the company, 3) development of a method and tool for qualifying the smaller components of the company's software using the benchmark, 4) introduction of a new weighting involving the developers and managers of the company, and 5) evaluation of a large number of components.

The results were discussed after the evaluation and compared to the developers' opinions. The industrial application of our method and model was successful, as the opinions of the developers highly correlated with the maintainability values produced by our C# maintainability model. This result shows that our probabilistic quality model is applicable in industry, as the industrial partner accepted the provided results and found our approach and tool very useful.

An obvious direction of the future research is to collect more votes to model weights and evaluate more C# systems. The manual inspection of the model based estimations could help improving the qualification process and model.

We also plan to extend the presented approach to different languages. We already have preliminary results on a simple PL/SQL and C maintainability model. Besides these languages there is also a plan to adapt our qualification approach to

C++ and Python languages. After completing the prototype versions of these models we plan to perform a similar industrial case study to that presented here.

References

[1] FxCop Home Page.
`http://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx`.

[2] Baggen, R., Schill, K., and Visser, J. Standardized Code Quality Benchmarking for Improving Software Maintainability. In *Proceedings of the Fourth International Workshop on Software Quality and Maintainability (SQM2010)*, 2010.

[3] Bakota, T., Hegedűs, P., Körtvélyesi, P., Ferenc, R., and Gyimóthy, T. A Probabilistic Software Quality Model. In *Proceedings of the 27th IEEE International Conference on Software Maintenance*, ICSM 2011, pages 368–377, Williamsburg, VA, USA, 2011. IEEE Computer Society.

[4] Bansiya, J. and Davis, C.G. A Hierarchical Model for Object-Oriented Design Quality Assessment. *IEEE Transactions on Software Engineering*, 28:4–17, 2002.

[5] Carvallo, J. P. and Franch, X. Extending the ISO/IEC 9126-1 Quality Model with Non-technical Factors for COTS Components Selection. In *Proceedings of the 2006 international workshop on Software quality*, WoSQ '06, pages 9–14, New York, NY, USA, 2006. ACM.

[6] Chua, B.B. and Dyson, L.E. Applying the ISO9126 Model to the Evaluation of an E-learning System. In *Beyond the comfort zone: Proceedings of the 21st ASCILITE Conference*, pages 184–190, Perth, Australia, 2004. Citeseer.

[7] Correia, J. P., Kanellopoulos, Y., and Visser, J. A Survey-based Study of the Mapping of System Properties to ISO/IEC 9126 Maintainability Characteristics. *IEEE International Conference on Software Maintenance*, pages 61–70, 2009.

[8] Ferenc, R., Beszédes, Á., Tarkiainen, M., and Gyimóthy, T. Columbus – Reverse Engineering Tool and Schema for C++. In *Proceedings of the 18th International Conference on Software Maintenance (ICSM 2002)*, pages 172–181. IEEE Computer Society, October 2002.

[9] Gatrell, M. and Counsell, S. Size, Inheritance, Change and Fault-proneness in C# Software. *Journal of Object Technology*, 9(5):29–54, 2010.

[10] Gatrell, M., Counsell, S., and Hall, T. Empirical Support for Two Refactoring Studies Using Commercial C# Software. In *Proceedings of the 13th international conference on Evaluation and Assessment in Software Engineering*, EASE'09, pages 1–10, Swinton, UK, 2009. British Computer Society.

[11] Goldschmidt, T., Reussner, R., and Winzen, J. A Case Study Evaluation of Maintainability and Performance of Persistency Techniques. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 401–410, New York, NY, USA, 2008. ACM.

[12] Heitlager, I., Kuipers, T., and Visser, J. A Practical Model for Measuring Maintainability. *Proceedings of the 6th International Conference on Quality of Information and Communications Technology*, pages 30–39, 2007.

[13] ISO/IEC. *ISO/IEC 9126. Software Engineering – Product quality.* ISO/IEC, 2001.

[14] Kanellopoulos, Y., Dimopulos, T., Tjortjis, C., and Makris, C. Mining Source Code Elements for Comprehending Object-oriented Systems and Evaluating Their Maintainability. *SIGKDD Exploration Newsletter*, 8(1):33–40, June 2006.

[15] Letouzey, J. L. The SQALE Method for Evaluating Technical Debt. In *Third International Workshop on Managing Technical Debt (MTD)*, pages 31–36. IEEE, June 2012.

[16] Muthanna, S., Ponnambalam, K., Kontogiannis, K., and Stacey, B. A Maintainability Model for Industrial Software Systems Using Design Level Metrics. In *Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, WCRE '00, pages 248–256, Washington, DC, USA, 2000. IEEE Computer Society.

[17] Oman, P. and Hagemeister, J. Metrics for Assessing a Software System's Maintainability. In *Proceerdings of the Conference on Software Maintenance*, volume 19, pages 337–344. IEEE Computer Society Press, 1992.

[18] Plösch, R., Gruber, H., Körner, C., Pomberger, G., and Schiffer, S. A Proposal for a Quality Model Based on a Technical Topic Classification. In *Proceedings of SQMB 2009 Workshop*, 2009.

[19] Sartori, V., Eshete, B., and Villafiorita, A. Measuring the Impact of Different Metrics on Software Quality: a Case Study in the Open Source Domain. In *Proceedings of the Fifth International Conference on Digital Society*, 2011.

Acta Cybernetica **21** (2013) 149–163.

An Empirical Study of Reconstructing hv-Convex Binary Matrices from Horizontal and Vertical Projections*

Zoltán Ozsvár† and Péter Balázs‡

Abstract

The reconstruction of hv-convex binary matrices (or equivalently, binary images) from their horizontal and vertical projections is proved to be NP-hard. In this paper we take a closer look at the difficulty of the problem. We investigate different heuristic reconstruction algorithms of the class, and compare them from the viewpoint of running-time and reconstruction quality. Using a large set of test images of different sizes and with varying number of components, we show that the reconstruction quality can depend not only on the size of the image, but on the number and location of its components, too. We also reveal that the reconstruction time can also be affected by the number of the so-called switching components present in the image.

Keywords: discrete tomography, reconstruction algorithm, hv-convex binary matrix, kernel-shell method, simulated annealing

1 Introduction

Tomography is a method of producing a 3D image of the internal structure of an object from its projections, without damaging it. This is usually achieved by reconstructing 2D slices from the projections and then assembling them. Applications of computerized tomography arise from various fields of science: image processing, medical imaging, nondestructive testing, electron microscopy, etc. The Filtered

*This work was supported by the European Union and co-funded by the European Social Fund under the grant agreement TÁMOP-4.2.2/B-10/1-2010-0012, "Broadening the knowledge base and supporting the long term professional sustainability of the Research University Centre of Excellence at the University of Szeged by ensuring the rising generation of excellent scientists". The work of Péter Balázs was also supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and the OTKA PD 100950 grant of the Hungarian Scientific Research Fund.

†Faculty of Science and Informatics, University of Szeged, Aradi vértanúk tere 1., H-6720 Szeged, Hungary, E-mail: Ozsvar.Zoltan@stud.u-szeged.hu

‡Department of Image Processing and Computer Graphics, University of Szeged, Árpád tér 2., H-6720 Szeged, Hungary, E-mail: pbalazs@inf.u-szeged.hu

Backprojection and variants of the Algebraic Reconstruction Methods are the most commonly used algorithms to reconstruct images from their projections [8, 12]. However, they require several hundreds of projections to ensure an acceptable image quality. In *Binary Tomography* (BT) we assume that the examined object is homogeneous, thus the image to be reconstructed contains only black (object) and white (background) pixels. With this additional information, algorithms of BT can often produce images of good quality, even from just a small amount (say, at most 10-20) of projections [9, 10]. The reconstruction of a binary image is also feasible from just its horizontal and vertical projections [15]. Nevertheless, in that case the task is usually extremely underdetermined, i.e., there may be numerous different binary images with the same two projections. To overcome this problem, we can assume that the image satisfies certain geometrical conditions, too. In this paper we study the reconstruction of hv-convex binary images from the horizontal and vertical projections, from an experimental point of view. We perform several tests to compare different algorithms for reconstructing hv-convex binary images. The structure of the paper is the following. Chapter 2 is for the preliminaries. In Chapter 3 we describe three heuristical algorithms for solving the abovementioned task. In Section 4 we give our experimental results. Finally, Section 5 is for the conclusion.

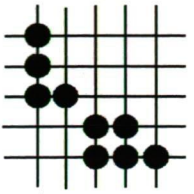
2 Preliminaries

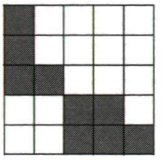
A *binary image* is a digital image containing just black (also called as object or foreground) and white (background) pixels. A binary image of size $m \times n$ (where $m, n \in \mathbb{N}$) can also be represented by a binary matrix $F = (f_{ij})_{m \times n}$ where value 1 (respectively, value 0) indicates that the color of the corresponding pixel is black (respectively, white). *Discrete sets* (finite subsets of the 2D integer lattice $\mathbb{Z}^2$) can also be used to represent a binary image, with the agreement that the rows are numbered from top to bottom (and the columns are numbered from left to right). A position of the lattice belongs to the discrete set if and only if the corresponding matrix position has value 1. In this paper we will use the three concepts equivalently. Since rows/columns with 0 projection value can be reconstructed easily in a preprocessing step and then be eliminated, we also will assume that each row and column of the matrix contains at least one 1. Figure 1 shows the three different representations of the same binary image.

The *horizontal* and *vertical projection* of the image F is the vector $\mathcal{H}(F) = H = (h_1, \ldots, h_m)$ and $\mathcal{V}(F) = V = (v_1, \ldots, v_n)$, respectively, where

$$h_i = \sum_{j=1}^{n} f_{ij} \ (i = 1, \ldots, m) \quad \text{and} \quad v_j = \sum_{i=1}^{m} f_{ij} \ (j = 1, \ldots, n). \tag{1}$$

For example, the binary image F in Fig. 1 has the horizontal and vertical projection $\mathcal{H}(F) = (1, 1, 2, 2, 3)$, and $\mathcal{V}(F) = (3, 1, 2, 2, 1)$, respectively. The class of all binary matrices having the horizontal projection H, and vertical projection V will be denoted by $\mathcal{BM}(H, V)$.





$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Figure 1: Different representations of the same object as a discrete set, as a binary image, and as a binary matrix (from left to right).

A *switching component* of a matrix is a 2×2 submatrix of the form

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{2}$$

To avoid confusion, we emphasize that the above submatrix is formed by selecting two (not necessarily consecutive) rows and two (not necessarily consecutive) columns of the matrix. It is easy to see that the horizontal and vertical projections of a matrix do not change if we invert the values in the four positions of a switching component. A much stronger statement is also true.

Theorem 1. *[15] Let $A, B \in \mathcal{BM}(H, V)$ $(B \neq A)$. Then A is transformable into B (or vice versa) by a finite sequence of switching components.*

The *reconstruction task* consists in finding a binary image with the given horizontal and vertical projections. However, due to the presence of switching components, those projections usually do not uniquely determine the image itself. Thus, further prior information is needed in order to reduce the number of possible solutions. Two positions $P = (p_1, p_2)$ and $Q = (q_1, q_2)$ in a discrete set F are said to be *4-adjacent* if $|p_1 - q_1| + |p_2 - q_2| = 1$. The positions P and Q are *4-connected* if there is a sequence of distinct positions $P_0, \ldots, P_k$ in F such that $P_0 = P$, $P_k = Q$, and P_l is 4-adjacent to P_{l-1} (for each $l = 1, \ldots, k$). A discrete set F is *4-connected* if any two points in F are 4-connected. Every discrete set F can be partitioned (in a uniquely determined way) into maximal 4-connected subsets, which are called the *components* of F. The discrete set is called *h-convex* (respectively, *v-convex*) if its elements follow consecutively in each row (respectively, in each column). The discrete set is called *hv-convex* if it is both h- and v-convex. Figure 2 demonstrates the above concepts.

3 Algorithms for Reconstructing hv-Convex Binary Images

It is known, that the reconstruction of general hv-convex discrete sets from their horizontal and vertical projections is an NP-hard problem [16]. On the other hand,

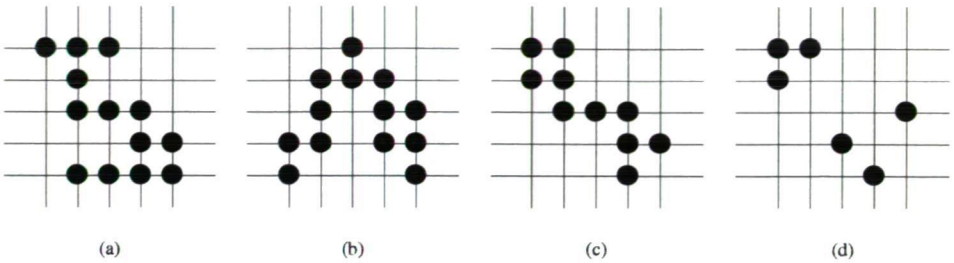


Figure 2: An h-convex 4-connected discrete set (a), a v-convex 4-connected discrete set (b), an hv-convex 4-connected discrete set (c), and a general hv-convex discrete set with 4 components (d).

if the hv-convex discrete set is also 4-connected, then the reconstruction can be solved in polynomial time [4, 5, 6]. Our goal is to reveal the reason of being the more general problem computationally hard. In this section, we shortly describe three heuristic algorithms from previous works to solve the problem.

3.1 Kernel-Shell Algorithm

The kernel-shell (or core-envelope) algorithm (see Algorithm 1) is a greedy type heuristic algorithm which approximates the discrete set F to be reconstructed by two sequences of discrete sets [13]. The first sequence is nondecreasing, and it consists of the so-called *core sets* which satisfy

$$C_0 \subseteq C_1 \subseteq \cdots \subseteq F, \tag{3}$$

while the second (nonincreasing) sequence of so-called *envelope sets* satisfies

$$S_0 \supseteq S_1 \supseteq \cdots \supseteq F. \tag{4}$$

Denoting the minimal bounding rectangle of F by T, as initial core and envelope sets we can use

$$C_0 = \emptyset \quad \text{and} \quad S_0 = T. \tag{5}$$

In every iteration, the new core set is constructed as the maximal hv-convex discrete set (operator J) from the current core set C_k, extending it to horizontal and vertical directions by taking into account the horizontal and vertical projection values (operator c) and the current envelope set S_k (Step 2). Similarly, the new shell is constructed as the intersection of the maximal possible extensions of the current core C_k, by taking into account the horizontal and vertical projection values (operator s) and the current envelope set S_k (Step 3).

If the kernel and the shell coincide, then we found a solution (Steps 4-6). Otherwise, if $C = C_k$ (i.e., the core set does not change in an iteration), then the core cannot be further increased. Thus, we use a stack memory P, and guess all the

positions of the shell not belonging to the kernel whether the core can be extended with that given position (Steps 7-9). This heuristic step might be repeated (if needed) several times (Steps 10-11). If we get that $C_k \not\subseteq S_k$, then the guess led to a contradiction, and we do a backtrack step (by deleting the tested position from the shell) if possible, i.e., if the stack is not empty (Steps 15-18). Otherwise, there is no solution (Steps 13-14).

Algorithm 1 Core-Shell Algorithm

Input: the vectors $H \in \mathbb{N}^m$ and $V \in \mathbb{N}^n$

Output: the binary matrix F with $\mathcal{H}(F) = H$, $\mathcal{V}(F) = V$ or the message "no solution"

1: $C_0 = \emptyset, S_0 = T, k = 0$, P is an empty stack
2: $C := J(c(S_k, H) \cup c(S_k, V) \cup C_k)$
3: $S := s(C_k, H) \cap s(C_k, V) \cap S_k$
4: **if** $C_k = S_k$ **then**
5: **return** F
6: **end if**
7: **if** $C = C_k$ **then**
8: $(C, S, ((i,j) \in S \backslash C) \rightarrow P); C := C \cup \{(i,j)\}; C_{k+1} := C; S_{k+1} := S; k := k+1$
9: **end if**
10: **if** $C_k \subset S_k$ **then**
11: **goto** Step 2
12: **else**
13: **if** $C_k \not\subseteq S_k$ and P is empty **then**
14: FAIL (no solution)
15: **else**
16: $P \rightarrow (C, S, (i,j)); C_{k+1} := C; S_{k+1} = S_k \setminus \{(i,j)\}; k := k+1$; **goto** Step 2
17: **end if**
18: **end if**

3.2 Algorithm Based on Simulated Annealing

The next algorithm is based on Simulated Annealing (SA) [14] where the objective is to maximize the number of adjacent ones in an unknown binary matrix $X = (x_{ij})_{m \times n}$, i.e., the following function

$$f(X) = \sum_{i=1}^{m-1} \sum_{j=1}^{n} x_{ij} x_{i+1,j} + \sum_{i=1}^{m} \sum_{j=1}^{n-1} x_{ij} x_{i,j+1} \tag{6}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = h_j \ (i = 1, \ldots, m) \tag{7}$$

$$\sum_{i=1}^{n} x_{ij} = v_j \; (j = 1, \ldots, n) \tag{8}$$

with $x_{i,j} \in \{0,1\}$ $(i = 1, \ldots, m$ and $j = 1, \ldots, n)$. Constraints (7) and (8) ensure that the matrix X satisfies the given projections.

Algorithm 2 outlines this method that was published in [11] on the basis of [7]. For generating an initial solution that satisfies the projections, Ryser's algorithm is used [15]. The neighborhood of a solution matrix is defined as the set of all matrices obtained by a single switching. As Theorem 1 says, by applying such switchings, all the binary matrices satisfying the given projections can be constructed. In Step 3, $p \in (0,1)$ is a random variable, generated in each iteration from a uniform random distribution. The initial temperature was set to 5, and the algorithm was terminated when the best solution did not improve for the last 1000 iterations or when the temperature reached 0.0005. Those parameters as well as the cooling factor were set empirically by a long process of trial and error.

Algorithm 2 Algorithm Based on Simulated Annealing

Input: X_{act} = computed initial solution, $Temp = 5$, $Nbr = 0$
Output: the binary matrix X with $\mathcal{H}(X) = H$ and $\mathcal{V}(X) = V$

1: **while** ($Temp > 0.0005$ and $Nbr < 1000$) **do**
2: $X_{next} \longleftarrow$ invert a randomly chosen switching component in X_{act}
3: **if** $(f(X_{act}) < f(X_{next}))$ or $(p < exp(-|f(X_{act}) - f(X_{next})|/Temp)$ **then**
4: $X_{act} \leftarrow X_{next}$; $Temp := Temp \cdot 0.9995$; $Nbr := 0$;
5: **else**
6: Nbr++;
7: **end if**
8: **end while**

3.3 Algorithm Based on the Location of the Components

Let F be a binary image with k $(k \geq 1)$ components such that $I_l \times J_l = [i_l, i'_l] \times [j_l, j'_l]$ is the minimal bounding rectangle of the l-th component of F $(l = 1, \ldots, k)$. We say that the components of F are *disjoint* if for any $l \neq l'$ (where $1 \leq l, l' \leq k$) $I_l \cap I_{l'} = \emptyset$ and $J_l \cap J_{l'} = \emptyset$ (see Fig. 3).

In [1] the author presented an algorithm to reconstruct discrete sets having disjoint components from their horizontal and vertical projections, by locating the possible positions of the components. It is clear, that the hv-convex images naturally consist of disjoint components. Moreover, those components are hv-convex 4-connected images, which can be reconstructed in polynomial time ([4, 5, 6]). Thus, this algorithm is also capable of the fast reconstruction of hv-convex images. The outline of the method is given as Algorithm 3, which uses the following definition.

Definition 1. *Let $\mathcal{S}$ be a class of 4-connected binary images, $H \in \mathbb{N}^m$ and $V \in \mathbb{N}^n$. We say that the intervals $[i_1, i_2]$ of H $(1 \leq i_1 \leq i_2 \leq m)$ and $[j_1, j_2]$ of V $(1 \leq j_1 \leq$*

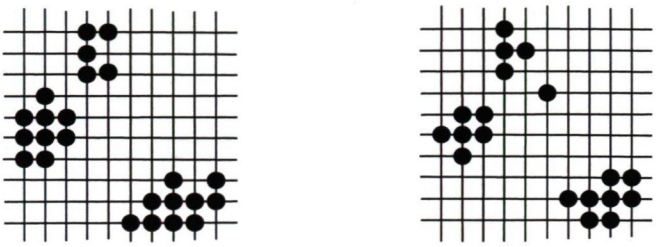


Figure 3: A discrete set with three disjoint components (left), and an hv-convex discrete set with four disjoint components (right).

$j_2 \leq n$) *are* compatible with respect to the class $\mathcal{S}$ *if a 4-connected binary image* $P \in \mathcal{S}$ *exits with* $\mathcal{H}(P) = (h_{i_1}, \ldots, h_{i_2})$ *and* $\mathcal{V}(P) = (v_{j_1}, \ldots, v_{j_2})$.

Algorithm 3 Algorithm Based on the Location of the Components

1: find and store all compatible interval-pairs $[i_1, i_2]$ of H and $[j_1, j_2]$ of V w.r.t. the class of hv-convex 4-connected binary images;
2: starting out from the first row connect interval-pairs found in Step 1, until all the components of the image are found, such that
a) the rows of the intervals of H are consecutive and pairwisely disjoint, and
b) the columns of the intervals of V are pairwisely disjoint;

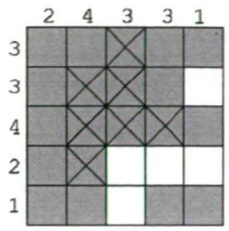
4 Experimental Results

4.1 Implementation Details

We implemented the algorithms of Section 3 in order to study their performance from the viewpoint of running-time and reconstruction quality.

The kernel-shell algorithm was implemented with two different data structures. We designed an *array data type* where we represented the set of core and shell positions by two two-dimensional arrays of size $m \times n$. The core positions were marked by 1s, and the shell points by 2s. The second data structure (called *first-last type*) stored the first and the last elements of the core and the shell in each row and each column (or alternatively -1, if there was no element of the set in the given row or column). This structure was suggested in [13] and it uses four one-dimensional arrays of size m and another four one-dimensional arrays of size n. The two data structures are presented in Fig. 4.

The two variants of the kernel-shell heuristic, the algorithm based on Simulated Annealing and the method based on the location of the components were implemented in JAVA, and the test run on an AMD Athlon X2, 2.1 GHz with 2GB RAM under Ubuntu 11.01. In the experiments we used a large set of hv-convex images,



$$\text{core}=\begin{bmatrix}0&0&1&0&0\\0&1&1&0&0\\0&1&1&1&0\\0&1&0&0&0\\0&0&0&0&0\end{bmatrix}\quad \text{shell}=\begin{bmatrix}2&2&2&2&2\\2&2&2&2&0\\2&2&2&2&2\\2&2&0&0&0\\2&2&0&2&2\end{bmatrix}$$

Core:	Shell:
RowFirst: 3,2,2,2,-1	RowFirst: 1,1,1,1,1
RowLast: 3,3,4,2,-1	RowLast: 5,4,5,2,5
ColumnFirst: -1,2,1,3,-1	ColumnFirst: 1,1,1,1,1
ColumnLast: -1,4,3,3,-1	ColumnLast: 5,5,3,5,5

Figure 4: Example image, where × represents the core and the gray pixels form the shell. The state is represented both by the array and the first-last data types.

with different sizes (from 10×10 to 50×50) and varying number of components (from 1 to 4), by picking them from uniform random distributions, using the methods of [2, 3].* For each size and number of components the data set contained 100 images. Since the algorithm based on Simulated Annealing uses random values, for each test data we repeated the reconstruction 5 times, and took the average speed and correctness of the five runs. In the following we present results just for a part of the test images, but we made the same observations by investigating the entire data set.

4.2 The Quality of the Reconstructions

First, we studied the quality of the reconstructions. Even if the image to be reconstructed is hv-convex, there can be a significant difference between two images having the same horizontal and vertical projections, due to the presence of the switching components (see, e.g., Fig. 5). To measure the error of reconstruction, we computed the conventionally used RME (relative mean error) [17] with the formula

$$RME = \frac{\sum_{i,j} |m^o_{i,j} - m^r_{i,j}|}{\sum_{i,j} m^o_{i,j}}\ , \tag{9}$$

where $M^o = (m^o_{i,j})$ is the original binary matrix (the expected image), and $M^r = (m^r_{i,j})$ is the binary matrix of the reconstruction.

The mean value and the variance of the RME for some of the data sets are given in Table 1. Note that the reconstruction quality of the kernel-shell method does not depend on the applied data structure. We can observe that – due to the fact that

* We used the benchmark collection available at `http://www.inf.u-szeged.hu/~pbalazs/benchmark/benchmark.html`

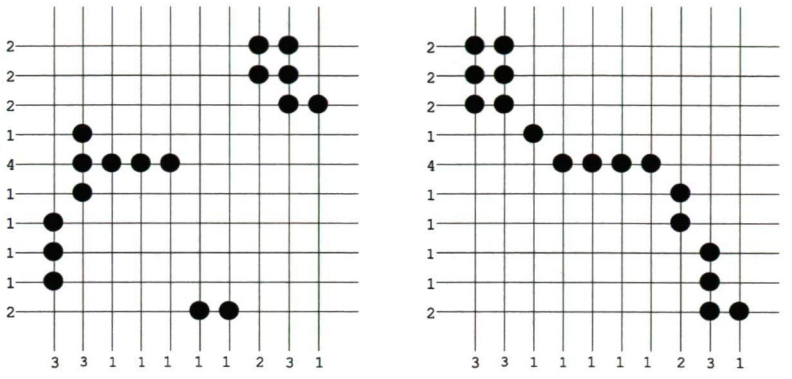


Figure 5: Two different hv-convex discrete sets with the same projections.

the SA algorithm does not guarantee hv-convexity – the quality of its results are usually worse than those of the other two methods. The core-shell method usually outperforms the algorithm based on the location of the components, although not significantly. However, all algorithms show an increasing trend in the RME value (yielding worse and worse quality) with the increasing number of components of the image. We found that images consisting of fewer components have usually significantly fewer switching components than those with more components, which could be an explanation for this trend.

Table 1: Quality of the reconstructions in RME as a function of the number of components (first column), and the size of the image (horizontally)

Core-Shell - Array/First-last data type										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	mean	variance	mean	variance	mean	variance	mean	variance	mean	variance
1	0.0181	0.0918	0.0027	0.0128	0.0002	0.0017	0.0002	0.0017	0.0001	0.0008
2	0.1005	0.3577	0.0138	0.0406	0.0062	0.0186	0.0320	0.1712	0.0114	0.0999
3	0.2669	0.5426	0.0483	0.1130	0.0173	0.0433	0.0668	0.2929	0.1234	0.3405
4	0.7000	0.7730	0.1171	0.2279	0.0570	0.1608	0.1044	0.4083	0.1890	0.5122
Simulated Annealing										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	mean	variance	mean	variance	mean	variance	mean	variance	mean	variance
1	0.3771	0.3239	0.5606	0.3573	0.6905	0.3711	0.7091	0.4536	0.6965	0.3957
2	0.7919	0.5172	1.0104	0.4949	1.0682	0.5091	1.1263	0.5371	1.1255	0.5326
3	0.9133	0.6889	1.1019	0.6027	1.2006	0.5825	1.2678	0.6125	1.2609	0.5963
4	1.0685	0.8217	1.3180	0.6548	1.3653	0.6394	1.3086	0.6827	1.2806	0.6590
Algorithm Based on the Location of the Components										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	mean	variance	mean	variance	mean	variance	mean	variance	mean	variance
1	0.0123	0.0401	0.0022	0.0094	0.0033	0.0019	0.0023	0.0014	0.0012	0.0009
2	0.0382	0.1105	0.0055	0.0257	0.0126	0.1002	0.0303	0.2217	0.0058	0.2406
3	0.1852	0.3217	0.0290	0.0676	0.0346	0.1772	0.1380	0.4023	0.1466	0.5303
4	0.5757	0.6457	0.0839	0.3402	0.1123	0.4233	0.1560	0.6073	0.2964	0.7169

It is easy to see that if the $X \in \mathcal{BM}(H,V)$ image is hv-convex, then

$$f(X) = \max\{f(M)|M \in \mathcal{BM}(H,V)\} = \sum_{m}^{i=1}(h_i - 1) + \sum_{n}^{j=1}(v_j - 1) \ . \tag{10}$$

However, SA can also produce non-hv-convex images, which occasionally can have better RME values than others which are not hv-convex (see Fig. 6 for an example). For this reason, we also calculated how the criteria of convexity defined in (6) is satisfied. Table 2 depicts for each reconstructed set the value of (6) divided by its possible maximum determined by (10). From this table it becomes evident that – regarding the SA based algorithm – not only the RME value but also the convexity of the reconstructed image gets worse and worse, as the size of the image and the number of its components is increasing.

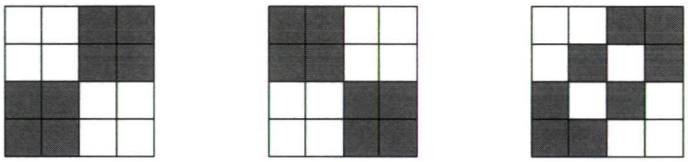


Figure 6: Original image (a). An hv-convex solution with RME=1 (b). A non-hv-convex image (c) with the same projections but better RME value (RME=0.5) than image (b).

Table 2: Quality of the reconstruction measured as the fraction of the number of adjacent ones and its possible maximum

Number of adjacent ones - Simulated Annealing										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	mean	variance	mean	variance	mean	variance	mean	variance	mean	variance
1	0.94	0.04	0.89	0.06	0.86	0.07	0.86	0.09	0.87	0.07
2	0.87	0.05	0.79	0.06	0.77	0.06	0.76	0.05	0.76	0.06
3	0.87	0.05	0.77	0.06	0.73	0.06	0.71	0.06	0.70	0.05
4	0.87	0.05	0.72	0.05	0.69	0.05	0.68	0.06	0.69	0.06

We also analyzed the effectiveness of the switching operators in the SA based algorithm. We calculated the difference of $f(S) - f(R)$ and again, divided by the maximum value given by (10), where S and R denoted the final reconstruction and the initial solution matrix provided by the Ryser algorithm, respectively. The results are shown in Table 3. We can deduce that the more components the discrete set has, the greater the difference is in quality between the initial and the final solutions. This again, can be justified by the observation, that – owing to the more switching components in the image – Ryser's algorithm gives a worse initial solution regarding the hv-convex property, in case of bigger number of components.

Table 3: Quality of the correction as a function of the number of components (first column) and the size of the image (horizontally)

Quality of the correction - Simulated Annealing										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	mean	variance	mean	variance	mean	variance	mean	variance	mean	variance
1	0.08	0.05	0.06	0.04	0.05	0.03	0.05	0.04	0.04	0.03
2	0.18	0.07	0.10	0.04	0.09	0.03	0.08	0.03	0.08	0.03
3	0.22	0.07	0.13	0.04	0.10	0.03	0.09	0.02	0.10	0.02
4	0.27	0.08	0.16	0.04	0.12	0.03	0.10	0.02	0.10	0.02

4.3 Running Time of the Reconstructions

Table 4 shows the running time (clear CPU usage, without garbage collecting) of the algorithms. First of all, the running time of the SA based algorithm is more or less independent of the number of components. It is rather influenced by the cooling schedule and the number of the switching components which increases with the size of the image. Furthermore, the two versions of the core-shell algorithm show similar performance. Images with 1 or 2 components can be reconstructed much faster than images with 3 or 4 components. In both implementations, the running time increases rapidly both by increasing the size of the set, and the number of its components. On the contrary, the speed of the algorithm based on the location of the components is mostly influenced by the size of the image, and not by the number of components. Finally, the implementation with the array data type is much faster than the one using the first-last data structure. The reason is, that this latter one needs more calculation for processing the new kernel and shell, when a heuristic step is taken.

One more observation is that in case of the kernel-shell method the size of the components has an important effect on the speed of the reconstruction. If the image contains a relatively big component, then it is more likely that in the first step a non-empty kernel can be produced, yielding less or no need for using the time-costly stack operators. In case of smaller components, the stack must be used more often for guessing, which means many steps of backtracking and a lot of execution time (see Fig. 7).

Finally, it is worth to note that the average time of the reconstruction with the kernel-shell method and the algorithm based on the location of the components can significantly differ with the same components but aligning them in different ways. Figure 8 shows just an example in case of images of size 30×30 with 4 components, but we found similar trends in any other test cases.

Table 4: The average and the variance of the running times in milliseconds

Core-Shell - Array data type										
	10		20		30		40		50	
	average	variance	average	variance	average	variance	average	variance	average	variance
1	39.73	4.92	52.82	11.33	74.77	37.12	147.81	126.75	246.45	325.99
2	42.53	4.20	71.63	45.65	146.87	97.71	334.67	351.82	735.52	1172.44
3	45.08	4.87	102.21	83.98	341.86	478.53	872.21	1341.80	1526.40	2783.91
4	47.33	13.61	292.23	402.12	1032.04	2333.96	1697.22	2264.28	3094.40	3277.27
Core-Shell - First-Last data type										
	10		20		30		40		50	
	average	variance	average	variance	average	variance	average	variance	average	variance
1	37.67	1.87	50.16	9.54	73.98	27.93	124.15	74.56	215.61	143.69
2	42.22	3.99	80.52	34.43	166.89	103.44	530.95	534.85	1349.71	1467.54
3	45.82	4.99	169.31	138.09	570.69	495.88	1719.58	1574.60	2223.90	2043.90
4	45.41	5.88	755.18	824.58	2006.66	2155.59	2934.82	2408.63	5889.01	4992.17
Simulated Annealing										
	10		20		30		40		50	
	average	variance	average	variance	average	variance	average	variance	average	variance
1	145.36	63.91	319.39	154.14	492.51	248.76	673.49	351.06	852.51	456.82
2	182.08	72.76	421.86	168.49	638.97	266.82	839.62	367.29	1053.75	461.08
3	184.24	69.54	386.23	143.50	612.47	245.05	832.27	334.91	1037.32	419.95
4	182.63	60.67	366.21	117.53	571.41	201.29	752.02	274.17	940.79	352.44
Algorithm based on the location of the components										
	10		20		30		40		50	
	average	variance	average	variance	average	variance	average	variance	average	variance
1	1.07	0.43	2.03	1.32	1.66	12.45	2.86	3.74	9.05	71.72
2	1.64	1.79	7.01	14.22	44.86	128.45	77.67	325.58	446.63	945.98
3	3.02	2.48	91.56	423.13	217.65	505.17	267.18	510.26	687.54	1591.19
4	4.47	10.07	356.46	790.07	436.50	1214.05	575.66	507.71	962.66	2212.41

5 Conclusion

In this paper we studied different algorithms for reconstructing hv-convex binary images. Knowing from theory that the task is NP-hard, we found that the difficulty of the problem depends on several factors. In case of the core-shell algorithm, the reconstruction speed and quality is in connection with the size of the image, and the number, the position, and the size of the components. The efficiency of the SA based algorithm depends on the number of the switching components in the image. Finally, the reconstruction efficiency of the algorithm based on the location of the components is mostly determined by the size of the image, and the number and position of the components. Thus, a fast algorithm for reconstructing hv-convex binary images must somehow combine the core-shell operators, the localization of the components, and the switching operators, too. In a future work we also intend to investigate how prior information on the number and size of the components can facilitate the reconstruction. We believe that the deeper insight we gained through our experiments can help us to design more efficient reconstruction algorithms for this class, in the near future. We also hope that this knowledge could also reveal the difficultness of the reconstruction in other classes of binary images, too.

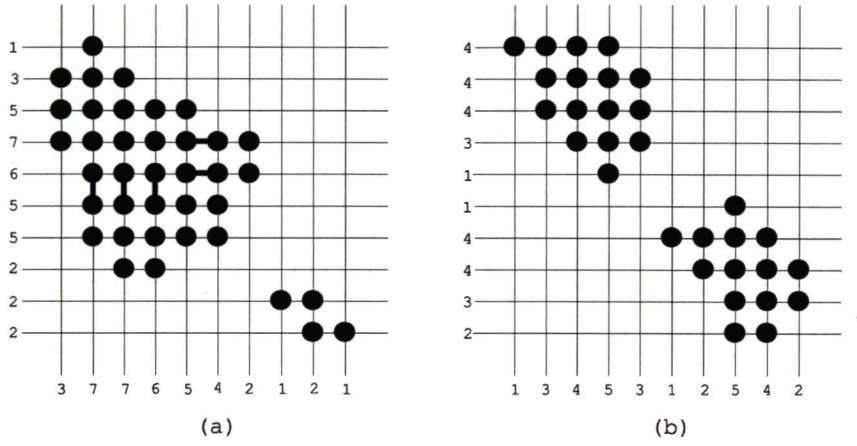


Figure 7: Two different images with the same number of components. (a) When the core-shell algorithm can give an initial non-empty kernel (indicated by thick lines), and (b) when the algorithm cannot give an initial non-empty kernel.

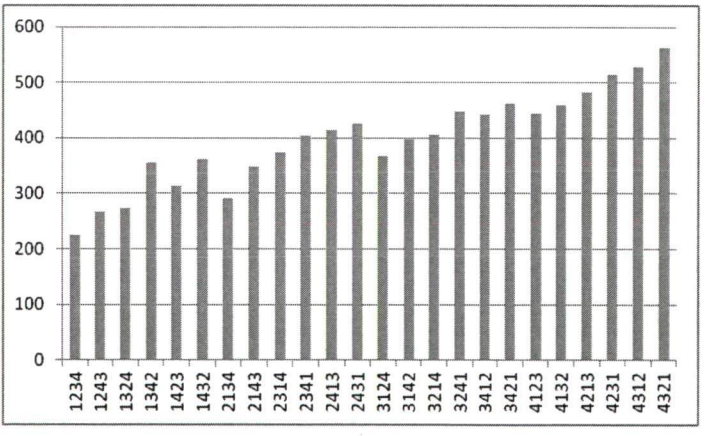


Figure 8: Average reconstruction time of the core-shell algorithm (with array data type) in milliseconds for images of size 30×30 with 4 components allocated them in different orders. Permutations on the horizontal axis indicate how the components are positioned relatively to each other. Components are numbered from top to bottom, and the permutation shows their orders from left to right.

References

[1] P. Balázs, Discrete tomographic reconstruction of binary images with disjoint components using shape information, *International Journal of Shape Modeling* **14:2** (2008) 189–207.

[2] P. Balázs, A benchmark set for the reconstruction of hv-convex discrete sets, *Discrete Appl. Math.* **157** (2009) 3447–3456.

[3] P. Balázs, A framework for generating some discrete sets with disjoint components by using uniform distributions, *Theoret. Comput. Sci.* **406** (2008) 15–23.

[4] E. Barcucci, A. Del Lungo, M. Nivat, R. Pinzani, Reconstructing convex polyominoes from horizontal and vertical projections, *Theor. Comput. Sci.* **155** (1996) 321–347.

[5] S. Brunetti, A. Del Lungo, F. Del Ristoro, A. Kuba, M. Nivat, Reconstruction of 4- and 8-connected convex discrete sets from row and column projections, *Lin. Algebra Appl.* **339** (2001) 37–57.

[6] M. Chrobak, C. Dürr, Reconstructing hv-convex polyominoes from orthogonal projections, *Inform. Process. Lett.* **69** (1999) 283–289.

[7] G. Dahl, T. Flatberg, Optimization and reconstruction of hv-convex (0,1)-matrices, *Discrete Appl. Math.* **151** (2005) 93–105.

[8] G.T. Herman, *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*, 2nd edition, Springer, 2009.

[9] G.T. Herman, A. Kuba (Eds.), *Discrete Tomography: Foundations, Algorithms and Applications*, Birkhäuser, Boston, 1999.

[10] G.T. Herman, A. Kuba (Eds.), *Advances in Discrete Tomography and Its Applications*, Birkhäuser, Boston, 2007.

[11] F. Jarray, G. Tlig, A simulated annealing for reconstructing hv-convex binary matrices, *Electronic Notes in Discrete Mathematics* **36** (2010) 447–454.

[12] A.C. Kak, M. Slaney, *Principles of Computerized Tomographic Imaging*, IEEE Service Center, Piscataway, NJ., 1988.

[13] A. Kuba, Reconstruction of two-directionally connected binary patterns from their two orthogonal projections, *Computer Vision, Graphics, and Image Proc.* **27** (1984) 249–265.

[14] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Equation of state calculation by fast computing machines, *J. Chem. Phys.* **21** (1953) 1087–1092.

[15] H.J. Ryser, Combinatorial properties of matrices of zeros and ones, *Canad. J. Math.* **9** (1957) 371–377.

[16] G.J. Woeginger, The reconstruction of polyominoes from their orthogonal projections, *Inform. Process. Lett.* **77** (2001) 225–229.

[17] A. Kuba, G.T. Herman, S. Matej, A. Todd-Pokropek: Medical applications of discrete tomography, Discrete Mathematical Problems with Medical Applications, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS*, 55:195–208 (2000).

Acta Cybernetica **21** (2013) 165–176.

Performance Metrics Based Mobile Resource Management*

Krisztián Pándi† and Hassan Charaf†

Abstract

Mobile computer technology has greatly evolved in the recent years. Cloud computing is recognized to be a new area for solving performance issues. Mobile terminal can take advantage from cloud computing. To cope with these new resources and fulfill new quality and performance requirements a more sophisticated architecture and resource management is necessary. The basis of effective resource management is a precise knowledge of the hardware and software capabilities. Performance metrics serve as an input for resource management. This study will present architecture of mobile resource management using cloud resources. The main task of such resource management is to decide which application where to run; on the mobile terminal or in the cloud. This paper identifies key components of resource management, settles the tasks and relationships between them.

Keywords: performance metrics, cloud, mobile, resource management

1 Introduction

A mobile terminal can use cloud for solving performance issues and to obtain richer user experience. The aim of this study is to present an architecture for mobile resource management, that can benefit from cloud computing. Performance measurement and usable metrics are necessary for our later research: decision making mechanism implementation. The goal of the mechanism is to decide where is the optimal place for a certain service/application to run; on the mobile terminal itself or on public cloud computing server. Hence a performance and usage of the mobile terminal should be determined.

Cloud computing promises [5] to provide high performance, flexible and low cost on- demand computing services. Emerging complexity of the application used in

*This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TÁMOP-4.2.2.C-11/1/KONV-2012-0013).

†Budapest University of Technology and Economics, Department of Automation and Applied Informatics, E-mail: {Pandi.Krisztian,Charaf.Hassan}@aut.bme.hu

mobile terminals implicate harnessing these extra performance resources. Applications with distributed components differ from traditional non distributed applications in numerous attributes, such as communication type and overhead, latency, concurrency etc.

The task of the proposed mobile terminal resource and service management is to decide where an application or service should be executed. To effectively fulfill this complex task a sophisticated and dedicated decision formula is needed. This formula uses dedicated software and performance metrics. Mobile terminal coupled with distributed system can be dynamic, changing over time, resulting CPU and network load changing. Therefore mobile terminal as a part of the distributed hierarchy needs to have very different metrics than traditional software and performance metrics. With mobile computer technology progress, the software and hardware platform becomes more and more complex, together with the amount of the tasks meant to be processed. Mobile terminals have some special features in comparison with traditional computing; small size, dependence on limited battery lifetime, computing power is changing, possible presence of 3D hardware, network bandwidth is limited, and almost exclusively wireless, relatively small display size and special user input.

Usually similar applications are used in mobile terminals and in traditional computers thus similar user experience is expected. Therefore, with comparably less performance nearly the same look and feel is required. In consequence of that capabilities of the mobile hardware should be efficiently harnessed with smart resource management and load balancing.

The main contributions of this paper are: (i) discussion on applicable mobile performance measurement and metrics ; (ii) recommendation to gather the characteristics of mobile phone usage and creating a user specific profile; (iii) discussion on the architecture of mobile resource management, which uses cloud computing resources to enhance the user experience of the mobile terminal.

The rest of this paper is organized as follows: Section II presents the related work. In Section III we discuss the performance measurement methodology, based on that we recommend a performance measurement and profile creation layer in Section IV. In Section V we discuss the architecture for mobile resource management. Finally, in Section VI questions for the future work is described.

2 Related work

The need for measuring performance of computing machines emerged early, and solutions exist for this problem. Early days were dedicated for creating standardized benchmark programs, such as SPEC [9] or EEMBC [11] etc. Common attributes of these benchmarks are the batch style execution, what is measuring program executing time and speed. The less is the executing time the higher is the system performance. All of these benchmarks are measuring not only the program execution, but the operating system itself, with its interrupts, caching etc, what make the result ambiguous. This batch like execution does not simulate fully the

everyday operation and usage; it is not detailed enough. Other benchmarks see the optimal metrics in picking commonly used applications from the application pool, and running them one by one. Although this is only a derivative of previous solution, there is a possibility to run these applications parallel. To summarize: currently available performance metrics mainly come from traditional computing world.

Adaptive Mobile Systems solutions proposed a model, where applications or services are traveling from performance lack devices to device in idle. Adaptation is an application attribute to change its behavior when surrounding environment changes (CPU load, battery power etc.). Adaptation strongly depends on current system performance, so some efforts are made to define metrics. One solution is to measure CPU load, and battery power, and send it to the application for decision making [14], as battery time is a key point in mobile terminals. Online collection of dynamic software metrics (number of invocations, and response time) was considered [13]; a drawback is that the application code must be changed to insert measurement code. In [15] metrics for service oriented systems are proposed, namely size, coupling, performance, and resource utilization.

Effective resource managements static and standard goal is to extract as much performance as possible from available hardware and software base. As technology changed, the resource management methods and focus may change from time to time as well. Efforts are made to research this field; mobile cloud computing middleware is presented in [16] focusing on service and service bus. Computer service performance is presented in [7], numerical modeling the response time of the services.

A summarization work on cloud and grid computing is presented in [12]; most interesting part is when it states it is often impractical to assume such detailed priori knowledge of management policies for all the resources will be available to resource brokers in a large and dynamic heterogeneous environment. Recommendation is to use resource management "in the dark". Another article [4] quantitatively compares the layered queuing and historical techniques including thoughts on how they could be combined. In [17] resource management mechanism for clouds is described, with compound framework that integrates hierarchical structure and P2P architecture and combines their characteristics. The mentioned paper is mainly focusing on traditional cloud topology with high bandwidth network access. Although some methods can be used for mobile architecture.

A potential synergy between mobile terminal and cloud is proposed in [10], it is found that cloud deployed applications that employ mobile devices as end-points are particularly exciting due to the high penetration of mobile devices. The value of the article in complex approach to both end of the cloud, the cloud itself and the possible endpoint (mobile as well).

Authors of [8] created a framework which partitions the workload of complex services in a distributed environment and keeps the Web service interfaces on mobile devices. Article describes a service like approach, assuming that every application can be executed as a web service. Our approach is similar to this, but with following difference:

- All the functions/applications are partly executed locally on mobile terminal, thus resources of the mobile device are always employed. In our opinion it must not be always the case. Video content can be streamed, gps content can be sent via network etc.

- Only heavy duty tasks are executed in the cloud. In our architecture, we do not make such a difference between tasks.

An interesting approach is the CloneCloud [6], a system that automatically transforms mobile applications to benefit from the cloud. It contains a flexible application partitioner and execution runtime that enables unmodified mobile applications running in an application-level virtual machine, placed in the cloud. This approach is transforms single-machine execution into distributed execution automatically, but part of the process must run on the mobile. A cloned virtual machine with virtual hardware is needed, with complex profiler, migration handler. Our architecture does not recommend such a granular execution, where threads and their content are travelling from Cloud to mobile terminal. We propose data migration rather than thread migration, and that full jobs would execute in one place, not partial part of the jobs.

Neither of the method focuses on profile creation, what can be beneficial during resource optimization. The proposed resource management strongly rely on information coming from profile.

3 Performance measurement - methodology

Performance measurement is a rather complex task, as we have seen in chapter II. and there is a decision to be made which path to follow. One approach is to measure pure performance of certain hardware component, namely CPU, GPU, storage, network bandwidth etc. This gives us a very good detailed picture of capability of given hardware component. Obviously in a complex system, there is no clean testing of component, because system parts strongly depend on each other. This dependency must be handled; impact can theoretically be minimized with careful design, dependency can be taken into account saying it comes with the method. Despite mentioned drawbacks, this method is suitable for checking basic capabilities of the hardware itself. From performance metric point of view, these tests have very limited usability; global throughput of the system often cannot be predicated from atomic parts of the system. Although, it can be used if the service or application component strongly depends on hardware component. For this a good real life example is hard to give. Taking games as an example (they are highly performance consuming applications), it can be seen that they do not exclusively depend on GPU capabilities of the hardware. Game application also depends on input peripheries, storage, or network bandwidth of a device. Other example is media streaming; network is a strong factor of the overall performance, beside CPU properties, not to mention storage size and speed. Cloud computing

is a good example where these separate benchmarks come in hand; to have a good service management mechanism, basic performance characteristics must be known.

Another approach is to measure performance on application base. The basic idea is to collect commonly used applications, and run them in a batch way, one after the other, in specific order. Problems with this approach are:

- how to decide which application to run in this benchmark
- can a common application basket be selected from numerous application pools
- which applications and how long to run
- how to assign a weight to an application, to calculate a final score
- what consequences can be derived from different application run for the new application

For commonly used applications this performance measurement is suitable for comparison, but for a new, or custom application it does not add much to our knowledge. Additional problem of this performance measurement method is that there is no knowledge about real application running on the device. This can be avoided if on the fly measurements are implemented on the device. We suggest this method; a data collector can create a profile, based on the mobile terminal usage. The concept is that long term measurement is implemented on the mobile device, which monitors the terminal usage. The data is collected and evaluated, and based on that a real life benchmark can be created and collected.

Other not frequently mentioned aspect of the performance measurements is the multithreaded multi-core environment. Todays mobile hardware CPU is multi-cored, and this must be included in the performance measurements. The simplest way is to gain performance measurement, to launch the same application twice, and see its behavior, or simply launch the application and see whether it scales with multiple-core. Data collection is also meaningful, because the application usage can be monitored, together with applications that are running in parallel, giving the resource management mechanism usable basic data.

4 Performance measurements

One of the focus of this study is what kinds of performance measurements are mandatory for effective application management. The management needs to decide where certain application should run, on mobile terminal or in the computing cloud. Special challenge of cloud computings resource management is that the application/service must run and finish in the fastest way. Driving factor of the resource management is to gain speed in application calculation and running time; enhanced user experience is expected. Decision has to be made how this requirement can be achieved. At first glance it seems that every application must run in

cloud (as [8] recommends for example), because that will lead to highest user experience. Unfortunately this is not always true. Cloud computing has a bottleneck, it can have a huge computing capacity, but there are several criteria to harness it:

- all data must be present in the cloud computing environment
- all data will be calculated in cloud computing environment
- data travel from client (currently mobile terminal) to server, or to client from server is expensive from performance point of view.
- client should be online

Effective cloud computing usage depends on the available network connection quality, mostly on latency and available bandwidth. Although network bandwidth is increasing, it cannot keep up with the CPU performance and storage capacity; the gap is opening. And even if several megabits are present in wired connection, wireless connection will have more limited bandwidth. A good benchmark will test the available bandwidth in longer term, to assist to effective decision making mechanism. Long term measurement can have more benefits; time and locality dependent map can be made containing data about available bandwidth, the time interval for this bandwidth can be used etc. With this profile application management can be enhanced. So, the most important metric is the available network bandwidth.

Additional straightforward metrics are; CPU, GPU, storage I/O, keyboard input capacity. For our architecture an important performance metric is the user mobile application usage characteristics. For example, if the user tends to use more application at the same time, it is a good idea to take it into account. User experience is enhanced, if time consuming applications are moved into the cloud, like a background task, while providing more performance to other applications.

For performance measurement currently available solution can be used (e.g. for GPU Basemark [1], GL benchmark [2], WP bench [3] etc.). For missing benchmarks a custom one can be made focusing on certain parts of the hardware. This applies for example to I/O performance of the mobile device storage, because the built in component may vary from model to model. The key is the comparability and the ease together with multi platform implementation.

In Fig. 1 code migration measures are listed. Although it is code and not application specific, it gives us an overview what performance metrics are necessary for further evaluation and decision making. It is worth to mention, that with increasing number of metrics, the decision matrix is getting more and more complex.

The question is what to do with performance measurements results. It is not necessary to test every device of the mobile terminal. An online database stored in the cloud is suitable for effective resource management needs. This online database can hold performance information about the known mobile terminal models, and if a terminal is connecting to cloud, it can be updated and used.

Fig. 2 shows profile creation procedure. Performance measurement results are collected from mobile terminal runtime results, or downloaded from network

Attribute	Metric
Software	Size of executable module (e.g. Java .class file) Size of a serialized object Size of an in-memory object Size of an extra-memory for execution of a method Number of executable statements/instructions Size of serialized parameters Number of method invocations
Performance	Method execution time Method invocation time Instance migration time Class migration time
Resource Utilization	Network bandwidth Network usage Size of available memory Memory usage Processing power Processing usage

Figure 1: Code Migration Performance Measures

database containing device specific information. The database prevents basic device characteristics to be rechecked all the time, for example CPU statistics, GPU presence etc. Performance metrics are handled to profile manager. Application and resource usage history is also collected and provided to profile manager, for further processing and decision making.

Table 1: Application categories based on performance requirement

Applications	Performance requirement			
	CPU	***Storage***	***Network***	***GPU***
Browser + Flash	+++	++	+++	++
Browser No Flash	+	+	++	
Games	+++	++	++	+++
Office	++	+	++	
Image viewer	++	++	++	
Chat text	+		+	
Voice chat	++		++	
Online media content	+++	++	+++	+++
Storage backup with compression	++	+++	+++	
VNS/SSH X	+		++	

Table 1 shows a categorization of applications used in mobile terminals. Some performance requirements are collected and weighted with + sign, showing the extent of the usage. This gives us an overview on performance critical parts. These application can run parallel, or if additional CPU core is available, on different cores. Without user experience dropdown, an application can run parallel, if it does not use the same resource heavily. If a race condition occurs; it will lead to

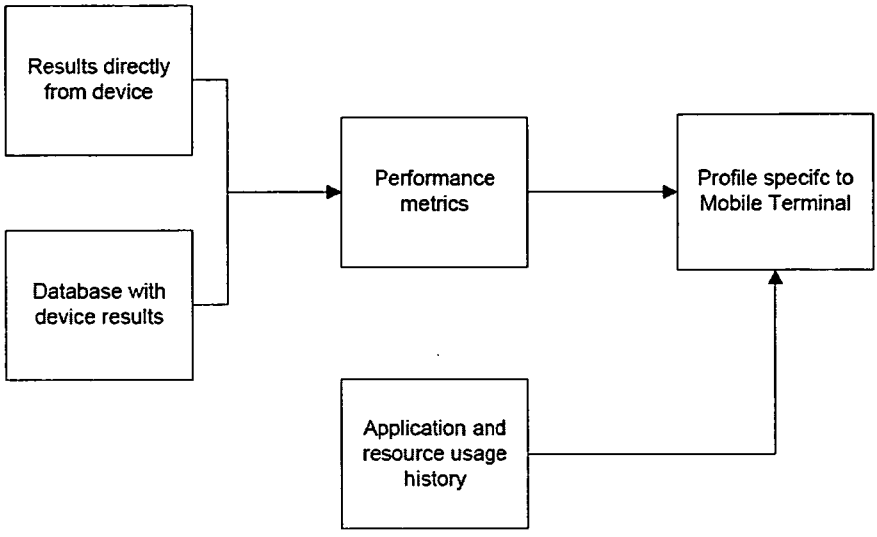


Figure 2: Performance metrics and profile creation

lags and slow interaction.

This paper does not deal with performance measurement of cloud. It is a task of later research, to verify whether it is necessary to verify the performance of the cloud. In this paper we can safely assume that beside the network overhead, cloud has more calculating and storage I/O capacity in comparison to mobile terminal.

5 Resource management in mobile terminal

In the previous chapters the performance measurement of mobile terminal was discussed, which will serve as an input for the resource management. The task of such management layer is to decide where certain application to be run.

In this chapter the architecture of a resource management system will be discussed. The place of the resource management is in the mobile terminal. If no network connection is available, or it is relatively slow, every application will run on the mobile terminal. Mobile terminal must remain usable if some hardware resource is not available or has low performance.

Currently we do not deal with how it can be technically done to have the same application or part of the application available in cloud or in the mobile terminal. It is a scope of future research. We assume that they are present, and every application can run either on the mobile terminal or in the cloud. This hypothesis might seem strong; in related work [12][13] solution for this can be found, although we listed the problematic parts of them. In future the topic will be checked in more detail.

The first problem to deal with; where the data physically is. For applications heavily dependent on input data, or dealing with large data, from performance point of view it is necessary to have the data locally. Mobile terminals have wireless connection, which is relatively slow; network interface should be warily used, to avoid unnecessary and frequent data sending.

As an example an image viewer application will manipulate pictures faster if they are available on local storage, and photos taken with the device are certainly there. Other case if the images are placed in the cloud, that way cloud manipulation can be faster. This leads us to another aspect; security and background synchronization.

Using cloud computing to aid resolving a performance bottleneck in mobile computing has an important aspect; financial cost of the used cloud resource. Detailed discussion of this topic is not the task of this study although the suggested resource management is capable of dealing with it. In the profile the user can define the cost attribute (limit, preferences etc) and the decision is influenced by these factors.

Data can travel from mobile terminal to the cloud if the synchronization is enabled. It can be done when the network is not used, and in this way large data can be transferred to or from cloud unnoticeably. To achieve this, earlier mentioned profile making is necessary. It needs to be known, or at least predicted, when the network will be in idle state, when it is heavily used. Table 1 comes in hand in this work item, it can be known that what type of application is using which resource of the terminal; with that information the smart synchronization can be realized. Another benefit from profile mechanism is that heavily used data can be identified. If a user is normally editing and modifying the attachments of an email, as an example, it is good to keep them on both storage places, to be ready for network unavailability and to allow data to be processed on mobile terminal and cloud as well.

From user profile parallel used application can be identified. This leads to an interesting opportunity; optimization for that type of usage can be done in the resource management. Application place to run can be modified not only based on performance metrics, but based on the user mobile usage. If application is producing large data usually serving for other applications as an input; it is good to run them in the same place. But if there is no large data involved, run them separately, and making some resource free for other applications. Network availability can be taken into account in this case, together with safe data storage.

There is a feature in every operating system, to put processes to background till further usage. This enables the operating system to do some optimization with resources, unload the application partly or totally from memory, and place it to swap memory. Similar feature might be feasible in the resource management too; background task can be moved to or from cloud freeing mobile terminal resource.

Battery usage and state should be monitored during decision making. If the mobile terminal is running out of the battery, application heavily using CPU can be moved to the cloud, regardless of performance degradation. This could be set as a configuration parameter similarly to some operation systems, user could choose between energy schemes, defining the behavior in such case. Performance of the

mobile terminal can be measured on the device itself, or the common data can be purchased form the online database. Profile information, or other user specific, but not so sensitive information must be stored on the mobile terminal. Beside obvious data protection, the resource management layers need this information constantly, even if network connection is not available.

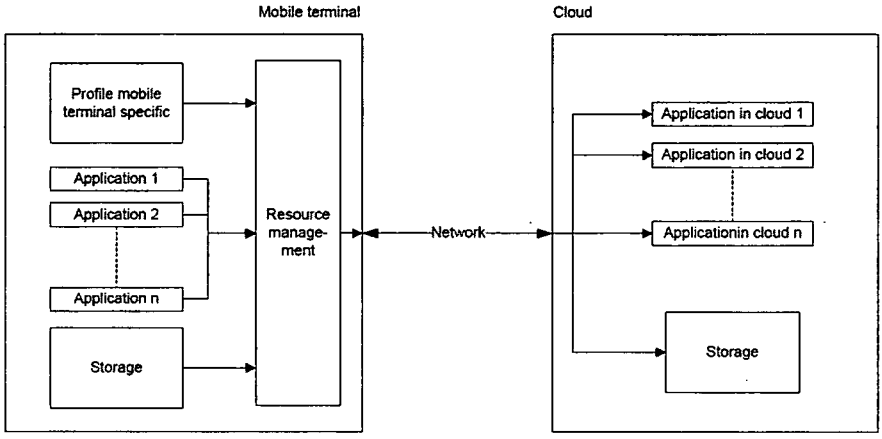


Figure 3: Resource management architecture

Fig. 3 shows the resource management architecture with key parts. Profile part, what is mobile terminal specific, already contains necessary performance metric information. Profile changes on the fly, with actualized performance information and application usage collected from mobile terminal. If the resource management decides that application will run faster, application from cloud is called, and returned information is transferred transparently to mobile terminal. This happens unnoticeably, the user should feel the speed enhancement from this management. Information is transmitted through available and constantly changing network bandwidth. Storage block represents mobile terminal local data; it can be synchronized automatically to cloud if user enables this feature.

6 Future work

The presented architecture is capable of enhancing the user experience and harnessing extra performance given by cloud computing environment. Realization in test implementation will follow, to confirm resource management architecture. Collectable performance metrics should be limited to extent that serves the effective resource management, without unnecessary data. Cloud computing environment should be checked from performance point of view. The financial cost of the cloud computing should be a part of the architecture as well, how and where is a part

of future work. We assumed that every application can be run in cloud or in the mobile terminal. Technical and theoretical background of this topic will be checked in future work.

References

[1] Basemark. http://www.rightware.com/benchmarking-software/product-catalog/, accessed on Jan. 20. 2013.

[2] Glbenchmark. http://www.glbenchmark.com/benchmarks.jsp, accessed on Jan. 20. 2013.

[3] Wp bench. http://www.windowsphone.com/hu-hu/store/app/wp-bench/e447e949-01c0-43f1-8b65-76d752d7d305, accessed on Jan. 20. 2013.

[4] Bacigalupo, D.A. Resource management of enterprise cloud systems using layered queuing and historical performance models. In *2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pages 1–8, April 2010.

[5] Buyya, Rajkumar. Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 1–, Washington, DC, USA, 2009. IEEE Computer Society.

[6] Chun, Byung-Gon, Ihm, Sunghwan, Maniatis, Petros, Naik, Mayur, and Patti, Ashwin. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[7] Gani, Hendrik, Dr, Supervisor, and Ryan, Caspar. The impact of runtime metrics collection on adaptive mobile applications. 2005.

[8] Hassan, Mahbub, Zhao, Weiliang, and Yang, Jian. Provisioning web services from resource constrained mobile devices. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 490–497, Washington, DC, USA, 2010. IEEE Computer Society.

[9] J.L.Henning. Spec cpu2000: Measuring cpu performance in the new millenium. *IEEE Computer*, 33(7):28–35, 2000.

[10] Lehman, Tobin J. and Vajpayee, Saurabh. We've looked at clouds from both sides now. In *Proceedings of the 2011 Annual SRII Global Conference*, SRII '11, pages 342–348, Washington, DC, USA, 2011. IEEE Computer Society.

[11] Levy, M. Evaluating digital entertainment system performance. *IEEE Computer*, 38(7):68–72, 2005.

[12] Majumdar, Shikharesh. Resource management on clouds and grids: challenges and answers. In *Proceedings of the 14th Communications and Networking Symposium*, CNS '11, pages 151–152, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[13] McGregor, John D., Cho, Il-Hyung, Malloy, Brian A., Curry, E. Lowry, and Hobatr, Chanika. Collecting metrics for corba-based distributed systems. *Empirical Softw. Engg.*, 4(3):217–240, September 1999.

[14] Noble, Brian D., Satyanarayanan, M., Narayanan, Dushyanth, Tilton, James Eric, Flinn, Jason, and Walker, Kevin R. Agile application-aware adaptation for mobility. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, SOSP '97, pages 276–287, New York, NY, USA, 1997. ACM.

[15] R.Rossi, Z. Tarri. Software metrics for the efficient execution of mobile services. In *in Proc of ECOWS06 Workshop on Emerging Web Services Technology*, December 2006.

[16] Wang, Qian. Software metrics for the efficient execution of mobile services. In *SOA's Last Mile-Connecting Smartphones to the Service Cloud*, pages 80–87, September 2009.

[17] Xingye, Han, Xinming, Li, and Yinpeng, Liu. Research on resource management for cloud computing based information system. In *Proceedings of the 2010 International Conference on Computational and Information Sciences*, ICCIS '10, pages 491–494, Washington, DC, USA, 2010. IEEE Computer Society.

Acta Cybernetica 21 (2013) 177–190.

x86 Instruction Reordering for Code Compression

Zsombor Paroczi*

Abstract

Runtime executable code compression is a method which uses standard data compression methods and binary machine code transformations to achieve smaller file size, yet maintaining the ability to execute the compressed file as a regular executable. With a disassembler, an almost perfect instructional and functional level disassembly can be generated. Using the structural information of the compiled machine code each function can be split into so called basic blocks.

In this work we show that reordering instructions within basic blocks using data flow constraints can improve code compression without changing the behavior of the code. We use two kinds of data affection (read, write) and 20 data types including registers: 8 basic x86 registers, 11 eflags, and memory data. Due to the complexity of the reordering, some simplification is required. Our solution is to search local optimum of the compression on the function level and then combine the results to get a suboptimal global result.

Using the reordering method better results can be achieved, namely the compression size gain for gzip can be as high as 1.24%, for lzma 0.68% on the tested executables.

Keywords: executable compression, instruction reordering, x86

1 Introduction

Computer programs, more precisely binary executables are the result of the source code compiling and linking process. The executables have a well defined format for each operating system (e.g. Windows exe and Mac binary [6, 15]) which usually consists of information headers, initialized and uninitialized data sections and machine code for a specific processor instruction set.

Compression of executables is mainly used to reduce bandwidth usage on transfer and storage needs in embedded devices [11]. Even today Linux kernels for embedded devices are stored in a compressed file (so called bzImage) [1], and every program for Android comes in a compressed format (apk) [20]. The literature distinguishes between two different approaches whether the produced compressed

*Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, E-mail: paroczi@tmit.bme.hu

binary remains executable without any additional software or hardware or not. In the first case the resulted binary includes the decompression algorithm and some bootstrapping instructions for restoring the original binary during runtime within the memory of the device. In the second case, the decompression is done on the operating system level or by special hardware, which has a predefined compression method with built in parameters. In this case the decompression method and the additional dictionaries is not an overhead in each binary, which allows decompression algorithms to use bigger dictionaries. If the decompression is hardware based, the performance can be significantly improved, which can also limit pure software based solutions.

Runtime executable code compression is a method which uses standard data compression methods and binary machine code transformations to achieve smaller file size, yet maintaining the ability to execute the compressed file as a regular executable. In our work we focus on machine code compression for Intel x86 instruction set with 32bit registers for both type of code compression approaches. It is important to note that the same method with minor modifications should work on each instruction set.

Various compression methods for the x86 machine code have been developed over the past years. Most of them use model based compression techniques (such as Huffmann coding, arithmetic coding, dictionary-based methods, predication by partial match and context tree weighting) with CPU instruction specific transformations such as jump instruction modification [2, 7]. These compression algorithms are lossless, so the binary before compression and after decompression is identical. In most cases the CPU instruction specific transformations are also reversible, the most common transformation is the jump instruction modification. x86 long jump instruction is 5 byte long. The first byte identifies the instruction, the last 4 is a relative jump address. The transformation modifies the jump address from relative to absolute before compression and does the inverse after decompression. This transformation may help producing smaller compressed binaries because most of the addresses used in x86 are absolute, so the transformed jump instructions have a better chance for model based matching and range encoding.

The compiled machine code is usually in one section of the executable, a continuous chunk of raw data, with a few known entry points. There is no clear indication, where each function or even instruction begins, it all depends on the actual execution. A function is a sequence of instructions in one continuous block with the following limitation: jump and call instructions from one function can transfer execution either into another part of the same function or to the first instruction of another function. If functions are called from outside, they always return and restore the stack according to the used call convention.

Very few compression methods try to identify functions. Most of them use binary pattern matching for known instructions - such as first byte match for jump transformation. When the executable is generated by a "standard" compiler certain patterns can help to identify functions in the code - such as stack protection stubs (push ebp; mov ebp, esp). Using a disassembler, an almost perfect instructional and functional level disassembly can be generated [21]. Using the structural information

of the compiled machine code, each function can be split into so called basic blocks, sequences of instructions including zero or one control transfer instruction at the end of the block [3, 9].

In our work we demonstrate that the reordering of instructions within a basic block can improve code compression without changing the behavior of the code. Our approach searches for local compression optimums on the function level for each function than combines these results to achive a suboptimal global result. With this novel approach we achieved the compression size gain for gzip can be as high as 1.24%, for lzma 0.68% on the tested executables.

2 Related work

Compression techniques for executables have been investigated by a number of researchers. Research directions can be categorized into two different groups: modification of the compiled (and sometimes not linked) machine code for smaller size or better compression, and different compression methods and models for the specific task of executable compression.

In the first category researchers tried to exploit the potentials of the same methods used in compilers, which resulted in aggressive inter-procedural optimalization of repeated code fragments, but without actual data compression methods. To increase the matching code segments the register renaming and local factoring transformation was used [11]. Using data dependency with instruction reordering was presented in the PLTO framework, which used the conservative analysis to gain performance with optimizing cache hit, instruction prefetching and branch predication [23]. Basic block reordering, without modifying the block itself, is also used to gain performance, mainly due to branch predication [18].

Reusing multiple instances of identical code fragments have been also exploited in the Squeeze++ framework, but it was done on various levels of granularity: procedures, code regions with unique entry and exit points, basic blocks and partially matched basic blocks [24, 8, 9].

Some works focus on the structural analysis of the whole program graph to identify common patterns [5, 10] or some other ways to simplify the structure of the graph [14, 5, 9] or even discover non-trivial equivalence in it [10, 13]. These methods focus on the control flow nature of the machine code, on the basic block level, they do not attempt to modify the code using the data flow information.

Refined compactation methods (unreachable code elimination, whole function abstraction, duplicate code elimination, loop unrolling, branch optimalization) can be used in various fields, which require specific tasks such as operation system kernel compactation [13] and ARM specific compactation [10].

For simplicity, all of these methods used additional link-time data to help separating the pointers (relocation data) from constant values, our method doesn't rely on such data. All of these compactation methods aim for smaller file size by modifying the machine code, yet maintaining the executability. But non of these methods exploit the results of the data compression research field, in fact some of these

compactations have a negative effect, if the machine code is later compressed [12].

Compression method evaluation and modification is also a widely researched field. Recompressing Java class files with various compression favorable modification resulted in significant gains, modifications include custom opcodes and string reordering, with the gzip compressor used as a black-box compressor [22]. Without further knowledge, in compiled machine code stream compressors, like gzip, are widely used as a black-box compressor, including android executables and linux kernels [5]. Compression can benefit from the known structure of the data, split-stream compression techniques conceptually split the input stream of instructions into separate streams, one for each type of instruction field. This lossless compression strategy can improve the compression rate significantly [19]. The same concept of separating the machine code into multiple context models can be used with instruction reordering, which can be compressed as non-sequential data [7]. In systems with limited memory, such as embedded systems, decompression methods have more limitations. Some algorithms decompress functions on demand, which means that the functions are decompressed on the first access [5].

All of these compression methods either use data compression as a black-box to simply compress the resulted binary as a stream, or use some special heuristic to supposedly encourage redundancy. This will always result in a suboptimal solution, neither of these works experiment with all the possible reorderings, due to the complexity. In our solution, the granuality of the search is fixed on the function level, but all the possible reorderings are tested, resulting the function level optimum of the compression.

The most closely related work to ours is PPMexe [12]. In this work the reordering is done globally (A1 algorithm) and on basic block level (A2, A3 algorithm), but the goal is to improve the predication rate of PPM by maximizing the number of n-symbol context occurrences. PPM highly depends on split-stream coding, a reversible transformation, which separates the structural information of an instruction into different streams. Our approach uses gzip and lzma and treats the machine code as one stream, searching for local optimum on the function level.

3 Reordering instructions

Original basic block	Reordered basic block
mov eax, ebx	**mov ecx, edx**
mov ecx, edx	**mov eax, ebx**
add eax,ecx	add eax,ecx
ret	ret

Table 1: Instruction reordering

Instructions have a predefined order in the executable, but in fact, every instruction could be executed once every required parameter is available. This statement is referred as data dependency. The basic idea behind reordering is that changing

the instruction order within a basic block will produce different raw data, which could lead to different compressed size, that may be smaller than the compressed original one. An example of reordering can be seen in Table 1. The original and the reordered basic block are functionally equal, after running both on various input the effect (observable registers and flags) are the same for both code sequence. The byte representation of the blocks in this example differ in 2 bytes. The original byte sequence is 6689d86689d16601c8c3 in hexadecimal notation while the reordered byte sequence is 6689d16689d86601c8c3, which can lead to better compression - for example if other parts of the code section has the pattern of d86601.

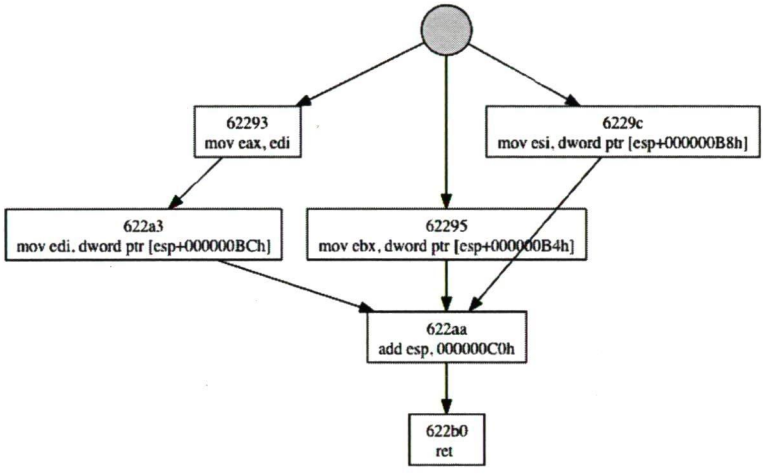


Figure 1: Data dependency graph inside a basic block

To ensure that the reordered block has exactly the same effect as the original one, the data dependency relations must be analyzed. The data usage for each instruction can be calculated using a disassembler [21], and from the dependencies a local data flow graph can be produced for each basic block. (See Figure 1.)

$$\begin{Bmatrix} CS: \\ DS: \\ SS: \\ ES: \\ FS: \\ GS: \end{Bmatrix} \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{Bmatrix} + \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix} * \begin{Bmatrix} 1 \\ 2 \\ 4 \\ 8 \end{Bmatrix} \right] + [\text{displacement}] \right] \tag{1}$$

The x86 instruction set in user mode uses 8 data registers (EAX, EBX, ECX, EDX, EBP, ESI, EDI, ESP), the instructions can access memory using a strict addressing mode (1). Due to the memory protection methods provided by modern operating systems, segment registers (CS, DS, SS, ES, FS, GS) are usually not used

in user space programs. It is safe to ignore those along with debug registers and control registers.

The x86 instruction set uses a special status flag register which contains the current state of the processor represented as different binary flags stored as bits of the register. These bits are used to store extra information about the last instruction and can be used to manipulate the outcome of the next instruction. For example the multiply instruction sets the overflow flag if the result cannot be represented on the current register due to overflow, the overflow flag can be tested in a conditional jump which may transfer execution to some error handling code.

Memory access uses a well defined formula but identifying whether two pointers are the same or not is impossible in the majority of the cases. For example [eax+14] , [00401000h] and [esp+14h] could point to the same address or to three completely different address. Some disassemblers, like Hex-Rays' IDA[1], can identify trivial cases for identical memory pointers but the analysis is complicated and should be done on the whole code section [21]. In our work we decided to treat the whole memory as one entity, the so called memory data. This simplifies the dependency model because every memory location can be treated as a single virtual register which can be used for both reading and writing. It is required to handle memory writes and reads because it will cause further data dependencies in the code.

We distinguish two kinds of data affection (read and write) and use 20 data types including registers: 8 basic x86 registers, 11 binary flags in the eflags register, and memory as a whole. Each instruction has a well defined data usage [16], which defines which registers / flags are used in different instructions, and how these registers affect the output. For simplicity each control flow instruction (such as condition, unconditional jump and call instructions) should be treated as if they write every data type. This ensures that during reordering within a basic block the control flow instructions remain at the end of the block.

The following rules are used to generate the data dependency :

A instruction should be before B instruction (considering original order A before B):

- if A reads any data type B writes, or
- if A writes any date type B writes, or
- if A writes any data type B reads.

The reordered and the original code have the same data flow graph, which means that they have the same effect (observable registers, flags and memory), only the control flow graph can be different, but at the end of each basic block the executed instructions have the same quantity, and only the execution order may vary [5, 12].

An example of basic block instruction data dependency is demonstrated in Figure 1. Each instruction is indicated by a separate box, the original position in the first line, and the actual instruction in the second line. The arrows represent data dependencies within the block. All the indirect dependencies are hidden. The circle marks the entry point of the basic block.

[1] http://www.hex-rays.com/products/ida/index.shtml (Last accessed: 2013-01-15)

4 Compression and permutation count

Most of the lossless data compression algorithms are designed to exploit statistical redundancy in the data. Reordering basic blocks could change the data statistics which may improve compression. Without any assumption about the actual data compression method we will consider a non-zero compression time which grows at least linearly with the size of the input data. To determine which input is the most compressible, the compression method should be executed on each input variation, then the one with the smallest compressed data size should be chosen.

The possible number of different reorderings for a basic block are bounded by $k!$, where k is the number of instructions within the block, but this number could be significantly lower because of the data flow constraints. Using the following formula the total permutation count can be calculated:

$$\prod_{i}^{N} \prod_{j}^{M_i} n_{i,j} \tag{2}$$

where N is the total number of functions, M_i is the number of basic blocks within the i^{th} function and $n_{i,j}$ is the permutation count for the j^{th} basic block in the i^{th} function. The best compression can be achieved only by testing all reorderings, which gives the global optimum. Due to the huge permutation count and the non-zero compression time this could be done only for very small files.

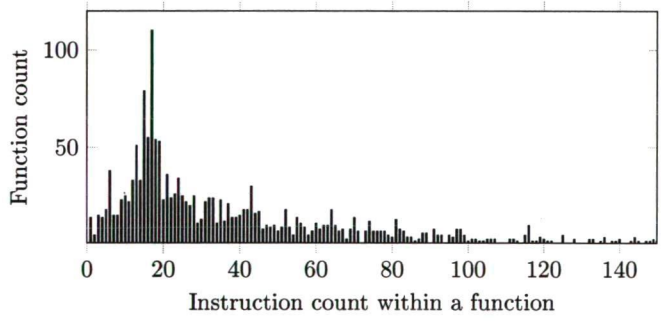


Figure 2: Function count by instruction count

Common programming methodology and practices suggest that the complexity (possible execution path count - branches) of each function should be relatively low. This way the source code can be easy to understand, maintain and test. During compilation the control flow can change after inlining/outlining functions but the distribution of the instruction count for functions does not change significantly. Analyzing a sample dataset which was taken from the libc system executable, a histogram can be created (Figure 2.), which shows the instruction count in functions. As expected, there are a lot of functions with only few instructions, the average function consists of 57 instructions, and only a couple of functions have more than 100 instructions.

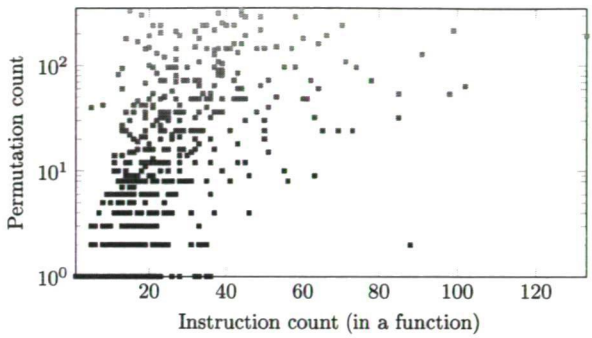


Figure 3: Permutation count

The permutation count (with valid data dependency) for each function was also calculated. The results can be seen on Figure 3. By increasing the instruction count, the permutation count of a function is also increasing. There is no common rule for the permutation count growth, low instruction count functions may also have a lot of permutations (e.g. "__strcspn_g" function in the dataset, 30 instructions, 76.951.350 possible permutations), and long functions may have only a few permutations (e.g. "getpass" function in the dataset, 133 instructions, 192 permutations). The permutation count heavily depends on efficient data type usage.

To keep the permutation count at a manageable level instead of searching for the global optimum in the compressibility local optimum places should be considered. The local optimum search is done on the function level, where each function is evaluated separately and every basic block permutation is tested within the function.The search algorithm is detailed in pseudo code in Algorithm 1.

first basic block perm. #1	second basic block perm. #1	third basic block perm. #1
first basic block perm. #2	second basic block perm. #1	third basic block perm. #1

...

first basic block perm. #K	second basic block perm. #1	third basic block perm. #1
first basic block perm. #1	second basic block perm. #2	third basic block perm. #1

...

Figure 4: Local optimum search iterations

This means that the optimal reordering for a selected function is calculated by compressing all the possible reorderings on the function level. In each iteration one

Algorithm 1 Pseudo code for the global suboptimum search

```
result=empty
for each function do
  perms=calculate every permutation of basic blocks in the function
  for each perms do
    compress the selected permutation
    if this is the smallest compressed size then
      incode=the uncompressed function
    end if
  end for
  append incode to result
end for
```

of the basic blocks gets another permutation and the whole function is recompressed and tested. As shown on Figure 4., this can be done with a simple limited counter function. This way the optimum search problem can be calculated in a distributed way, the algorithm should not have any information regarding the jump/call target for the control flow instructions.

In a basic block there is only one (if any) control flow instruction. This instruction is always at the end of a basic block. Among x86 instructions only control flow instructions have relative to current address pointers, that is why reordering instructions can be done by simply changing the instruction's order.

5 Implementation

In our implementation the function splitting is done by starting from the entry point and dynamic library export addresses then tracing the code using a disassembler and following each control flow edge with a depth-first-search. On each call instruction target a new function splitting point must be created. This way most of the code section gets analyzed, and using function split points the code can be split into functions. This result is double checked with the constraint for functions introduced in the second section, and if needed, the result gets modified.

Splitting functions into basic blocks are done by identifying the control flow instructions within a function, then splitting the code after the control flow instructions and finally at the point before the jump target points. This way the constraint for basic blocks can be granted (an example is shown in Figure 5., where 9 basic blocks have been identified in the example function). According to data flow analysis among these basic blocks only two can be reordered at the instruction level. The stack is heavily used to store data, which results in a large number of memory writes and reads.

The data dependency analysis is done using the free open source x86 disassembler called BeaEngine[2]. A binary dependency matrix gets defined which defines the

[2]http://www.beaengine.org/ (Last accessed: 2013-01-15)

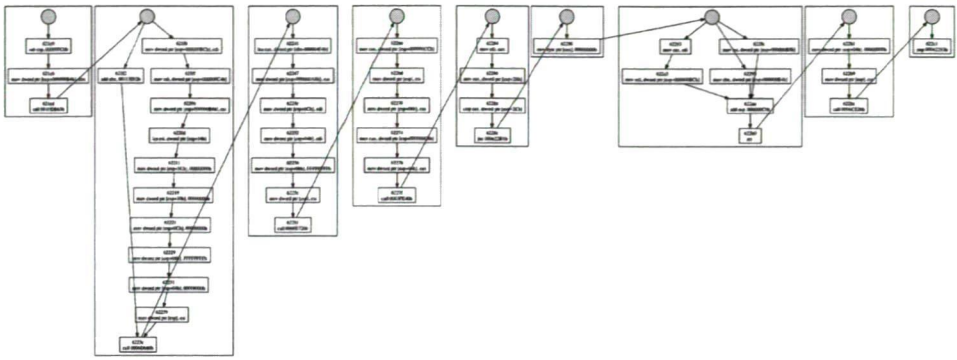


Figure 5: Data flow and basic blocks in a function

instruction order constraints. In the matrix the item $n_{i,j}$ is *true* if the i^{th} instruction within the basic block must be before the j^{th}. This constraint is transitive, so by propagating the items in the matrix, the transitive closure can be generated for easier access of the rules.

Some functions can still have a lot of permutations therefore only functions with less than 2.000 possible permutations were tested due to the computational limits. In the example shown on Figure 5. the 2nd block has 11 different permutations, the 7th block has 12 different permutations, so overall 132 cases have been tested.

The latest stable gzip[3] (version 1.3.12) and lzma[4] (version 5.0.3) software were used to compress the produced code. The parameters used for compression are:

- gzip: -9 (best compression)
- lzma: -e (extreme compression)

6 Results

For the sample function shown on Figure 5. the original code was 195 bytes compressed with gzip and 176 bytes compressed with lzma. Using reordering the compressed size was decreased by 2 bytes using gzip and 3 bytes using lzma, so the original code was not optimally ordered for compression purposes as shown on Figure 6.

To evaluate the explained method and the implementation, several files were tested from various compilers and operating systems. The filenames, operating systems, compilers and sources for the testfiles are shown in Table 2. In case of node.js, due to the high number of functions, only the first 1000 were tested. These binaries are commonly used on each operating system. All of these programs / libraries are written on a high level language (usually C++), which means that a

[3]http://www.gzip.org/ (Last accessed: 2013-01-15)
[4]http://tukaani.org/xz/ (Last accessed: 2013-01-15)

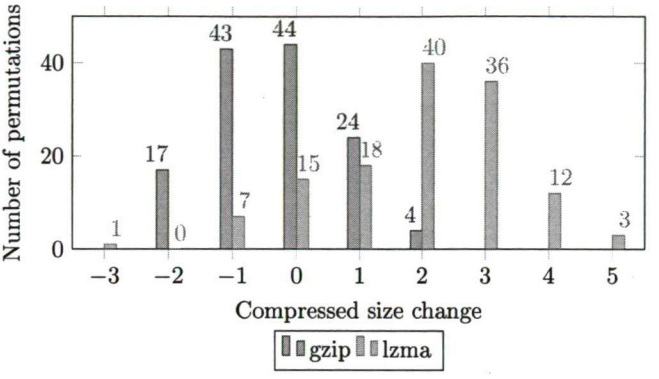


Figure 6: gzip and lzma compression results

Name	OS	Compiler	Source (Last accessed: 2013-01-15)
libc-2.13.so (-0ubuntu13.1)	Ubuntu	gcc	http://packages.ubuntu.com/natty/libc6-i386
unzip (6.0-4)	Debian	gcc	http://packages.debian.org/squeeze/unzip
libconfig.dll (1.4.8)	Windows	VS2008	http://www.hyperrealm.com/libconfig/
node.js (0.8.8)	Mac	llvm	http://nodejs.org/dist/latest/node-v0.8.8-darwin-x86.tar.gz

Table 2: Source of the files used in tests

compiler generates and optimizes the executable part of the binary. This excludes macro optimization which usually done at the machine code level by a human expert.

For verifying the statement, that the reordered code is functionally equal to the original, we performed several unit tests on the reordered functions. During the tests there were no major performance regressions which may arise due to reordering of code which was optimized for speed.

Name	Code section in bytes		Compressed size in bytes without reordering		Compressed size in bytes with reordering		Gain percentage	
	all	processed	gzip	lzma	gzip	lzma	gzip	lzma
libc-2.13.so	413.619	110.944	46.353	39.848	45.778	39.576	1.24%	0.68%
unzip	74.905	5.012	2.933	2.944	2.903	2.924	1.02%	0.67%
libconfig.dll	17.123	8.982	4.128	3.952	4.127	3.944	0.02%	0.20%
node.js	303.248	93.544	39.554	33.688	39.451	33.616	0.26%	0.21%

Table 3: Compression results

For evaluating the compression result, the non-machine code part of each file have to be omitted, the result should be compared using the reordered and the original data in compressed form. In Table 3. the result for compression tests can be seen, the function level statistics are in Table 4. Using the reordering method

Name	Method	Byte gain per function	
		Avg.	Std. dev.
libc-2.13.so	gzip	2.666	3.663
libc-2.13.so	lzma	0.993	2.094
unzip	gzip	1.288	1.766
unzip	lzma	1.244	2.227
libconfig.dll	gzip	0.583	1.096
libconfig.dll	lzma	0.366	1.269
node.js	gzip	0.902	1.650
node.js	lzma	0.543	1.604

Table 4: Detailed compression results

better results can be achieved: the compression size gain for gzip can be as high as 1.24%, for lzma 0.68%.

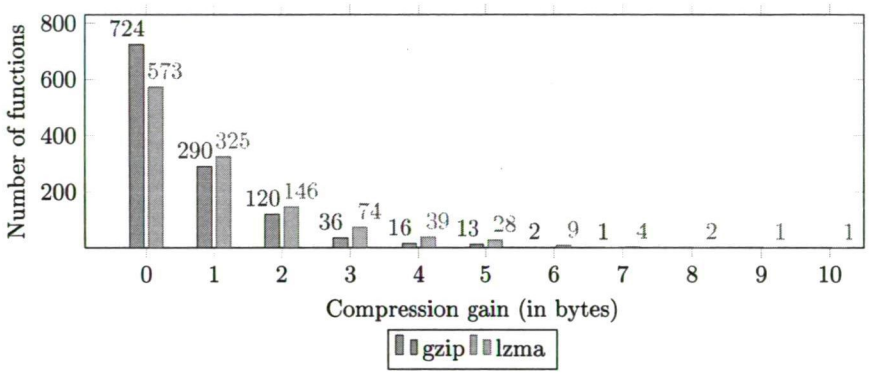


Figure 7: Compressed code size change

A detailed statistics on compression gain can be seen on Figure 7, these results are for libc-2.13.so. In this case more than 40% of all the functions can have a better compressible reordering than the original one but the compression gain is usually small (1-3 bytes/function) but significantly bigger gains are also possible (5-10 bytes / function).

7 Future work

Future work will include evaluation of other code transformation methods used together with instruction reordering, such as pattern based instruction substitution [14, 4, 17] and also exploiting the basic principles of the split-stream compression. The data types used in this work can be also refined using much more sophisticated analysis on memory access, especially on the stack. Instead of local

optimum search on function level, other search methods or optimum criteria should be considered, such as genetic algorithms.

References

[1] Linux kernel 2.6.30 changelog. Online. Last accessed: 2013-01-15. http://kernelnewbies.org/Linux_2_6_30.

[2] Beszédes, Árpád, Ferenc, Rudolf, Gyimóthy, Tibor, Dolenc, André, and Karsisto, Konsta. Survey of code-size reduction methods. *ACM Comput. Surv.*, 35(3):223–267, September 2003.

[3] Bruening, Derek L. Efficient, transparent and comprehensive runtime code manipulation, 2004.

[4] Bruschi, Danilo, Martignoni, Lorenzo, and Monga, Mattia. Code normalization for self-mutating malware. *IEEE Security and Privacy*, 5(2):46–54, March 2007.

[5] Chanet, Dominique, De Sutter, Bjorn, De Bus, Bruno, Van Put, Ludo, and De Bosschere, Koen. Automated reduction of the memory footprint of the linux kernel. *ACM Trans. Embed. Comput. Syst.*, 6(4), September 2007.

[6] Corporation, Microsoft. Microsoft portable executable and common object file format specification. Online. Last accessed: 2013-01-15. http://msdn.microsoft.com/en-us/library/windows/hardware/gg463119.aspx.

[7] Dai, Wenrui, Xiong, Hongkai, and Song, Li. On non-sequential context modeling with application to executable data compression. In *Data Compression Conference, 2008. DCC 2008*, pages 172 –181, march 2008.

[8] De Sutter, Bjorn, De Bus, Bruno, and De Bosschere, Koen. Sifting out the mud: low level c++ code reuse. *SIGPLAN Not.*, 37(11):275–291, November 2002.

[9] De Sutter, Bjorn, De Bus, Bruno, and De Bosschere, Koen. Link-time binary rewriting techniques for program compaction. *ACM Trans. Program. Lang. Syst.*, 27(5):882–945, September 2005.

[10] De Sutter, Bjorn, Van Put, Ludo, Chanet, Dominique, De Bus, Bruno, and De Bosschere, Koen. Link-time compaction and optimization of arm executables. *ACM Trans. Embed. Comput. Syst.*, 6(1), February 2007.

[11] Debray, Saumya K., Evans, William, Muth, Robert, and De Sutter, Bjorn. Compiler techniques for code compaction. *ACM Trans. Program. Lang. Syst.*, 22(2):378–415, March 2000.

[12] Drinić, Milenko, Kirovski, Darko, and Vo, Hoi. Ppmexe: Program compression. *ACM Trans. Program. Lang. Syst.*, 29(1), January 2007.

[13] He, Haifeng, Trimble, John, Perianayagam, Somu, Debray, Saumya, and Andrews, Gregory. Code compaction of an operating system kernel. In *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '07, pages 283–298, Washington, DC, USA, 2007. IEEE Computer Society.

[14] Hundt, Robert, Raman, Easwaran, Thuresson, Martin, and Vachharajani, Neil. Mao – an extensible micro-architectural optimizer. In *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '11, pages 1–10, Washington, DC, USA, 2011. IEEE Computer Society.

[15] Inc., Apple. Os x abi mach-o file format reference. Online. Last accessed: 2013-01-15. https://developer.apple.com/library/mac/#documentation /DeveloperTools/Conceptual/MachORuntime/ Reference/reference.html.

[16] Intel. Intel 64 and ia-32 architectures software developer manuals. Online. Last accessed: 2013-01-15. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.

[17] Kumar, Rajeev, Gupta, Amit, Pankaj, B. S., Ghosh, Mrinmoy, and Chakrabarti, P. P. Post-compilation optimization for multiple gains with pattern matching. *SIGPLAN Not.*, 40(12):14–23, December 2005.

[18] LIU Xian-Hua, YANG Yang, ZHANG Ji-Yu CHENG Xu. A basic-block reordering algorithm based on structural analysis. *Journal of Software*, 2008/19:1603–1612, 2008.

[19] Lucco, Steven. Split-stream dictionary program compression. *SIGPLAN Not.*, 35(5):27–34, May 2000.

[20] Morrill, Dan. Inside the android application framework. 2008.

[21] Paleari, Roberto, Martignoni, Lorenzo, Fresi Roglia, Giampaolo, and Bruschi, Danilo. N-version disassembly: differential testing of x86 disassemblers. In *Proceedings of the 19th international symposium on Software testing and analysis*, ISSTA '10, pages 265–274, New York, NY, USA, 2010. ACM.

[22] Pugh, William. Compressing java class files. *SIGPLAN Not.*, 34(5):247–258, May 1999.

[23] Schwarz, Benjamin, Debray, Saumya, Andrews, Gregory, and Legendre, Matthew. Plto: A link-time optimizer for the intel ia-32 architecture. In *In Proc. 2001 Workshop on Binary Translation (WBT-2001*, 2001.

[24] Sutter, Bjorn De, Bus, Bruno De, Bosschere, Koen De, and Debray, Saumya. Combining global code and data compaction. Technical report, 2001.

Acta Cybernetica 21 (2013) 191–203.

Geometric Newton-Raphson Methods for Plane Curves

Gábor Valasek*, Júlia Horváth, András Jámbori, and Levente Sallai

Abstract

Our paper reviews Kallay's results on a geometric version of the classic Newton-Raphson method, in the context of plane curve queries, e.g. curve-curve intersection, point-curve distance computation. Variants of the geometric Newton-Raphson methods are proposed and empirically verified.
Keywords: curves, distance, intersection

1 Introduction

Plane curves are fundemantal tools in many applications, ranging from being building blocks of aesthetically pleasing figures [5][9], to defining NC machine tool paths [3] or highway roads [4].

Many applications require certain queries to be carried out on these curves, e.g. finding the closest point of the plane curve to a given point in the plane, or finding the intersection of two plane curves.

These queries can be formulated as systems of nonlinear equations, and to answer one of these queries, the corresponding system has to be solved. A popular choice for solving such systems of equations is the Newton-Raphson (NR) method.

The NR method starts from an initial guess and refines it iteratively, producing a sequence of guesses converging to the roots of the equations, provided an appropriate initial guess was used.

The NR method creates a linear approximation of the problem at each guess, and the next guess is the solution of this linear approximation.

Kallay proposed a geometric Newton-Raphson iteration in [1]. At each step, the curves in the query are substituted by higher order geometric approximants at the current guess, and then the query in question is solved on these and the guess refinement makes use of the solution on the geometric approximant.

Our goal was to empirically compare the traditional and the geometric NR method, and to propose variants of Kallay's method that retain its robustness and low iteration counts, at a reduced computational cost, if possible.

*E-mail: valasek@inf.elte.hu

The paper is organized as follows. Section 2 briefly compares the traditional and geometric NR algorithms and reviews Kallay's main results. Section 3 details our main contribution, the variants of Kallay's scheme by geometric approximant choice (parabola instead of circle) and construction (derivative-free geometric approximant creation). Section 4 shows the plane curve queries that served as the test problems for the comparison of the various NR methods and that the 3-point circle fitting variant is a viable, derivative-free modification of Kallay's geometric NR method in these problems. Section 5 summarizes the results.

2 Geometric Newton-Raphson methods

The Newton-Raphson (NR) method is a powerful tool for solving nonlinear equations. It defines an iterative process, which given a suitable starting point, converges to the roots of the equations. Each step of the iteration refines the current guess by using the solution of a linear approximation of the problem at the current guess.

The classic NR method is sensitive to the choice of the initial guess, that is, its convergence is not guaranteed for abritrary starting points. If the initial guess is within a certain neighborhood of an isolated guess, then the NR method produces a sequence of guesses that converges to the root quadratically. If the iteration is started from the neighborhood of a multiple root, then the rate of convergence is linear. [10]

The idea of using higher order approximations within the NR method, instead of a linear one, to gain higher convergence rate has been present in the literature [8]. Let the $(m+1)$th derivative of the function, whose root we are looking for, be bounded. Then the mth degree Taylor polynomial will be an mth order approximation, yielding a convergence rate of $m+1$ (for isolated roots) [1].

Let us consider the case of finding the zeros of an univariate equation.

In the generalized NR method, the solution of a degree m equation is required in the guess-refinement step, making the cases $m > 2$ less practical.

In addition to being computationally feasible, the case of $m = 2$ also has a straightforward geometric interpretation: at each step, an approximating parabola has to be intersected with the x-axis. This version of the Newton-Raphson method has a convergence rate of 3. Lee et. al. [6] and Park et. al. [7] proposed the use of osculating circles instead of parabolas, which are also second order approximations of the curve, however, the computational overhead, resulting from the curvature computations, made their approach less practical. [1]

It was Kallay who noted, that for geometric problems involving plane curves, using osculating circles has advantages when the osculating circles are used to approximate the curves in the query instead of approximating the equations formalizing the query on the curves.

In [1] Kallay proposed a geometric variant of the Newton-Raphson method, which utilizes a transformation that preserves the geometric domain of the original query, yet it retains the attractive quadratic convergence speed properties of the original NR method and improves on robustusness (at the expense of computational

cost, required due to the construction of geometric approximations).

Kallay's geometric NR method used geometric approximants to the original curves at each guess, and solved the queries on these. Then the results on the proxies were transformed back to the domain of the original curves to compute the new guess.

Let $I \subset \mathbb{R}$ be an open set and $\mathbf{p}(t) : I \to \mathbb{R}^2$ a smooth, regular parametric plane curve. Let us consider the following simple, general pseudo-code for the Newton-Raphson methods:

```
x = InitGuess( p(t) )

while (not IsDone( p(t), x ))
{
  x = NextStep( p(t), x )
}
```

InitGuess() creates the initial guess for the root of the function $f(t)$, $f(t)$ being the mathematical formulation of the geometric query on the plane curve $\mathbf{p}(t)$.

IsDone() is the termination condition of the NR iteration, which will be discussed in more detail in section 4.

NextStep() is the guess-refinement step. In the case of the classic NR method, it would return $x - \frac{f(x)}{f'(x)}$.

For geometric NR methods, NextStep() can be defined as follows:

```
NextStep( p(t), x )
{
  proxy gp = ProxyCreate( p(t), x )
  proxy_solution = ProxySolveQuery( gp, x )
  return x + ProxyReparam( p(t), x, gp, proxy_solution )
}
```

ProxyCreate() constructs a geometric proxy for the curve, which approximates $\mathbf{p}(t)$ at the current guess x. ProxySolveQuery() computes the result of the query on the geometric proxy and ProxyReparam() transforms the solution back into the original curve's domain.

3 Variants of the geometric Newton-Raphson methods

In this section we overview the line and circle proxy creation based on differential geometry, required for Kallay's geometric NR method.

To be able to provide an empirical comparison of the use of different geometric proxies, we propose a parabola proxy creation heuristic.

It must be noted that the curvature computation comes with an overhead which makes the geometric Newton methods more expensive to implement. To alleviate

this, we propose line and circle proxy creation heuristics, that do not require the evaluation of the derivatives of the curve.

3.1 Line proxies

Line proxies were used in Kallay's geometric Newton method when the curvature of the curve $\mathbf{p}(t)$ became 0 at the current guess x_n, x_n being a regular point of $\mathbf{p}(t)$. In this case, the tangent line was used as the proxy for $\mathbf{p}(t)$.

The line proxy can be represented by a point $\mathbf{p}_0 \in \mathbb{R}^2$ and a tangent direction vector $\mathbf{v}, |\mathbf{v}| = 1$. Its parametric form is $\mathbf{l}(t) = \mathbf{p}_0 + t\mathbf{v}$.

At the current guess x_n, $\mathbf{p}_0 = \mathbf{p}(x_n)$. We used the following two tangent direction definitions in the tests:

- Direction computed from the derivative of the curve: $\mathbf{v} = [\mathbf{p}'(x_n)]$, as Kallay
- Direction estimated from forward differences: $\mathbf{v} = [\mathbf{p}(x_n + h) - \mathbf{p}(x_n)]$, so that the derivaties do not have to be evaluated

where $[\mathbf{a}] = \frac{\mathbf{a}}{|\mathbf{a}|}$.

The second version comes from the geometric definition of the tangent line [2]: keeping $\mathbf{p}(x_n)$ fixed, the tangent line is the limit of the lines passing through $\mathbf{p}(x_n)$ and $\mathbf{p}(x_n + h)$ as $h \to 0$.

3.2 Circle proxies

The circle proxy can be represented by one of its points $\mathbf{p}_0 \in \mathbb{R}^2$, its normal $\mathbf{n}, |\mathbf{n}| = 1$ pointing towards the center of the circle, and its radius $r > 0$.

Kallay detailed the use of osculating circles as geometric proxies. The signed curvature of a smooth parametric curve can be computed as [3]

$$\kappa(x_n) = \frac{(\mathbf{p}'(x_n) \times \mathbf{p}''(x_n)) \cdot \mathbf{z}}{|\mathbf{p}'(x_n)|^3},$$

where $\mathbf{z} = \mathbf{x} \times \mathbf{y}$ denotes the unit vector perpendicular to the plane of the curve, $\mathbf{x}, \mathbf{y}$ being the orthonormal basis vectors of the plane of the curve.

If $\kappa(x_n) \neq 0$, the radius of the osculating circle of $\mathbf{p}(t)$ at x_n is $\rho(x_n) = \frac{1}{\kappa(x_n)}$. In this case, the circle proxy can be defined by setting $\mathbf{p}_0 = \mathbf{p}(x_n)$, $r = \rho(x_n)$, and $\mathbf{n}$ to the principal normal of the curve $\mathbf{p}(t)$ at x_n.

It is important to recall that the osculating circle of $\mathbf{p}(t)$ at x_n can be defined geometrically as well [2]: the osculating circle is the limit of the circles through $\mathbf{p}(x_n)$, $\mathbf{p}(y_k)$, $\mathbf{p}(z_k)$, where $y_k, z_k \to x_n$ as $k \to \infty$.

This leads to a circle proxy construction that does not require the computation of curvatures and derivatives:

Consider the circle through the points $\mathbf{p}(x_n - h)$, $\mathbf{p}(x_n)$, $\mathbf{p}(x_n + h)$, $h > 0$. Let us denote its center by $\mathbf{c}$ and its radius by R. Then the circle proxy can be defined by $\mathbf{p}_0 = \mathbf{p}(x_n)$, $\mathbf{n} = [\mathbf{c} - \mathbf{p}_0]$, $r = R$. Later we refer to the circle proxy constructed this way as the 3-point circle proxy.

3.3 Parabola proxies

The parabola proxy is represented by its apex $\mathbf{p}_0 \in \mathbb{R}^2$, an orthonormal frame $\mathbf{t}_0$ and $\mathbf{n}_0$, and $a \in \mathbb{R}$. The parametric form of the parabola we use is then

$$\mathbf{r}(t) = \mathbf{p}_0 + t\mathbf{t}_0 + at^2\mathbf{n}_0.$$

Recall that the curvature of $\mathbf{r}(t)$ at $t = 0$ is $\kappa(0) = 2a$.

An approximating parabola proxy to $\mathbf{p}(t)$ at x_n can be constructed by setting $\mathbf{p}_0 = \mathbf{p}(x_n)$, $\mathbf{t}_0 = [\mathbf{p}'(x_n)]$, $a = \kappa(x_n)/2$, and $\mathbf{n}_0$ to the principal normal of $\mathbf{p}(t)$ at x_n.

4 Testing

4.1 The problem

Let $I \subset \mathbb{R}$ be an open set and let us consider the problem of finding the closest point of a smooth, parametric plane curve $\mathbf{p}(t) : I \to \mathbb{R}^2$, to an arbitrary point of the plane, $\mathbf{q} \in \mathbb{R}^2$.

The above problem can be formulated as finding a curve parameter $t \in I$, such that it minimizes the squared distance

$$d^2(\mathbf{p}(t), \mathbf{q}) = \langle \mathbf{q} - \mathbf{p}(t), \mathbf{q} - \mathbf{p}(t) \rangle, \tag{1}$$

where $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the dot product of $\mathbf{a}, \mathbf{b} \in \mathbb{R}^2$.

Differentiating (1), we find that the parameter $t^* \in \mathbb{R}$ corresponding to the point of the curve that is closest to $\mathbf{q}$ should satisfy

$$\langle \mathbf{q} - \mathbf{p}(t^*), \mathbf{p}'(t^*) \rangle = 0,$$

where the prime denotes differentiation with respect to the curve parameter. Let $f(t) = \langle \mathbf{q} - \mathbf{p}(t), \mathbf{p}'(t) \rangle$. The classic Newton-Raphson method can be used to find the roots of $f(t)$.

Given an initial guess $x_0 \in \mathbb{R}$, let

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{\langle \mathbf{x} - \mathbf{p}(x_n), \mathbf{p}'(x_n) \rangle}{\langle \mathbf{x} - \mathbf{p}(x_n), \mathbf{p}''(x_n) \rangle - \langle \mathbf{p}'(x_n), \mathbf{p}'(x_n) \rangle}, \quad n = 1, 2, \ldots \end{aligned}$$

If x_0 is chosen from within a neighborhood of an isolated root, this yields a quadratically converging iteration. If x_0 resides in a neighborhood of multiple roots, then the rate of convergence is linear. It is important to note, however, that the iteration may not converge at all.

One can plot convergence figures of Newton-maps $N_p(z) = z - \frac{p(z)}{p'(z)}$ with various functions $p(z)$. E.g. Figure 1 shows which initial guesses of the complex plane result

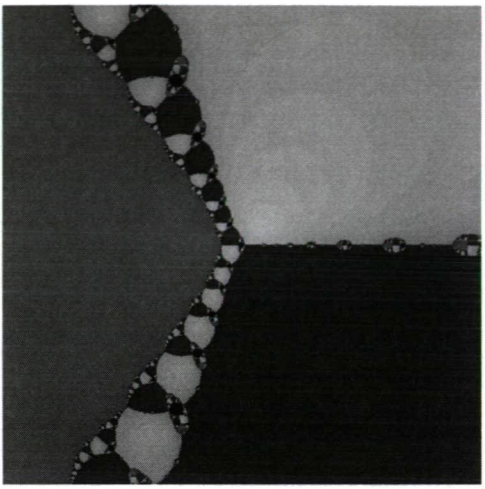


Figure 1: The application of the Newton-map $N_p(z) = z - \frac{p(z)}{p'(z)}$ to the polynomial $p(z) = z^3 - 2z + 2$, $z \in \mathbb{C}$. The red, green, and blue points of the complex plane denote initial guesses that create convergent iterations to different roots. The black points, forming Julia sets, denote elements of the complex plane, that do not yield a convergent Newton-Raphson iteration, if used as initial guesses.

in a convergent Newton-Raphson iteration in the case of $p(z) = z^3 - 2z + 2$, $z \in \mathbb{C}$. The discussion of starting point selection - which also affects the geometric NR methods - falls beyond the scope of our paper, we refer the interested reader to e.g. [10].

4.2 Test framework

The implementation of the NR methods was based on the pseudo-code of the generalized NR algorithm listed in section 2.

Each variant of the Newton-Raphson method uses the same InitGuess method, so that each algorithm starts from the same initial guess. We have taken equidistant parameter values and computed which one creates the closest point on the curve to $\mathbf{q}$ and used it as x_0.

The termination condition IsDone() has to be chosen carefully. Since the traditional NR solution finds a root of the function

$$f(t) = \langle \mathbf{q} - \mathbf{p}(t^*), \mathbf{p}'(t^*) \rangle,$$

the termination condition $|f(x)| \leq \epsilon$ seems to be a reasonable choice at first. However, upon closer inspection, one can find that this condition is very sensitive to the parametrisation of the curve. That is, if the magnitude of $\mathbf{p}'(t)$ over I is bounded and $M = max\{|\mathbf{p}'(t)| : t \in I\}$, $N = max\{|\mathbf{q} - \mathbf{p}(t)| : t \in I\}$, then the

reparametrisation $t \leftarrow \frac{\epsilon}{NM}t$ will make any initial guess $x_0 \in I$ the final guess as well, since

$$\begin{aligned}|\langle \mathbf{q} - \mathbf{p}(\frac{\epsilon t}{NM}), (\mathbf{p}(\frac{\epsilon t}{NM}))' \rangle| &= \frac{\epsilon}{NM}|\langle \mathbf{q} - \mathbf{p}(\frac{\epsilon t}{NM}), \mathbf{p}'(\frac{\epsilon t}{NM}) \rangle| \\ &\leq \frac{\epsilon}{NM}|\mathbf{q} - \mathbf{p}(\frac{\epsilon t}{NM})| \cdot |\mathbf{p}'(\frac{\epsilon t}{NM})| \\ &\leq \frac{\epsilon}{NM} NM = \epsilon.\end{aligned}$$

The iteration never start, since the termination condition is true for any initial guess.

Choosing $|x_{n+1} - x_n| < \epsilon$ for IsDone() instead, to anticipate the slow-down of root refinement, is still parametrisation-dependent, but to a much smaller extent than the previous one.

Given the fact that the domain is geometric in the case of geometric NR methods, one is motivated to find purely geometric termination conditions, which are also parametrisation independent. An example of this is the following:

Let the algorithm stop once the tangent at the current guess is (very close to being) perpendicular to the vector pointing from the current guess to $\mathbf{q}$. In practice, this can be checked more easily by comparing the cosine of the angle between the difference vector $\mathbf{q} - \mathbf{p}(t)$ and $\mathbf{p}'(t)$. Using this, the algorithm stops when $|\langle [\mathbf{q} - \mathbf{p}(t)], [\mathbf{p}'(t)] \rangle| < \epsilon$, where $[\mathbf{a}] := \frac{\mathbf{a}}{|\mathbf{a}|}, \mathbf{a} \neq \mathbf{0}$. Please note, that this is the geometric interpretation of condition $|f(t)| \leq \epsilon$.

The geometric, parametrisation-independent version of condition $|x_{n+1} - x_n| < \epsilon$ would be to stop the iteration if the arc-length between x_n and x_{n+1} is smaller than ϵ. In practice, however, it is more efficient to use $|\mathbf{p}(x_{n+1}) - \mathbf{p}(x_n)| < \epsilon$.

In addition to the above, IsDone should return true when the iteration count has exceeded a certain limit.

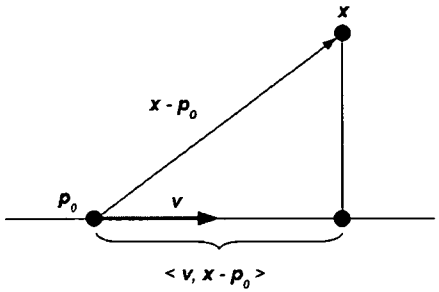
NextStep() is $x = x - \frac{f(x)}{f'(x)}$ in the case of the traditional NR method. The implementation of the geometric NextStep() is detailed in the following subsection.

4.3 Implementation of the geometric NR methods

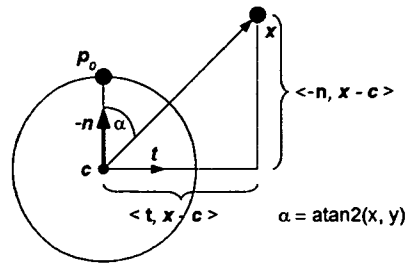
Section 3 has shown the proxy creation strategies for the various proxies, required for ProxyCreate(). In this subsection ProxySolveQuery() and ProxyReparam() are being investigated.

Finding the closest point of a line proxy to a given point in the plane is straightforward. Let ProxySolveQuery() return $t_n = \langle \mathbf{v}, \mathbf{q} - \mathbf{p}_0 \rangle$, as explained in Figure 2a. The closest point $\mathbf{x}$ of the line proxy to $\mathbf{q}$ is $\mathbf{x} = \mathbf{p}_0 + t_n \mathbf{v}$.

In general, the ProxyReparam() function uses the following strategy, as proposed by Kallay [1]: compute the arc-length s_n on the proxy between the current point of the iteration and the solution of the query. Let us estimate the parameter difference required to travel s_n along the original curve! This can be achieved by



(a) Finding the closest point $\mathbf{x}$ of a line, represented by one of its points $\mathbf{p}_0$ and its tangent direction $\mathbf{v}$, to a point $\mathbf{q}$ in the plane



(b) Finding the closest point $\mathbf{x}$ of a circle, represented by one of its points $\mathbf{p}_0$, the unit normal vector pointing from $\mathbf{p}_0$ to the center and the radius r, to a point $\mathbf{q}$ in the plane

Figure 2: Finding the closest point $\mathbf{x}$ of line and circle proxies to a given point $\mathbf{q}$ in the plane.

assuming that the parametric speed along the original curve can be estimated by $|\mathbf{p}'(x_n)|$ sufficiently, and setting $\Delta t = \frac{s_n}{|\mathbf{p}'(x_n)|}$.

In the case of line proxies, ProxyReparam() returns $\frac{t_n}{|\mathbf{p}'(x_n)|}$.

Finding the closest point of a circle to $\mathbf{q}$ requires only elementary geometry as well, see Figure 2b. $\mathbf{c}$ denotes the center of the circle and $\mathbf{t} = [\mathbf{p}'(x_n)]$. The closest point of the circle to $\mathbf{q}$ is the closest intersection of the circle and the line going through $\mathbf{c}$ and $\mathbf{q}$. Then $\alpha = atan2(\langle \mathbf{t}, \mathbf{q} - \mathbf{c} \rangle, \langle -\mathbf{n}, \mathbf{q} - \mathbf{c} \rangle)$. ProxyReparam returns $\frac{r\alpha}{|\mathbf{p}'(x_n)|}$.

There is no elementary solution for this query in the case of parabola proxies. Let (x_0, y_0) be the coordinates of the query point in the coordinate system of the parabola (with origin $\mathbf{p}_0$, and $\mathbf{t}_0$, $\mathbf{n}_0$ as the x and y axes, respectively), $\mathbf{p}(x_n) = (x, y)$. Then

$$d^2(\mathbf{q}, \mathbf{p}(x_n)) = (x - x_0)^2 + (2ax^2 - y_0)^2,$$

from which it follows that

$$d^{2'}(\mathbf{q}, \mathbf{p}(x_n)) = 4a^2x^3 + (2 - 4ay_0)x - 2x_0 = 0$$

has to be solved using the formula for cubic equations.

Since the apex of the parabola is $\mathbf{p}(x_n)$, ProxyReparam returns

$$\frac{2ax\sqrt{4a^2x^2 + 1} + asin(2ax)}{4a|\mathbf{p}'(\mathbf{x_n})|}$$

4.4 Results

The tests consisted of generating 100 random Bezier curves, with random degree between 10 and 100, and 100 random points for each curve. Then the following NR algorithms were used to find the closest point of the curve to the given random point:

- The classic NR method
- Kallay's geometric NR method using osculating circle proxies
- A geometric NR method using tangent line proxies
- A geometric NR method using estimated tangent lines from forward differences
- A geometric NR method using estimated osculating circles (the circle through 3 points of the curve)
- A geometric NR method using osculating parabola proxies

Each method was called three times for every point to investigate the effect of the following iteration termination conditions:

- Condition 0: $|x_{n+1} - x_n| < \epsilon$
- Condition 1: $|\mathbf{p}(x_{n+1}) - \mathbf{p}(x_n)| < \epsilon$
- Condition 2: $|\langle [\mathbf{p}'(x_n)], [\mathbf{q} - \mathbf{p}(x_n]\rangle| < \epsilon$

Everything else was the same for all methods, including initial guesses, comparison and error thresholds.

For each method, we collected the following data: the ratio the method yielded a convergent iteration within 100 steps, the average amount of steps until a root is found in the case of convergent iterations, the standard deviation of the average amount of steps (for convergent cases), and the success rate (how many times did the given method found the closest point on the curve to q compared to the other methods).

The focus of our interest in the curve-point distance tests was the average amount of steps required to finish the iteration, and to find the geometric NR variants, that can match the average iteration counts of Kallay's geometric NR.

The actual performance time depends on evaluation costs of the curve's points and derivatives, making it dependent on the problem and type of curves as well. In the case of integral polynomial curves, the classic Newton method performed by far the fastest - the proxy set-up and query evaluation costs surprassed that of the derivatives'. The classic NR was followed by the tangent line and 3-point circle proxy variants, then Kallay's. The osculating parabola proxy variant was the slowest.

Method	Eval. $\mathbf{p}(t)$	Eval. $\mathbf{p}'(t)$	Eval. $\mathbf{p}''(t)$
Classic NR	1	1	1
Kallay gNR	1	1	1
Tangent line gNR	1	1	0
Fwd diff line gNR	2	0	0
3-point circle gNR	3	0	0
Osculating parabola gNR	1	1	1

Table 1: The table shows at how many distinct parameter values the curve and its derivatives have to be evaluated during a single step of the given NR variants.

The derivative-free variants become more appealing when the evaluation of the derivatives become more expensive, e.g. in the case of rational curves. Table 1 shows the curve evaluation costs of the various methods.

The rate of convergence is an indicator of how sensitive the given variant is to the choice of the initial guess. A variant performing poorly in this regard might be better handled by a different initial guess strategy, however, that investigation is beyond the scope of our paper.

The win rate is the least decisive attribute in our the tests, it simply tells that among the algorithms that finished within the prescribed relative error, which one ceased its iteration at a point of the curve that is closest to the point in the query.

Method	Avg. iter. cnt.	Std. dev.	Converges	Wins
Classic NR	8.616472	3.61186	72%	15%
Kallay gNR	8.5057535	5.71219	73%	15%
Tangent line gNR	9.570799	4.6349697	56%	6%
Fwd diff line gNR	8.679719	5.4383	75%	23%
3-point circle gNR	3.8460969	2.1435	95%	37%
Osculating parabola gNR	8.092985	5.59519	68%	4%

Table 2: Test results using termination condition 0, relative error 10^{-6}.

Method	Avg. iter. cnt.	Std. dev.	Converges	Wins
Classic NR	14.52	6.06115	63%	41%
Kallay gNR	4.63	3.44918	88%	14%
Tangent line gNR	12.1	6.1206	53%	22%
Fwd diff line gNR	13.22	6.0626	44%	17%
3-point circle gNR	4	2.74608	79%	3%
Osculating parabola gNR	10.89	5.1206	59%	3%

Table 3: Test results using termination condition 1, relative error 10^{-6}.

Tables 3 and 4 show that the parametrisation-independent geometric termination conditions 1 and 2 put Kallay's and the 3-point circle fitting geometric NR

Method	Avg. iter. cnt.	Std. dev.	Converges	Wins
Classic NR	10.799472	6.65479	49%	37%
Kallay gNR	2.290768	1.47857	88%	11%
Tangent line gNR	12.099338	7.63352	35%	27%
Fwd diff line gNR	8.105676	7.21142	76%	1%
3-point circle gNR	2.8278252	2.25557	94%	5%
Osculating parabola gNR	6.3981366	7.74148	32%	19%

Table 4: Test results using termination condition 2, relative error 10^{-6}.

methods forward considerably in terms of average iteration counts. They also show that the 3-point circle fitting NR variant has the most similar characteristics to Kallay's geometric NR method in regards of average iteration count behaviour.

To evaluate its relative performance more in detail, table 5 shows the result of a more comprehensive test for the classic, Kallay's geometric, and the proposed 3-point circle fitting geometric NR methods, with termination condition 1. Termination condition 1 was chosen because it is the more general parametrisation independent termination condition.

A set of 30000 curves, chosen randomly from families of regular and non-regular curves, were used, and for each curve, 10 random points were generated to compute the closest point of the given curve to the random point. The curves used in the test were integral and polynomial Bézier curves, trigonometric curves, piecewise curves with first and higher order derivative discontinuities.

Method	Avg. iter. cnt.	Std. dev.	Converges	Total time
Classic NR	14.32251	6.06115	59%	173s
Kallay gNR	8.85953	5.64918	67%	470s
3-point circle gNR	9.20167	6.74608	75%	284s

Table 5: Test results using termination condition 1 on 30000 random curves, relative error 10^{-6}. Column Total time shows the run time of a given NR variant on the random curve and point set, in seconds.

The average iteration counts of the methods are closer to each other than in the case of integral polynomial curves. Kallay's geometric NR method still requires the fewest steps on average, but due to the computational cost of a single step, its run time is the longest. The classic Newton method finished the fastest, followed by the proposed 3-point circle NR variant.

Table 5 shows that the proposed 3-point modification of Kallay's method succeeded in speeding up the algorithm while retaining its robustness in the point-curve closest point computation problem.

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János