LATVIJAS UNIVERSITĀTES
RAKSTI

770. SĒJUMS

Datorzinātne un informācijas tehnoloģijas

SCIENTIFIC PAPERS
UNIVERSITY OF LATVIA

VOLUME 770

Computer Science and Information Technologies

SCIENTIFIC PAPERS
UNIVERSITY OF LATVIA
VOLUME 770

Computer Science and Information Technologies

University of Latvia

LATVIJAS UNIVERSITĀTES
RAKSTI
770. SĒJUMS

Datorzinātne un informācijas tehnoloģijas

Latvijas Universitāte

UDK 004(082)
Da 814

ISSN 1407-2157
ISBN 978-9984-45-377-4

Contents

Zane Galviņa, Darja Šmite
Software Development Processes in Globally Distributed Environment 7

Līva Šteinberga, Darja Šmite
Towards a Contemporary Understanding of Motivation
in Distributed Software Projects: Solution Proposal 15

Renārs Liepiņš
lQuery: A Model Query and Transformation Library 27

Andris Paikens, Zane Bicevska, Janis Bicevskis
Testing Computer Skills In Productive Environment:
Solution and Know-How 46

Leo Truksans, Edgars Znots, Guntis Barzdins
File Transfer Protocol Performance Study for
EUMETSAT Meteorological Data Distribution 56

Alina Vasilieva, Taisia Mischenko-Slatenkova
Computing Relations in the Quantum Query Model 68

Dmitrijs Rutko
Fuzzified Algorithm for Game Tree Search with Statistical
and Analytical Evaluation 90

Solvita Zariņa
Computer Scientists as Early Digital Artists 112

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2011. Vol. 770
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 7–14 P.

Software Development Processes in Globally Distributed Environment

Zane Galviņa[1], Darja Šmite[1, 2]
[1] University of Latvia, Latvia, [2] Blekinge Institute of Technology, Sweden
zane.galvina@lu.lv, darja.smite@{lu.lv | bth.se}

As a result of globalization, software is nowadays more often produced by development effort from multiple locations. While global software development is regarded as more challenging than even the most complex project managed entirely in-house, standards or methodologies dedicated for this type of projects are still lacking. Based on an extensive literature review towards an understanding of industrial practice regarding software development processes, authors of this paper conclude that the evidence on how such projects are organized is scarce. Despite the limited evidence, authors present the deduced models of development processes presented in selected literature and summarize the main challenges that may affect project management processes.

Keywords: Software Engineering, Distributed development, Software Development Process, Global Software development.

1 Introduction

Due to continuous increase in competition in the field of Information Technologies, companies are forced to provide products and services, which coincide with efficient and effective development, and high quality standards that are economically viable. Agile development is a movement that has entered software engineering to bring faster and cheaper development through lightweight process thinking and entrusting work to skilled people rather than enforcing heavy documented standardized process models. Another way to achieve competitiveness is to start distributing development globally in order to gain a benefit from getting more and cheaper resources. In this paper, the focus is brought on the latter. Unfortunately, many of the assumed benefits of global software development are associated with significant challenges [1] that hinder smooth project performance. Thus, a better understanding of processes undergoing in software projects in distributed environment may help to find the necessary improvements and reach the promised benefits.

A closer look at the main difficulties reveals that geographical distance inherent in distributed environment, in which software project team members are separated in space and time, has significant impacts on communication, coordination and other processes

[2, 3]. Many reams have been written to describe these challenges while solutions in this area are still not well-represented [1]. While traditional software engineering is managed with the help of various well-known life cycle models, development and management methodologies and standards, the area of distributed work is relatively unexplored and there is still no standard approach to run distributed projects.

The underlying assumption of software process research that stresses the importance of this area is the direct correlation between the quality of the process and the quality of the developed software [4]. Since the invention of the waterfall lifecycle [5], many process models have been introduced varying from disciplined well-defined processes to undisciplined and ill-defined processes, and from heavy and slow to light and agile processes [6]. Software lifecycle, in fact, is a skeleton and a philosophy, which defines, how software processes will be carried out, and specifies such characteristics as tools, infrastructure, environment, methods and techniques, organizations and people etc. [4]. Along with the distribution of software projects, the role of a clear understanding of software lifecycle and processes emerges. Because distributed software projects are regarded as more complex than even the most complicated projects performed entirely in-house [7], we conjure that it is also important to explore whether the very notion of the lifecycle does not change with distribution. In other words, the dualistic (or multiplistic) nature of distributed software projects could also affect the way one should view, specify, and perhaps execute distributed software project lifecycles.

Motivated by this gap, authors of this paper aim to explore different lifecycles and work division approaches and their relation to existing process models. Thus, the following research questions are put forward:

RQ1: How are distributed projects organized in terms of process patterns?
RQ2: Are there successful or unsuccessful process patterns?

The rest of the paper is structured as follows. Section 2 presents the overview of the research process. Section 3 presents the results of the literature review – presented development process models and challenges affecting project management as well as process specifics. Finally, Section 4 concludes the paper.

2 Research Overview

In order to answer the research questions mentioned above, an extensive selected literature review was conducted. Research papers were collected from the following venues being regarded as the key publication sources in the area of global software engineering:

- Proceedings of the Inter. Conf. on Global Software Engineering 2006-2009;
- Communications of ACM special issue on GSE – 49(10)/2006;
- SPIP special issues on GSE – 8(4)/2003, 13(3)/2008, 13(6)/2008, 14(5)/2009;
- IST special issue on GSE – 49(9)/2006;
- IEEE Software special issues on GSE – 18(2)/2001, 23(5)/2006.

In total, 261 papers were collected and reviewed. Since global development is quite a broad concept, several criteria were taken into account in order to screen the papers and select the relevant ones. The screening was performed by the first author.

- **Criterion 1.** The paper reports information about distributed software development project. Thus, if the title clearly states that the article is about a comparison of different tools used in distributed development, the paper was rejected. In case of very general titles abstracts were read. Most of the authors present detailed information about the paper in the abstract, so to understand whether this criterion was satisfied, a review of introduction and/or conclusion was necessary only in few situations.
- **Criterion 2.** The paper presents more specific information about the work division methods or process location (detailed description of a process) in a project. Since this criterion is more specific, we could identify it only after reviewing introductions and conclusions.

After reviewing the 261 initially collected papers, only 8 papers were regarded as relevant for further analysis. Since the amount of selected papers was very low, which threatened the validity of conclusions, we continued searching through the reference list of the selected 8 papers. The gathered references resulted in the identification of 97 additional hints. Further collection was based on 4 criteria:

- **Criterion 3.** Availability of reference (6 papers were not found).
- **Criterion 4.** Reference written in English (5 papers failed).
- **Criterion 5.** Conference proceedings or journal paper (40 references failed).
- **Criterion 6.** Unique paper (8 references were duplicates).

Finally, 34 papers were included for a more thorough review of their relevance for the topic under investigation, but they failed after the evaluation using criterion 1 and criterion 2. After applying the next four criteria, only 4 papers were considered for further analysis. A total of 12 papers were carefully read, and the information about the process distribution, locations, work division and associated challenges was mapped for further analysis. The information was captured in a protocol and then broken down to the following topics of discussion (see Fig. 2.1).

Fig. 2.1 Emerging observations

3 Research findings

Our observations show that the most commonly discussed issues are related to general project information or description of methodology and processes, which are followed. Authors often describe the issues faced in the studied projects and present guidelines how to overcome these challenges. In the majority of the cases, these challenges are related to the whole project, and not to a specific project lifecycle or work division approach. This was also one of the reasons for excluding many studies from the final analysis. Nonetheless a few useful observations were made and are discussed below.

3.1 Definition of processes

While every organization adopts a suitable lifecycle or process model, the notion of following one lifecycle or process model in distributed projects is challenged by collaboration among several organizations often having their own processes, methodologies, and tools. Lack of standards in the activities between distributed teams is regarded as a problematic setting [8]. A lack of coherent development or execution processes is also reported in several other studies [9, 10, 11]. According to Sudershana et al., lack of clear process for project execution leads to increased level of frustration and decreased feeling of ownership, which ultimately results in a very poor acceptance level at the remote location [9]. This finding indicates that a process definition including work division and allocation strategies that clarify roles and responsibilities are essential.

3.2 Distribution of processes

It is difficult to decide whether a particular project should be developed by globally dispersed teams – and where it can be better developed, as well as how to divide it across sites [8]. Among the accepted papers there are several describing a certain distribution of phases and also phase allocation strategies. At the same time, several projects include a detailed description of each phase.

The available information was sufficient only to study different variations of phase allocation strategies. In particular, we studied Requirements engineering, Design, Coding and Testing phases and their allocation to different organizations or sites. One of our goals was however to find out what kind of work division approaches were adopted in each project, including different activities and roles each organization played. This is particularly important when a phase is shared by two organizations, since this can be done in a variety of ways. We failed at this point because the presented details about the projects were scarce. Therefore, only conclusions from studying different phase allocation strategies are discussed.

For simplicity reasons, we demonstrate different approaches by depicting phases in a sequential manner distributed between only two sites. This has been done to categorize all the selected models. Three possible alternative divisions would have been available – 1) strict phase separation, 2) joint execution, and 3) hybrid approach (see Fig. 3.2.1)

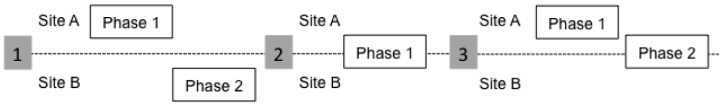


Fig. 3.2.1 Work division approaches.

In the answer to the **RQ1**, we conclude that hybrid division of work [9, 12, 13, 14] is more common than a strict allocation of phases or joint execution of the whole project [10] (see Fig.3.2.2). Possible reasons for the dominance of hybrid models are twofold. We assume that joint execution is an option for site A to mitigate the risks through control and partial participation in the work of site B, and vice versa – involvement of the site B in the activities executed by the site A is an option to mitigate the challenges of information sharing and knowledge transfer in later phases. At the same time, certain key or critical phases, such as requirements engineering, are executed independently due to, e.g., proximity to the customers.

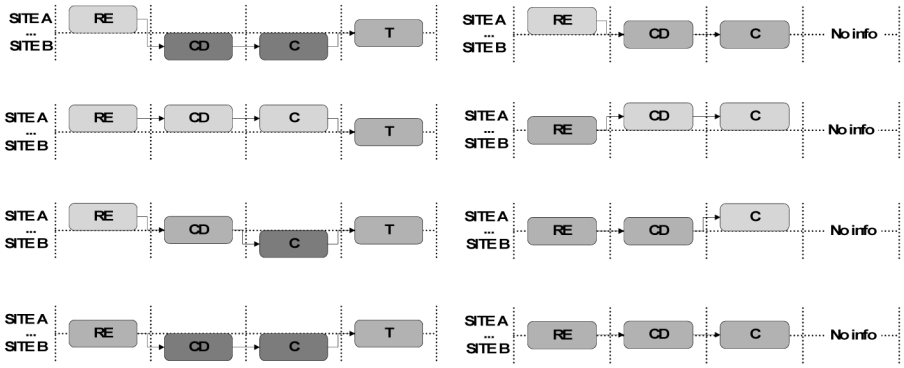


Fig. 3.2.2 Work division models [9,10*,12, 13, 14].
*presents many division models.

While reading project descriptions, several common uncertainties were identified. First of all, one may wonder, whether a phase is performed jointly, are all activities performed jointly too, or do organizations split the work? Secondly, if employees from one site are moved to another site for the entire phase, does that mean that the phase is allocated at one site, or performed jointly? From these questions we conclude that the phase visualization has its limitations and the process description shall allow reflecting answers to all these questions.

3.3 Processes and project success

Unfortunately, the success or failure of the work activities or work division approaches was not explicitly discussed. From the studied research, it was not possible to determine the factors, which have had an influence (positive or negative) on projects. This is yet another case were we have stumbled upon a lack of description of the project lifecycle models, work distribution or even phase allocation, which significantly affected the ability to drive any meaningful conclusions regarding successful or unsuccessful process patterns (**RQ2**).

Despite the different approaches to work division, each of the studied hybrid projects faced many challenges. It shows that there are no universal development model for all projects and organizations. Each organization has to choose the most appropriate model for their situation.

3.4 Processes and challenges

One of the objectives was to understand whether the selected work division model in a project makes an impact on a project management. Project management is the application of knowledge, skills, tools and techniques to project activities in order to meet project requirements [15]. The following key challenges found in the literature can be linked with one or several knowledge areas of project management.

Joint execution with no standards for the interface between the sites.

Usually, every organization has their own standards, defined processes, which they follow, but no standards determine how to join them or how to work together [8]. This has mainly an effect on the project integration management.

Joint execution with limited synchronous interaction.

This is a challenge that is manifested in projects involving sites separated in time. It is more complicated to share information between team members that lack synchronous communication or have limited overlap in working hours [8]. In such cases, tools for sharing information commonly used in software projects are of little help. Hampered communication and coordination delay are also consequences of asynchronous interaction [11]. This affects project coordination, scope and time management as well as communication management significantly.

Joint execution requires more time than co-located work.

Tasks in global software development projects often take much longer than in co-located environments [11]. In order to estimate activity duration or develop a common schedule for project, it is not enough to rely on experience from co-located projects. This affects the project time management as well as the scope management.

3.5 Specifics of processes

From our observations we deduce that the decisions of process model are closely affected by the work allocation to different sites and the chosen level of sharing the processes, phases or activities. In the following figure, we offer our understanding of process-specific factors of concern, based on a class diagram of a software process model defined in [16].

Accordingly, we emphasize that each phase can be allocated to an organization (or its site). However, for the clarity of roles and responsibilities it is also important to specify which activities are going to be performed by each location or joint.

A study depicting the proposed work division details would be sufficient to categorize the projects according to the following four main work division approaches [17]:

- *Phase-based approach* – This is a division of work by phase/process step, when globally dispersed sites engage in different phases of a project in a sequential manner [17].
- *Model-based approach* – Division of work by product structure (product module), when each product module/feature is developed in a single site [17].
- *Distributed approach* – Division of work that minimizes requirements for cross-site communication and synchronization; however, only for particular types of product architectures [17].
- *Customization-based approach* – This is a division of work based on product customization, so that one site develops the product and other sites perform

customization, that is, changes such as adding features and enhancements for specific customers [17].

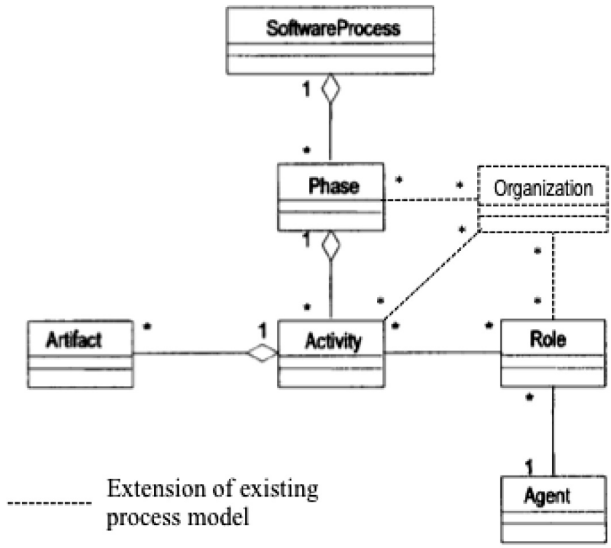


Fig. 3.5.1 Process model (extension of [16])

We consider it to be important that the above-discussed details must be specified by authors of empirical studies of distributed projects who report the success or failure of certain projects, processes, approaches, or practices, in order to further trace the main reasons to the process organization. This could further facilitate determination of successful or unsuccessful process patterns.

4 Conclusion

The efforts directed towards capturing different approaches used in practice and reported in academic literature showed that research on description of process models and work division practices in globally distributed projects is scarce. More often, the studies present the main challenges but do not mention the necessary information about the project context, which confirms the findings from a systematic review in this area [1]. Our findings also show that it is not possible to strictly define or make any conclusions about the best strategy for process allocation and work division. Despite that, we were able to identify several key challenges related to project management that arise from joint execution of task. This paper also presents an extension of an existing process model and emphasizes the necessity for accurate description of distributed projects.

By evaluating the ratio between the amount of selected papers for further analysis and the rest of the papers initially collected from the venues that represent research in the studied field, we can also conclude that there is a need for additional studies in this area, i.e. the understanding of software processes in distributed projects. To foster the

progress, we present specifics of processes necessary to be identified in future research. We also encourage reporting complete and comprehensive information about the studied projects.

References

1. Conchúir, E. Ó., Holmström, H., Ågerfalk, P.J., & Fitzgerald, B. "Exploring the assumed benefits of global software development". In P. Fernandes et al. (Ed.), IEEE International Conference on Global Software Engineering. Los Alamitos, CA: IEEE Computer Society, 2006, pp.159-168.
2. Damian D, Zowgui D. "The Impact of stakeholders' geographical distribution on managing requirements in a multi-site organization". Proceedings of International Conference on Requirements Engineering, Monterey,CA, 2002.
3. D. Šmite, C. Wohlin, R. Feldt, T. Gorschek "Empirical Evidence in Global Software Engineering: A Systematic Review", In: Journal of Empirical Software Engineering, Vol. 15, Nr. 1, February 2010, pp. 91-118.
4. Fuggetta A. "Software process: a roadmap". In Proceedings of the Conference on The Future of Software Engineering (ICSE '00). ACM, New York, NY, USA, 2000, pp. 25-34.
5. Royce W. W., "Managing the development of large software systems: Concepts and techniques," Proc. WESCO, 1970.
6. Rodriguez-Martinez L.C., Mora M., Alvarez F.J, "A Descriptive/Comparative Study of the Evolution of Process Models of Software Development Life Cycles (PM-SDLCs)," Mexican International Conference on Computer Science, 2009, pp. 298-303.
7. Karolak, D. **"**Global Software Development: Managing Virtual Teams and Environments". Wiley-IEEE Computer Society Pr; 1st edition, December 27, 1998.
8. Prikladnicki R.,Audy J., Evaristo R. "Global Software Development in Practice Lessons Learned". Software Process: Improvement and Practice.2003, pp. 267-281.
9. Sudershana S., Villca-Roque A., & Baldanza J. "Successful Collaborative Software Projects for Medical Devices in an FDA Regulated Environment: Myth or Reality". Proceedings of International Conference of Global Software Engineering, 2007, pp. 217-224.
10. Berenbach, B., "Impact of Organizational Structure on Distributed Requirements Engineering Processes: Lessons Learned", Workshop on Global Software Development for the Practitioners at ICSE, Shanghai, 2006, pp. 15-19.
11. Peter Faßbinder, Volker Henz, "Improving Global System Development and Collaboration across Functions: Experiences from Industry," Proceedings of International Conference of Global Software Engineering, 2009, pp.262-266.
12. Caprihan G. "Managing Software Performance in the Globally Distributed Software Development Paradigm". Proceedings of International Conference of Global Software Engineering, 2006, pp. 83-91.
13. Burger W. "Offshoring and Outsourcing to INDIA". Proceedings of International Conference of Global Software Engineering, 2007, pp. 173-176.
14. Cusick J., & Prasad A. "A practical Management and Engineering Approach to Offshore Collaboration". IEEE Software, 2009.
15. A Guide to the Project Management Body of Knowledge (PMBOK Guide). American National Standard, 2004.
16. Karolak, D. "Global Software Development: Managing Virtual Teams and Environments". Wiley-IEEE Computer Society Pr; 1st edition, December 27, 1998.
17. Oshri I., Korlarsky J., & Willcocks L.P. "The Handbook of Global Outsourcing and Offshoring". Palgrave Macmillan 2009.

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2011. Vol. 770
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 15–26 P.

Towards a Contemporary Understanding of Motivation in Distributed Software Projects: Solution Proposal

Līva Šteinberga[1], Darja Šmite[1,2]
[1]University of Latvia, [2]Blekinge Institute of Technology
liva.steinberga@lu.lv, darja.smite@{lu.lv | bth.se}

Team motivation in software engineering is reported to have the largest impact on productivity, software quality and project overall success. Yet, as it is a soft factor and thus difficult to quantify, it usually takes a backseat. In fact, research on motivation and its interplay with different environmental factors in software engineering projects is scarce. Built on the basis of a recent systematic review on software engineers' motivation, in this paper, we emphasize the lessons learned from related studies regarding the factors determining team motivation. Further implications of distributed environment on motivation and the positive impact of agile practices are discussed and illustrated by examples from related studies. The contrasted project types (distributed and agile) suggest that blending agility into distribution might solve problems inherent in this type of environment. Finally, future research directions are suggested for project improvement through motivation.

Keywords: Motivation, Distributed Teams, Agile Teams, Distributed Agile Teams.

1 Introduction

Nowadays, in a quest for cheaper and faster high-quality software development many organizations have turned towards globally distributed software development [1]. Global software development (GSD) claims to enable the benefits of accessing larger resource pool, and reducing development costs in organizations that overcome the challenges of geographical distance [2]. As a result, nowadays, software projects more often involve geographically and temporally distributed and culturally diverse team members. These unique characteristics create significant challenges for communication, coordination and social networks [3] and being unable to overcome the new challenges a high number of global projects fails [4]. The emphasis in global projects is thus believed to be required for human aspects [5], which among other potential reasons of failure are often neglected since they are not easy definable and quantifiable.

Team motivation in software engineering is reported to have the largest impact on productivity, software quality and project overall success [6]. Nonetheless, motivation in global software teams has been relatively unexplored. To fill this gap we investigate what

motivates and what de-motivates software engineers, and how these influencing factors are manifested in global software projects. Furthermore, we consult related studies of agile teams from the past few decades, in which considerable effort has been devoted to exploring the social characteristics of software projects, as compared to research on traditional projects. This approach is built on team-centred philosophy and has inherent factors influencing motivation of software engineers. Inspired by related studies, we draw attention to potential areas of improvement with implications for practice and future research.

2 Background — Motivation in Software Teams

Motivation has been described as a source of performance improvement, which leads to productivity gains through exploitation of effective teamwork, in which team members act selflessly and contribute to the greater good [7]. In such teams, the team is more than the sum of its parts [7].

First experiments that demonstrated the influence of the team motivation on productivity were conducted by Elton Mayo in 1924, but later studies have confirmed the same results in several industries [8]. Mayo's experiments for the first time evidenced that workplaces are social environments, where people are motivated by many factors other than economic interest. He concluded that recognition, security, and sense of belonging were more important to productivity and morale or motivation, and a friendly relationship with the supervisor was very important in securing the loyalty and cooperation of the team [8].

In 1981, Barry Boehm reported that motivation has the single largest influence on quality and productivity than any other factor in software development, while in 1999, DeMarco and Lister's survey showed that the lack of motivation is one of the most frequently cited causes of software development project failure [6]. So far there are no other aspects claimed to have bigger influence on project effectiveness than developers' motivation, therefore it is important to understand what motivates software engineers to perform well and also what de-motivates them.

In 1978, Couger and Zawacki began a discussion whether software engineers form a distinct occupational group with similar needs and motives, and the discussion is still on-going. They surveyed the job perceptions of more than 6000 people from different professional areas and concluded that software engineers found their work less meaningful and rated their jobs less favourably than other professionals, their need to interact with others was negligible; they had very high growth needs and were concerned about learning new technology [9]. Beecham et al. in their systematic review on motivation confirmed that a little more than a half of empirical studies (54%) conducted in this field until year 2006 regarded software engineers as a distinct occupational group, while 24% of the studies denied this categorization [6].

There are several classical motivation theories, which can be applied in order to explain what motivates software engineers (such as Abraham Maslow's Hierarchy of needs, Clayton Alderfer's ERG theory, Frederick Herzberg's Motivation-hygiene theory, David McClelland's achievement motivation theory and others). One of the contemporary aspects of understanding motivation prescribes a division of the influencing factors into intrinsic and extrinsic [8, 9]. Intrinsic factors address the work

itself and the goals and aspirations of the individual, such as achievement, possibility for growth, social relationships, security, etc. Extrinsic factors are concerned with the surrounding environment brought by the organization to the individual, such as praise, communication, office space, responsibility, money etc.

Beecham et al. in their report on the motivation of software engineers have gathered motivators and de-motivators for software engineers from 92 empirical studies [6]. They have identified some of those factors as inherent for software engineering. In total, 29 motivators and 15 de-motivators have been collected; those manifested in distributed and agile projects are listed in tables 1 and 2.

It is worth noting that authors of the systematic review have also captured the external signs associated with motivated and de-motivated software engineers, such as retention, productivity, project delivery time, budgets, absenteeism and project success [6]. While global projects are often suffering from similar negative impacts [4], our investigation in this paper is driven by the question whether a lack of motivation and presence of de-motivators can be the cause of failure.

3 Research Overview

The aim of our investigation is to understand the implications of motivational research regarding distributed software teams. More specifically, we consult research on agile software development projects for learning from team-centred approaches to increase the productivity and success ratio of distributed software projects. The research is thus driven by the following research questions:

RQ1: How the known motivators and de-motivators of software engineers are manifested in distributed software development projects?

RQ2: What can we learn from agile projects to eliminate de-motivators and enable motivators in distributed software development projects?

To address these research questions, we base our investigation on the results of the systematic review performed by Beecham et al. [6]. Manifestation of the identified motivators and de-motivators is explored through relating research findings from the field of global software engineering. It is worth noting that no extensive literature review is performed. Our goal in this paper is to exemplify manifestation of motivating and de-motivating factors. Further, we highlight the main areas of concern and conjure these as problems inherent in the nature of the distributed environment. We explore the factors related to motivation enabled by inherent characteristics of agile methods, and propose the potential areas of improvement for distributed projects inspired by agile approaches.

4 Findings

4.1 Motivation in Distributed Projects

To the best of our knowledge, no research dedicated to motivation can be found in the area of commercial distributed or global software development (with an exception of Open Source development projects, which are also distributed and global, but not

commercial). By distributed projects, we mean such software development projects, in which tasks (be it a phase or a development task) are split among several geographically distant locations. These locations can be represented by sites of the same company (offshore insourcing) or different companies (offshore outsourcing). The findings from related research in the area of global software engineering point out evidences of various de-motivating factors caused by distribution. Our observations suggest that some of these factors are inherent in the very nature of distributed projects, and thus de-motivation among software engineers in such projects may be manifested more often than in similar projects with co-located members. Hereby, we discuss motivators and de-motivators that have a special meaning in distributed projects in a concise way with examples of specific manifestation in distributed projects supported by the references to related research.

4.1.1 Manifestation of motivators in distributed projects

We have found that motivators such as Change, Benefit and Problem solving, Science, Experiment, Identification with the task, Career path, Variety of work, Recognition for work done, Development needs addressed, Making contribution or task significance, Rewards and incentives, Feedback, Job security, Good work life balance, Appropriate working conditions, Working in a successful company, and Sufficient resources are more generic and are thus not affected by distribution.

Challenge / Intrinsic motivator: Enabled. Software engineering is regarded as a challenging profession [6]. In its turn, globally distributed development is recognized to be considerably more challenging than even the most complex project managed entirely in-house [4, 10]. Thus, we can claim that challenges are triggered by distribution.

Team work / Intrinsic motivator: Challenged. Developers are recognized to be motivated by working in a team of other professionals rather than alone [6]. Distributed teams on the contrary tend to be divided into sub-teams by location and often experience cross-site competition instead of collaboration [10], which is regarded as "us and them" attitude [11]. Also, distribution makes it difficult to apply mutual adjustments and supportive behaviour, which are commonly used in effective teamwork and especially important for dealing with complex tasks [12]. Thus, we can conclude that although teamwork can be established inside the co-located sub-teams, motivation from the cross-site teamwork perspective is likely to be challenged.

Development practices / Intrinsic motivator: Challenged. While developers can feel united around the use of a certain methodology or development practice, for example, object-oriented, agile, or prototyping practices [6], cultural and also organizational differences associated with distributed projects often lead to discrepancies in the work habits [13] and sometimes to enforced use of the other site's methodology, which might de-motivate the developers.

Technically challenging work / Intrinsic, general motivator: Challenged. Work is found to be motivating if it is not mundane and is technically challenging [6]. Unfortunately, companies often practice outsourcing of routine tasks and keeping the more interesting work for themselves. This could be one of the reasons why offshoring, in particular to India, is often associated with rapid turnover of employees.

Autonomy / Intrinsic, general motivator: Challenged. Freedom to carry out tasks letting roles evolve is regarded as a motivator for software engineers [6]. In distributed

projects, remote sites are often closely supervised by their headquarters [14] and even regarding technical decisions autonomy may be limited [15]. This is one of important factors to be considered in distributed projects, in which site inequity (discussed later) can further complicate the morale of the remote sites.

Empowerment or responsibility / Intrinsic, general motivator: Challenged. Responsibility for the task is recognized as a motivator for software engineers to perform better [6]. However, remote sites, e.g., in offshore development relationships, are more often involved only in partial activities (only coding, only testing or only maintenance activities [16]) and do not receive the responsibility for the whole development project. While the onshore part of the team is often working on the functionality holding the ownership and thus, in addition, creating an iniquity between the parts of the team.

Trust, respect or equity / Intrinsic, general motivator: Challenged. Trusting and respecting other people, treating and managing them fairly has been identified as one of the key motivators for software developers [6]. Meanwhile, there is a number of studies reporting lack of trust as one of the major problems in globally distributed software development teams (such as [14], [17], [18]). Key factors that cause the lack of trust are poor socialization and socio-cultural fit, increased monitoring, inconsistency and disparities in work practices, reduction of communication, lack of face-to-face meetings, poor language skills, lack of conflict handling, lack of cognitive-based trust [14], and others. Those factors also relate to other motivators identified by Beecham et al. [6]. For instance, autonomy is another key motivator, hindered by increased monitoring.

Employee participation / Intrinsic, general motivator: Challenged. This motivator can be described by involvement in the company and working with others [6]. Herbsleb and Mockus have found that distributed work items appear to take about two and a half times as long to complete as similar items where all the work is co-located [3]. Aiming to find the reasons behind such a gap in productivity, they had observed that communication, coordination and social networks in distributed teams may differ from single-site counterparts in a way that it requires more people to participate thus introducing a delay [3].

Sense of belonging / Extrinsic motivator: Challenged. Sense of belonging to the team and supportive relationships between team members is regarded as an extrinsic motivator. In distributed teams, however, this remains one of the greatest challenges. According to Herbsleb and Mockus, people at different sites are less likely to perceive themselves as a part of the team [3]. This is illustrated by conflicting work styles adopted in the remote locations and by the perception that distant colleagues are less likely to help out when workloads are especially heavy. They thus conclude that cross-site relationships compared to same-site relationships are less oriented towards mutual benefit [3].

4.1.2 Manifestation of de-motivators in distributed projects

We have found that de-motivators such as Risk, Stress, Unfair reward system, Uncompetitive or poor pay and unpaid overtime, Unrealistic goals and phoney deadlines, and Poor management are more generic and thus are not directly attributed to distribution.

Inequity: Triggered. Recognition based on management intuition or personal preference is regarded as de-motivating in software engineering [6]. Inequity in distributed

projects is expressed in various ways. It can be related to the already mentioned "us and them" attitude [11]. It is also manifested in various ways through unequal rules and requirements on remote sites. For example, members of highly distributed teams that work across multiple time-zones are often required to shift their working hours [19] creating dissatisfaction in long term. In addition, the time shifting is often tolerated by the offshore sites (late shifts [19]), while the work hours for the headquarter site or onsite teams often remain unchanged.

Interesting work going to other parties: Triggered. Beecham et al. [6] also indicate that software engineers may be de-motivated by being separated from the other team members and by outsourcing the most interesting work to other sites [6]. This is the case in companies that are downsizing their development budgets and sending the work to outsourcing suppliers. In such projects (we call them onsite), personnel will be de-motivated. In other projects, where boring work is sent offshore to free up onshore employees for new projects [20], the offshore site will become de-motivated.

Lack of promotion opportunities/ stagnation/ career plateau/ boring work/ poor job fit: Triggered. In line with the above, it appears that some companies practice offshoring through a relocation of more stable work, such as software maintenance and bug fixing, in order to free up onsite resources for the new development and thus more interesting work [20]. This is another example of inequity, boring work and stagnation, and also a manifestation of poor motivation in offshore sites, which could be an explanation for the sequential rapid turnover of employees in the sites that receive boring work.

Poor communication: Triggered. Feedback deficiency from colleagues and software users is de-motivating for software engineers [6], but one of the most demoralizing bad practices is the loss of direct contact to all levels of management for each staff member [21], which is very likely to occur in distributed setting due to geographic and often temporal distance.

Bad relationship with users and colleagues: Triggered. Software engineering is a complex field and the most problems can nowadays be effectively solved only by effective teamwork performed by engineers with a help from software users. Therefore, it is very important to form a cohesive team and maintain friendly and supporting relationships with users. Thus, bad relationships with users and colleagues can hinder one from motivation to work effectively [6]. In a distributed setting, it can be complicated to establish good and close relationships with remote colleagues and users because of the mentioned geographic and temporal distances, because it is more and more common that developers from different sites never meet [14].

Poor working environment: Triggered. Wrong staffing levels and unstable, insecure working environment lacking investment and resources is likely to de-motivate employees [6]. Being physically separated from the team can negatively affect software engineer's performance [6]. This factor is inherent for staff from remote sites in distributed software development.

Poor cultural fit/ stereotyping/ role ambiguity: Triggered. Software engineers share national, occupational and organizational cultures. If one does not feel like belonging to the culture, it could be disturbing and serve as a de-motivating factor. There is evidence about reciprocal stereotyping of software engineers by managers as a de-motivator [22] and role ambiguity, which involves uncertainty about role responsibilities,

expectations, or tasks [23]. These factors are likely to occur in a distributed setting due to socio-cultural distances that are associated with these projects [19].

Producing poor quality software: Triggered. It is observed that poor outcome of the work de-motivates software engineers [6]. Meanwhile, a transfer of software work from one site to another is recognized to have a negative impact on both productivity of software engineers and resulting quality of the outcome. It happens due to significant challenges of learning to handle the already developed software being unknown [20].

Lack of influence or no involvement in decision-making: Triggered. Distribution among the offshore sites and the headquarters, and especially temporal dislocation, leads to the feeling of lagging behind [19]. This is likely to have an impact on teamwork and all associated activities, such is decision-making. Thus, offshore sites are naturally left with a limited influence on the project.

4.2 Motivation in Agile Projects

Agile movement has emphasized the importance of human aspects in software development by manifesting one of its values "Individuals and interactions over processes and tools" and its principle "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done". Melnik and Maurer have described agile methods as human-centric bodies of practices and guidelines for building software in unpredictable and highly-volatile environments [24].

So far, the most popular agile practices are Extreme Programming (XP) practices, which among other benefits originally hold values of daily face-to-face communication between all project team members and have a regular feedback on the work done. There are several evidences that XP environment gives a grater job satisfaction and increases code quality [7]. Since those two signs are closely connected with external signs of the motivation of software engineers mentioned beforehand in section 2, it can be derived that XP environment is highly motivational. Besides, there are studies, which confirm that the more practices are incorporated in the development process, the higher is the positive emotions experienced by the developers [25]. Further, we discuss how different practices address motivation of software engineers.

Iterations and small releases. Iterations and small releases enable software engineers to receive early and frequent feedback and recognition for well-done job from all the stakeholders. It increases the self-esteem of the software engineers and the level of trust between them and the customer [8].

Enables: Feedback, Recognition, Trust/ respect.

Helps to avoid: Unrealistic goals/ phoney deadlines.

Simple design, continuous testing and continuous integration. Continuous software development process provides early feedback which prevents from delays and integration problems later thus avoiding employee dissatisfaction.

Enables: Feedback [25].

Helps to avoid: Unrealistic goals/ phoney deadlines.

Regular face-to-face meetings. Daily stand-up or weekly face-to-face meetings address the need for developers to feel they are making progress, which is common in groups of professionals with a high need for personal growth and development [26]. Besides, regular meetings ease and speed communication and collaboration [27]. This

practise and small chunks of action assigned to individuals support sense of belonging to the team and maintain team awareness of member activity [26].

Enables: Sense of belonging, Good management, Identify with the task, Empowerment / responsibility, Employee participation, Team work.

Helps to avoid: Poor communication, Poor management, Producing poor quality software [27], Unrealistic goals/ phoney deadlines [7].

Pairing. Pairing motivates software engineers because it addresses the need for learning, autonomy and social activity [7], and it also provides feedback [25]. However, pairing can be de-motivating if pairs have personality conflicts [7] or continuous pairing is too intense [26]. Individuals can become bored with regular rhythm of development and become dependent on working in pairs thus losing their confidence to work alone [26].

Enables: Autonomy, Feedback, Development needs addressed, Bad relationships with colleagues.

Helps to avoid: Poor communication, Producing poor quality code.

Self-organizing team. Software engineers in XP environment are working in self-organising teams which means that individuals are given an autonomy or freedom to carry out tasks themselves, allowing roles to evolve. As reported by Beecham et al., this trait of XP teams is one of the main motivators for software engineers in general [6]. It also discourages from working in an insular myopic fashion that de-motivates other members of the team [26]. In self-organizing teams, a principled leadership is applied which allows software engineers to make the decisions that affect them and participate in personal goal setting which is found to increase the job satisfaction [6]. Agile developers have been found to be motivated by working with people who possess very good communication skills [26] and who are highly competent [8]. On the other hand, the lack of these traits has been found de-motivating. De-motivating factor in XP environment is that individual inputs into the project might be subsumed by the whole team, thus impacting promotion opportunities [26].

Enables: Autonomy, Empowerment/ responsibility, Career path, Lack of promotion opportunities.

Helps to avoid: Lack of influence/ not involved in decision making/ no voice.

5 Discussion — Blending Agility with Distribution

Evidence from empirical studies in the area of global software engineering suggests that this type of environment is full of challenges that trigger de-motivators and hinder motivators for software engineers. In the following tables, we present a summary of the findings from exploring distributed projects and agile projects.

Table 1

Motivators (based on [6]) Manifested in Distributed and Agile Projects

Motivators	**Challenged by distribution**	**Triggered by distribution**	**Solved in Agile**
INTRINSIC MOTIVATIORS			
Challenge		✓	
Team work	✓		✓
Development practices	✓		✓
General motivators			
Identify with the task	✓		✓
Variety of work	✓		✓
Recognition for work done	✓		✓
Technically challenging work	✓		
Autonomy	✓		✓
Empowerment / responsibility	✓		✓
Trust / respect / equity	✓		✓
Employee participation	✓		✓
EXTRINSIC MOTIVATORS			
Sense of belonging	✓		✓
Feedback	✓		✓

Table 2

De-motivators (based on [6]) Manifested in Distributed and Agile Projects

De-motivators	**Triggered by distribution**	**Solved in Agile**
Inequity	✓	
Interesting work going to other parties	✓	
Lack of promotion opportunities/ stagnation/ career plateau/ boring work/ poor job fit	✓	
Poor communication	✓	✓
Bad relationship with users and colleagues	✓	
Poor working environment	✓	
Poor cultural fit/ stereotyping/ role ambiguity	✓	
Lack of influence/ not involved in decision making/ no voice	✓	✓

Although our study does not differentiate and systematically evaluate diverse implementations of global projects, it raises an important question about the motivators hindered and de-motivators triggered by geographic and temporal distribution. We therefore conjure that a contemporary understanding of motivation in software projects is required as more and more companies become distributed and utilize resources from all around the world. Another contemporary aspect that is not well understood to date is related to cultural diversity. In particular, one may wonder whether motivators and de-motivators are equally strong across different countries. Thus, further empirical studies into the phenomenon of motivation in distributed projects are important.

Having said that distributed software projects trigger a lot of de-motivating factors, in this paper, we also aim at understanding how to solve these environmental flaws. One source of inspiration for us has been agile projects. Even though agility and distribution may seem as an incompatible mix [28], the team-centred approaches suggested by agile methods are already evidenced in distributed environment [28, 29]. While previous studies have solely focused on positive impacts of agile methods on team communication and coordination experiences, we emphasize the necessity to address motivation as the prime object of study.

In this study, we have explored the current understanding of motivation in traditional software teams based on related studies to date. However, one of our goals is to emphasize that it is fair to assume that environment including software projects changes over time and existing studies could provide a limited explanatory power to understand the motivation in, e.g., projects involving developers from all around the world. In order to be successful, organizations should strive to understand how to motivate their employees in actual project settings and keep an eye on the rapidly changing factors influencing the motivation.

Our future research includes empirical evidences of the motivation of software engineers in different types of distributed projects where software engineers with different cultural backgrounds are involved. We are also planning to investigate what we can learn from open-source projects in order to raise the team motivation in GSD.

6 Conclusions

Our observations suggest that many motivating factors are challenged and de-motivating factors are inherent in the very nature of distributed projects, and thus we argue that de-motivation among software engineers in such projects may be manifested more often than in similar projects with co-located members. We trace the negative effect on motivation to geographic and temporal distance and cultural diversity, and claim that without explicit concern of motivation, global projects will ultimately fall into the category of dissatisfactory for those involved.

Ramasubbu and Balan states that productivity and quality can be modelled as a function of personnel-related factors and software methodology-related factors [29]. In our investigation, we confirm that software methodology and also project environment matters. We study two distinct approaches – distributed projects and agile projects –, which are analysed in the aspect of motivational studies. We suggest that the equation

shall also include factors that influence motivators and de-motivators for software engineers.

Further, we discuss the potential of agile approaches to solve the problems of de-motivation in distributed projects and propose future research to focus on evaluating the impact of agility on motivation. Future work also involves empirical investigation of motivation in different types of culturally diverse distributed projects and investigation of what we can learn from open-source projects in order to raise team motivation in GSD.

Acknowledgements

This work has been supported by European Social Fund project
No. 2009/0216/1DP/1.1.1.2.0/09/APIA/VIAA/044.

References

1. D. Šmite, C. Wohlin, T. Gorschek, and F. Robert, "Empirical evidence in global software engineering: a systematic review," Empirical Software Engineering, vol. 15, no. 1, p. 91, 2010.
2. L. Layman, L. Williams, D. Damian, and H. Bures, "Essential communication practices for Extreme Programming in a global software development team," Information and Software Technology, vol. 48, no. 9, p. 781, 2006.
3. J. D. Herbsleb, "An empirical study of speed and communication in globally distributed software development," IEEE Transactions on Software Engineering, vol. 29, no. 6, p. 481, 2003.
4. C. Ebert, "Optimizing supplier management in global software engineering," in Proceedings of the International Conference on Global Software Engineering, 2007.
5. T. Hall, H. Sharp, S. Beecham, N. Baddoo, and H. Robinson, "What do we know about developer motivation?," Software, IEEE, vol. 25, no. 4, p. 92, 2008.
6. S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in software engineering: a systematic literature review," Information and Software Technology, vol. 50, no. 9-10, p. 860, 2008.
7. A. Law and C. Raylene, "Effects of agile practices on social factors," in Proceedings of the 2005 workshop on Human and Social Factors of Software Engineering - HSSE 05, 2005, p. 1.
8. G. Asproni, "Motivation, teamwork, and agile development," Agile Times, IV (1), p. 8–15, 2004.
9. B. L. Mak and H. Sockel, "A confirmatory factor analysis of IS employee motivation and retention," Information & Management, vol. 38, no. 5, pp. 265-276, 2001.
10. A. Piri, T. Niinimäki, and C. Lassenius, "Descriptive analysis of fear and distrust in early phases of GSD projects," presented at IEEE International Conference on Global Software Engineering, 2009.
11. D. Šmite and C. Gencel, "Why a CMMI level 5 company fails to meet the deadlines?," Product-Focused Software Process Improvement, p. 87–95, 2009.
12. D. Šmite, N. Moe, and R. Torkar, "Pitfalls in remote team coordination: lessons learned from a case study," Product-Focused Software Process Improvement, p. 345–359, 2008.
13. J. S. Olson and G. M. Olson, "Culture surprises in remote software development teams," Queue, vol. 1, no. 9, p. 52–59, 2003.
14. N. B. Moe and D. Šmite, "Understanding a lack of trust in global software teams: a multiple-case study," Software Process Improvement and Practice, vol. 13, no. 3, p. 217, 2008.
15. R. Prikladnicki, J. L. N. Audy, and F. Shull, "Patterns in effective distributed software development," Software, IEEE, vol. 27, no. 2, p. 12–15, 2010.
16. S. Islam, M. M. A. Joarder, and S. H. Houmb, "Goal and risk factors in offshore outsourced software development from vendor's viewpoint," in Proceedings of IEEE International Conference on Global Software Engineering, 2009, p. 347–352.
17. A. Piri, T. Niinimäki, and C. Lassenius, "Fear and distrust in global software engineering projects," Journal of Software Maintenance and Evolution: Research and Practice, 2010.
18. S. Jalali, C. Gencel, and D. Šmite, "Trust dynamics in global software engineering," in Proceedings of

the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM, Bolzano-Bozen, Italy, 2010.
19. H. Holmstrom, E. O. Conchuir, P. J. Ågerfalk, and B. Fitzgerald, "Global software development challenges: a case study on temporal, geographical and socio-cultural distance," IEEE International Conference on Global Software Engineering, 2006.
20. D. Šmite and C. Wohlin, "Software product transfers: lessons learned from a case study," IEEE International Conference on Global Software Engineering, 2010.
21. S. A. Frangos, "Motivated humans for reliable software products," Microprocessors and Microsystems, vol. 21, no. 10, pp. 605-610, Apr. 1998.
22. S. Ramachandran and S. V. Rao, "An effort towards identifying occupational culture among information systems professionals," in Proceedings of the 2006 ACM SIGMIS CPR, Claremont, California, USA, 2006, p. 198.
23. M. F. Reid, M. W. Allen, C. K. Riemenschneider, and D. J. Armstrong, "Affective commitment in the public sector," in Proceedings of the 2006 ACM SIGMIS CPR, Claremont, California, USA, 2006, p. 321.
24. G. Melnik and F. Maurer, "Comparative analysis of job satisfaction in agile and non-agile software development teams," Extreme Programming and Agile Processes in Software Engineering, p. 32–42, 2006.
25. S. L. Syed-Abdullah, J. Karn, M. Holcombe, T. Cowling, and G. Marian, "The positive affect of the XP methodology," in Extreme Programming and Agile Processes in Software Engineering, Sheffield, UK, 2005.
26. S. Beecham, H. Sharp, B. Nathan, T. Hall, and H. Robinson, "Does the XP environment meet the motivational needs of the software developer? An empirical study," in AGILE 2007, 2007, p. 37.
27. E. Whitworth and R. Biddle, "The social nature of agile teams," in AGILE 2007, 2007, p. 26–36.
28. D. Šmite, N. B. Moe, and P. J. Ågerfalk, "Agility across time and space: making agile distributed development a success," Springer, Heidelberg, Germany, 2010.
29. N. Ramasubbu and R. K. Balan, "Globally distributed software development project performance: an empirical analysis," in Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering - ESEC-FSE 07, 2007, p. 125.

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2011. Vol. 770
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 27–45 P.

lQuery: A Model Query and Transformation Library

Renārs Liepiņš
Institute of Mathematics and Computer Science
University of Latvia, Raina boulevard 29
Riga, LV-1459, Latvia
renars.liepins@lumii.lv

Transformation languages make it easy to work with models, but they are bound to one particular data store. That makes them hard to adopt in projects where data is stored in a different repository, which hinders more widespread use of transformations. Instead of adopting a transformation language to a new data store, we propose to build a query and transformation library for the general-purpose language that is already used in the project. We demonstrate how it can be done by implementing such a library for EMOF-like data store in a Lua scripting language.

Keywords: model transformations, model query.

1 Introduction

Model-driven engineering (MDE) has shown the usefulness of models and model transformations in software development. Its advent has fostered development of numerous languages specifically tailored for model transformations. Although these languages have largely solved the problem of working with models, there are still some problems that hinder wider use of transformations. The main difficulty is that each transformation language works only with a specific repository, and can be easily extended (if at all) only with a specific general purpose language. Consequently if we want to use a transformation language with another data store we need to either import/export our data to the data store that is supported by the transformation language or we need to write a wrapper for our data store so that the transformation language can work with it. This is problematic because the import/export approach can work only in situations where the transformations can work in a batch mode. Writing a wrapper is even worse because it requires detailed knowledge about the implementation of the desired transformation language. Another problem with existing transformation languages is extensibility, i.e. if we need some new primitive operation that the transformation language does not have, then how to add it? In principle, there are a few options: we can either extend the transformation language compiler or runtime ourselves, ask the transformation language developers to do it for us, or find a workaround. Neither of

these options is satisfactory: first two are too time-consuming, the last one would defeat the purpose of using a domain specific language.

To avoid these problems, we propose an alternative approach: instead of adopting an existing transformation language to the new data store, let us build a new query and transformation library in the general-purpose language we are already using in our project. We assume that the general-purpose language has lambda expressions. We think that it is justified because most mainstream languages either already have lambda expressions or will add them in the next major revision. Although, at first it may seem that it is unfeasible to build a library with the same expressive power as a domain specific language, it is actually quite doable using ideas from combinatorial parsing [1].

We will show how this can be done by developing a query library in the Lua scripting language [2] for working with an EMOF-like [3] data store. We chose Lua because it has first-class functions, C-like syntax, and very few core constructs, so it can be easily explained and understood. And we chose an EMOF-like data store because most transformation languages work with such data stores and that in turn makes it easier to compare the library features and expressiveness with existing transformation languages.

2 lQuery Library

The lQuery library is a set of functions for querying and modifying models stored in a model repository. It is implemented in the Lua scripting language and has been used for building meta-case tools as well as specific modeling tools in GRAF platform [4]. Before going into details about lQuery we will first give a brief overview of the Lua scripting language and the API of the model repository for which the library is implemented.

2.1 Brief Overview of Lua

Lua is dynamically typed scripting language, i.e. variables do not have types, but each value carries its own type. Comments, in Lua, start with double hyphens (‘`--`’) and run till the end of the line. In the following examples, we will use comments starting with ‘`-->`’ to indicate the result of preceding code.

Lua has only a couple of primitive value types: nil, strings, numbers, booleans, and functions. And there is only one data structure: an associative array, commonly called *table*. The indices and values in a table can be any Lua value: strings, numbers, booleans, functions, or other tables. Lua has a special syntax for creating tables: `{}` creates an empty table, and `{x=1, y="a"}` creates a table where index `"x"` has value `1` and index `"y"` has value `"a"`. There are two syntaxes for getting the value that is associated to a given key in a table: `t.y` and `t["y"]`, the former is just a syntactic sugar for the later:

```
t = {x=1, y="a"}
print(t.x)   --> 1
print(t["y"])       --> "a"
```

Lua has a standard set of control structures: **if** for conditions and **for** for iterations. All control structures have an explicit terminator: **end**.

```
if a < 2 then
  print(“a less than 2”)
else
  print(“a greater than 2”)
end

t = {“a”, 100, true}
for i, v in ipairs(t) do
  -- i is index, v is value, .. is concatenation operator
  print(“the value of index ” .. i .. “ is “ .. v)
end
```

Functions in Lua are first-class values meaning that functions can be constructed at runtime, assigned to variables, passed as arguments, and returned as results from other functions. All functions in Lua are anonymous. The statement `function (x) ... end` is a function constructor, just as `{}` is a table constructor. For example, to create a function that adds one to its argument, we write:

```
add_one = function(n)
            return n + 1
          end

add_one(3)       --> 4
```

Here, `add_one` is a variable to which we assign the constructed anonymous function.

Tables can also be used as objects. To make it more convenient, there is a special syntax for calling methods: `obj:foo(args)`. It gets the anonymous function stored at key `“foo”` in the table `obj` and calls it passing the table itself as the first argument. In this case, the table plays the role of *self* or *this* from other object oriented languages.

2.2 Overview of a Model Repository API

lQuery, like other model transformation languages, works on a model repository. The repository can be divided into two parts (Fig. 1): the schema part (upper three boxes) and the data part (lower three boxes). The data part is the actual part with which lQuery works, and the schema part is like annotations that help to understand what each data item means. The schema part consists of three things: *classes*, *attributes*, and *links*. Classes are used to group objects together, and the *super/sub* relation between classes is used to state that if an object belongs to a subclass then it also belongs to the superclass. Attributes are used as keys for associating string values to objects. Links are used for associating objects with other objects. The data part consists of: *objects*, *attribute values*, and *link assertions*. Objects are the actual values that are stored in repository. Each object has exactly one class. Attribute values are strings that are associated to some object with a particular attribute. Each object can have at most one attribute value for a particular attribute. Link assertions are a collection of objects that are associated to a

particular object for a particular link. An example of a repository content is given in the next chapter.

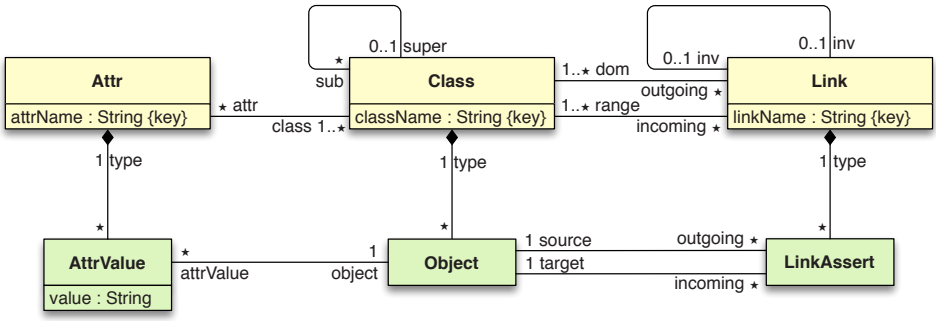


Fig. 1 Model Repository Metamodel

Each schema entity has a unique string id, and there is an API function to get an entity with a specific id. There are also functions to get all objects that belong to a specific class, check whether an object belongs to a specific class, create an object, delete an object, get the value of an object attribute, set the value of an object attribute, get linked objects, add link between two objects, and delete link between two objects. The repository API functions are listed in the Appendix 1.

Theoretically, these functions are sufficient to write any transformation, but the resulting code would be very repetitive, i.e. some patterns would repeat again and again, e.g. navigation through multiple link chains, or filtering by some condition. To make the transformations more readable, the redundant parts need to be abstracted away. lQuery functions help to do it.

2.3 Example Model

In Fig. 2 we can see a simple model and an instance diagram. We will use it throughout the rest of the paper for demonstrating lQuery constructs. The model is on the left side, it consists of two classes: `Person` and `Animal`. `Person` has `name` and `age` attributes and associations to other persons that are his `parents` and `children`, and an association to `Animals` that are his `pets`. On the right side we can see a couple of instances of this model.

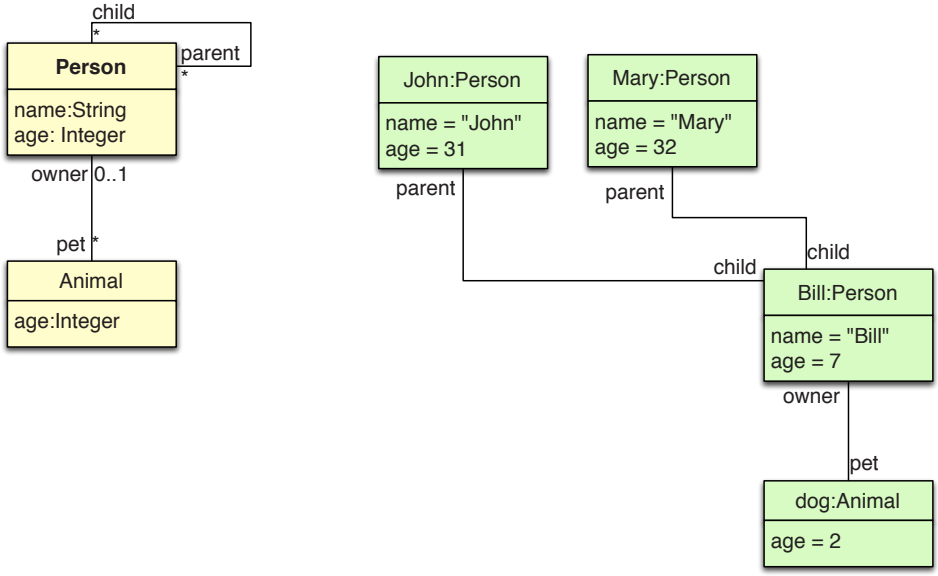


Fig. 2 Example model and instances

Typical queries that we would like to make on this model are: get instances of a particular class (e.g. all persons), get instances with a specific attribute value (e.g. persons with name `"John"`), or get all pets of a person's children. If we needed to perform these queries using only the repository API, then the code would mostly contain iterator constructs. For example, to get persons that are `42` years old, we would need to write:

```
persons_with_age_42 = {} --empty table for storing
results
for i, o in ipairs(allObjects()) do --iterate over all
objects
  --check that object is a person and the value of age is
42
  if isKind(o, "Person") and getAttrVal(o, "age") == 42
then
     --insert person into results table
     insert(persons_with_age_42, o)
  end
end
```

It is far from readable, even for such a simple query, especially if we compare it with path expressions from XPath language [5], where it would look something like `".Person[@age=42]"`. Our goal is to create a query language where selector expressions would be as compact as that. One way to do it is to create a function that receives an XPath-like selector string and returns the resulting object collection, but this

approach is too limiting because there are common queries that cannot be adequately represented as strings, e.g. getting objects with a link to a specific instance because in our repositories an instance does not have an externally accessible id, so it cannot be encoded in a string. That is way we will take another approach: we will define selector functions, and function combinators, so that we can easily reference objects and object collections by passing them as arguments to those functions. For the common cases, where string expressions would suffice, we will define an XPath-like selector shorthand notation (string expressions) that can be easily mixed with selector functions and combinators. The result will be the lQuery library.

2.4 lQuery Core

The core of lQuery is a single function: **`query`**. It has two arguments: an ordered collection of repository objects and a selector. The selector specifies what will be the result of the query operation on the source collection. There are two types of selectors: *filters* and *navigators*. Filters are used to return a subset of the initial collection based on some condition. Navigators are used to get a new ordered collection of objects from the initial collection. Examples of filter selectors are: filter by class membership, or filter by attribute value. Examples of navigation selectors are: getting the collection of objects that are reachable from current collection by a given role name, or getting the collection of values of some attribute. For each of these primitive selectors, there is a constructor function that creates it. Constructor function names have been chosen to maximize readability when used as arguments in query calls. The list of built-in primitive selector constructors is given in Table 1. For example, to get all persons from Fig. 2 that are 42 years old we would write:

```
persons = query(allObjects(), kind("Person"))
query(persons, hasAttrValue("age", 42))
```

Table 1

Primitive Selector Constructors

Selector Constructor	Description
`kind``(className)`	returns a filter selector that will match only those elements that are instances of a class with id *className* or instances of some class in its subclass chain
`hasAttrValue``(attrName, attrValue)`	returns a filter selector that will match only those objects that have an attribute with id *attrName* whose value is equal to *attrValue*
`linked``(roleName)`	returns a navigator selector that will match all those objects that are reachable by a link with id *roleName*
`attrValue``(attrName)`	returns a navigator selector that will return a collection of values that are associated to attribute with id *attrName*

It is much more concise than the same query written using the base repository API and an explicit `for` loop (see previous chapter). But there are still some problems, e.g. we needed to introduce a temporary variable: `persons`, and we have to call the `query` function twice. It would be better if we could combine the two query steps into one then we do not have to introduce a temporary variable, and we can call the `query` function only once.

Another problem is how to perform filters on more complex conditions. Currently, there are only two primitive filters: filter by kind, and filter by attribute value. If we need to make a more complex query, e.g. select persons that have at least one child, we have to resort to an explicit iterator.

```
parents = {}
for p in query(allObjects(), kind("Person")) do
  children = query(p, linked("child"))
  if #children > 0 then
    parents:insert(p)
  end
end
```

In the next chapter we will look at selector combinators that will address these problems.

2.5 Selector Combinators

In the previous chapter, we introduced the `query` function and some primitive selectors for filtering and navigation object collections, but they were not powerful enough to cover many typical use-cases. To solve those problems, we will introduce functions (*selector combinators*) for building new selectors from existing ones. They will receive selectors as arguments and return a new selector that can be used elsewhere as if it was a primitive. Let us look at a couple of selector combinators in more detail (the complete list of selector combinators is shown in Table 2).

One of the most frequently used selector combinators is **`chain`**. It receives any number of selectors as arguments, and returns a new selector that when evaluated in a query function will apply the first selector to the initial collection, then pass the result of that evaluation to the next selector and so on through all the selectors that where passed to it. Thanks to it, we can write long selector expressions in a very readable way, because we do not need to manually call query functions and pass them arguments. For example, to get all persons and then to get all animals that are pets of those persons, we can write:

```
query(allObjects(), chain( kind("Person"), linked("pet") )
```

Another frequently needed task is filtering not just by a predefined selector (like filter by kind, or filter by attribute value), but by a result of another selector. For this task, there are two selector combinators: **`has`** and **`hasNot`**. Selector combinator **`has`** accepts a selector as an argument and creates a filter selector, that when applied to collection of repository objects will return a new collection with only those objects for which the passed selector returns a *non*-empty collection. The selector combinator

hasNot works similarly, but returns the objects for which the passed selector returns an empty collection. For example, to select persons that have children

```
query(allObjects(), chain(kind("Person"),
                          has(linked("child"))))
```

Another pair of selector combinators is **union** and **intersect**. Both receive one or more selectors and return a new selector. In the case of **union**, the returned selector returns a union of object collections (multi set) of all the results of applying each selector to the initial collection. The **intersect** selector returns an intersection of object collection that are returned by all of the passed selectors. For example, let us say a person is responsible for someone, if that someone is either its child, or its pet. To get all persons that are responsible for someone we would use a filter and union:

```
query(allObjects(), chain(kind("Person"),
                          has(union(linked("child"),
                                    linked("pet")))))
```

The selector combinators `chain` and `intersect` can be interchanged in some situations, but in general they are different. When combining selectors with `chain`, each selector will be performed on the result of the previous selector, but when they are combined with `intersect` then all selectors are performed on the original collection and only then the results are intersected. When all the selectors are filters, chain and intersect can be interchanged and chain is actually the preferred, because it will be more efficient, i.e. every subsequent selector will be applied to a smaller collection of objects. But they will return a completely different result, if some of the selectors are navigators, because then the `intersect` will perform each selector in the context of source collection, but the `chain` will navigate through the chain of links. For example, `intersect(linked("children"), linked("pet"))` will return objects that are children and pets at the same time, while

`chain(linked("children"), linked("pets"))` will return children's pets.

The final combinator is **closure**. It receives one selector and returns a new selector that when applied to a repository object collection will return a new collection with all the objects from the initial collection together with objects that can be found by repeated application of the passed selector to the resulting collection until no new objects are found. It is impossible to go into an infinite loop, because closure will notice cycles and will not evaluate the passed selector on them again. A typical example for closure is to get all descendants of a person (here we assume that each person is a descendant of himself, in the next section we will see how to implement a combinator *closure_plus* that will not have this problem). The `closure` will first find all person's children, then find all his children's children, and so on, until no more children can be found. It can be written as follows, assuming that p is a collection of persons for whom we want to find all descendants:

```
query(p, chain(kind("Person"),
               closure(linked("child"))))
```

Combinator **closure** can be used not only with simple selectors like `linked`, but also with more complex selectors: like `chain` of links, or links followed by filters. For

example, if the class `Person` had an attribute `gender`, then we could create a selector for getting only male descendants by writing:

```
closure(chain(linked("child"),
              hasAttrValue("gender", "Male")))
```

Table 2

Selector Combinators

Selector Combinators	Description
chain(sel1, sel2, ..., selN)	creates a selector that applies each of the supplied selectors in order, first selector gets applied to the initial collection, and each subsequent selector is applied to the result of the previous selector
has(sel)	creates a selector that filters initial collection based on the result of supplied selector: if the result is a *non*-empty collection or a *non*-false value, then the object will be in the result collection, otherwise it will be dropped
hasNot(sel)	creates a selector that returns the complement of the one has selector would have returned
union(sel1, sel2, ..., selN)	creates a selector that returns a union of all supplied selector results
intersect(sel1, sel2, ..., selN)	creates a selector that returns an intersection of all the selector results
closure(sel)	returns a transitive closure of repeatedly applying the selector to the initial collection and then to each of results until no new object is added (checks for cycles and is not applied repeatedly if an object is found multiple times)

2.6 Selector Reuse and Custom Selector Combinators

When building any reasonably complex application, there usually are some selector patterns that repeat again and again, e.g. the compound selector from previous chapter for getting persons that are responsible for someone, i.e. that have a child or a pet. One way to avoid the repetition is to create this selector once and assign it to a variable. Later, when we need to use that selector, we can pass the variable instead of building it from scratch, like this:

```
responsible_persons = chain(kind("Person"),
                            has(union(linked("child"),
                                      linked("pet")))
query(allObjects(), responsible_persons)
```

This works if the pattern is constant, but what if the pattern is like a template? For example, we could want to get all objects that are reachable though a selector chain with length at least one. We can use functions to create these selectors for us. In a way, the selector combinators from previous chapter did just that. For example, to define a new selector combinator (`closure_plus`) that will receive a navigation selector and return a new selector that will match all objects that are reachable through a navigation chain with length at least 1, we write:

```
function closure_plus(selector)
  return chain(selector, closure(selector))
end
```

Now we can use this new selector combinator just as if it was a library primitive. In real life tasks, this allows the programmer to build a task-specific selector library on top of the primitive selectors and selector combinators that is tailored for his problem domain.

2.7 Custom Primitive Selectors

Although the ability to create higher-level selector combinators is very powerful, it is not enough, because we are still bound by the primitives that came with the library. There are situations when we need a genuinely new kind of selector that cannot be expressed with the existing primitives, e.g. get all persons from Fig. 2 whose name starts with a letter 'B'. Of course, we can always resort to explicit `for` loops, but the downside of this approach is that we cannot use them in our selector chains, i.e. we will have to split our chains in parts: till the `for` loop, and after it. The situation is even worse if we want to use that selection in the `closure` combinator, because there is no way to do it, and we would be forced to re-implement `closure` specifically for this case. To alleviate these problems, there is a mechanism for constructing new primitive selectors. In fact, all of the primitive selectors have been implemented through it.

There are two primitive selector constructor functions (Table 3). First operates in the context of one repository object, like primitive selectors returned by `linked` and `kind` constructors. The second operates in the context of repository object collection. The `closure` selector is implemented through it.

New selectors with single object context can be created with a function **`soloSelector`** that accepts a one-argument function as an argument (remember that functions are first-class objects in Lua, and can be passed as arguments—see section 2.1). When the resulting selector will be used in a query invocation, it will apply the passed function to each element from the initial object collection. It is expected that the function will return either a repository object, an object collection, or a boolean. If it returns an object or a collection, then all results are collected in a list that is flattened afterwards. If the functions returns a boolean, then it acts as a filter, i.e. only those objects for which it returned `true` are included in the result collection.

For example, if we were working with the repository that is shown Fig. 2 and needed to get all persons who have underage children, then we would have a problem, because

there is no selector for checking if an attribute value is less than a given integer, and would have to introduce an explicit `for` loop. But now we can construct a selector and use it with other combinators:

```
underage = soloSelector(function(p)
                            age = getAttrValue(p, "age")
                            if age < 18 then
                              return true
                            else
                              return false
                            end
                          end)

query(allObjects(), chain(kind("Person"),
                          has(chain(linked("child"),
                                    underage)))
```

Actually, all of the primitive selectors are implemented through `soloSelector`. For example, the primitive selector `kind(className)` is implemented like this:

```
function kind(className)
  return soloSelector(function(o)
                        return isKindOf(o, className)
                      end)
end
```

The second primitive selector constructor creates a selector from a one-argument function that will work on all of the initial collection at once, thus its only argument will be the initial object collection. The result of the passed function on the initial collection is the result of the whole selector. This selector constructor is useful for creating custom selectors that must have the whole object collection, e.g. getting the first object from a collection, getting the number of objects in a collection, or checking if an object collection contains a specific object. For example, to get the first child of every person we would first define a new primitive selector `first` (it is universal and can be used in other situations) and then use it to get the first child:

```
first = collSelector(function(coll)
                       return coll[1] -- table value by index
                     end)

query(allObjects(), chain(kind("Person"),
                          chain(linked("child"),
                                first))
```

Table 3

Custom Primitive Selector Constructors

Custom Selector Constructors	Description
soloSelector(fn)	creates a selector from a one-argument function; when the selector is used, the function will be applied to each element in the collection; if it returns an object or an object collection, then all the results will be collected and flattened; if it returns a boolean then it will act as a filter
collSelector(fn)	creates a selector from a one-argument function, in contrast to *soloSelector,* the whole object collection is passed to the function; the result of the function is the result of the selector

2.8 Shorthand Notation

The primitive selectors and selector combinators allow us to write complex query expressions in a modular and readable way, but in cases where the selector is constant and simple, the combinator approach is a bit longer than the analogous expressions in OCL [6] or XPath. To reach the maximum compactness and readability, we introduce a shorthand string notation for most common primitive selectors and combinators. The string form can be used anywhere in place of a selector: when the `query` function gets a string in place of a selector it will compile it to the corresponding primitive selector constructor or selector combinator calls. This allows us to mix the shorthand string notation together with ordinary selectors to achieve maximum compactness and expressiveness. Currently, there is no way to introduce shorthand notation for custom defined selectors and selector combinators, except for redefining the `compile` function.

The shorthand notation is adapted from the XPath navigation language. Function **`compile`**`(shorthand_string)` compiles a shorthand string into the corresponding selectors. It works as follows: string that starts with a dot followed by an alphanumeric string, e.g. `".ClassName"`, is compiled to the selector constructor `kind("ClassName")`, string that starts with a slash, e.g. `"/roleName"`, is compiled to `linked("roleName")`, and string that starts with brackets followed by `'@'` and a name, e.g. `"[@attrName = value]"`, is compiled to `hasAttrValue("attrName", "value")`. The shorthand notation for selector combinators is as follows: `":has(sel)"` is compiled to selector combinator `has(compile("sel"))`. The complete list of shorthand notation is given in Table 4.

Table 4

Selector Shorthand Notation

Shorthand Notation	Equivalent Form
`".ClassName"`	`kind("ClassName")`
`"/roleName"`	`linked("roleName")`
`"[@attrName = value]"`	`hasAttrValue("attrName", "value")`
`":has(sel)"`	`has(compile("sel"))`
`"sel1 sel2 ... selN"`	`chain(compile("sel1"), compile("sel2"), ..., compile("selN"))`
`"sel1, sel2, ..., selN"`	`union(compile("sel1"), compile("sel2"), ..., compile("selN"))`

Let us look at how some of the examples from previous chapters can be rewritten using the shorthand notation. The very first example was, get all persons that are 42 years old. Using shorthand notation we can write:

```
query(allObjects(), ".Person[@age=42]")
```

The shorthand notation can also be used in selector combinators. For example to get the descendants of person collection p, we write:

```
query(p, closure("/child"))
```

In that way, we can use the shorthand where possible, but fall back to selector combinators or custom selectors when the shorthand is not expressive enough.

2.9 Manipulation with Whole Sets of Objects

Selection of repository objects is only one part of the model interpretation task. The other, is actually doing something with the selected objects. Usually, the doing and the selection is intertwined, i.e. we select some objects, do something with them, and then use that collection to find next set of objects and do something with them. Because the repository API has functions only for manipulating one object at a time, we would have to use explicit iterators for manipulation and it would break up the selection-manipulation-selection chain into multiple statements that in turn would hinder readability. To avoid this problem, we define a number of methods for repository object collection that will allow us to manipulate sets of objects at once and intermix selection and manipulation steps. The list of methods is given in Table 5. We use the Lua object notation, where ':' is used for method invocation (see section 2.1 for details). Let us look at each method in more detail.

There are three manipulation methods: **`setFeatures`**, **`deleteLinks`**, and **`delete`**. Method **`setFeatures`** receives a Lua table as an argument. Each key in the table is a feature (attribute or link) name and the corresponding value is either a string for an attribute value, or an object or a object collection for a link value. The method adds the given features to each object in the source collection. In case of an attribute value, the

current value is replaced with the given value. In case of a link, new link assertion is created for the given object, or for each object in the object collection. For example, to set the attribute "age" of all persons from Fig. 2 to 18 and add a link "pets" to some object p, we would write:

```
p = createObject("Animal") -- create a new animal
query(allObjects(), ".Person")
  :setFeatures({
                 age = 18,
                 pets = p
               })
```

Method **deleteLinks** receives a Lua table as an argument, where each key is a link name and the corresponding value is either a single repository object or a repository object collection. The method deletes link assertions that correspond to the given key from each object in source collection to the corresponding key value. If there are no link assertions, then nothing is done. The result of this method call is the same collection on which it was called, so that further selection or modification operations can be done. For example, to delete the link "child" from all persons in Fig. 2 to persons whose name is "Bill", we would write:

```
persons_with_name_bill = query(allObjects(),
                               ".Person[@name = Bill]")
query(allObjects(), ".Person")
  :deleteLinks({
                 child = persons_with_name_bill
               })
```

Method **delete** removes all objects that are in the source collection from the repository and returns an empty collection.

There is also a higher-order method **each**(fn, args), i.e. a method that receives a function as an argument. It can be used to call some function on each object from the source collection for its side-effects, like making some changes in repository. The result of the method **each** is the same collection on which it was called. This allows us to make multiple such calls one after another. The supplied function fn will be called on each object in the source collection: its first argument will be the current object, and the rest arguments will be args, which where passed to the method each. For example, if we have defined a function for incrementing the attribute age by a given number, then we can make every person two years older as follows:

```
function increment_age (person, n)
  current_age = getAttrValue(person, "age")
  setAttrValue(person, "age", current_age + n)
end

query(allObjects(), ".Person"):each(increment_age, 2)
```

To allow mixing manipulation and selection steps, there is a method **`find`**`(selector)` that returns the result of the function `query` on the given collection and selector, i.e. `p:find(sel)` is equivalent to `query(p, sel)`. In addition the method creates a selection stack, so that each collection that is a result of the `find` method remembers from which collection it was derived. This information is used by a method **`back`**, to return the collection from which the current collection was derived. These two methods together with manipulation methods provide a very readable way to write tree-like visitors. To see these methods in action, let us consider a somewhat contrived example: we want to find all persons in Fig. 2, then increment the age of their children by one year and the age of their children's pets by two years, then we want to go back to the children and find a child with the name "Bill" ,rename him to "Bob", and delete his pets. To perform these actions, in the given order, we would write:

```
allObjects()
  :find(".Person")
     :find("/child")
        :each(increment_age, 1)
        :find("/pet")
           :each(increment_age, 2)
           :back()
        :find("[@name = Bill]")
           :setFeatures({name = "Bob"})
           :find("/pet")
              :delete()
```

Note that `allObjects()` returns an object collection, so we can use the method `find` on it. We use indentation to make the traversal more readable, i.e. after each `find` we increase the indentation to signify that we have a new object collection, after each `back` call we decrease the indentation to signal that we have returned to the previous collection. Also note that the result of methods `each` and `setFeatures` is the same collection they were called on (this style of methods is inspired by fluent interface approach to API design).

Although all of the previous examples used the shorthand selector notation in the `find` method, it is by no means the standard situation. In real life tasks, we would use custom selector combinators or predefined patterns, because in any complex task we would have built a domain specific selector language on top of the primitives.

Table 5

Object Collection Methods

Object Collection Method	Description
coll:find(selector)	returns a new object collection that is the result of applying query function to coll and selector
coll:back()	returns the collection from which the coll was derived
coll:setFeatures(feature_table)	feature_table is a table where each key corresponds to a feature name and each value corresponds to the new value of the feature; this method sets these values for each object in coll and returns the same collection coll
coll:deleteLinks(feature_table)	feature_table is a table where each key corresponds to a link name and value corresponds to the objects to whom the link must be deleted; the method deletes those links and returns the same collection coll
coll:delete()	deletes all objects that are in coll from repository, and returns an empty collection
coll:each(fn, args)	for each object in coll a function fn is called; first argument is the current object and the rest arguments are args; returns the same collection coll

3 Related Work

As far as we know, there are no libraries built specifically for model interpretation, i.e. optimized for selecting object collections, traversing them and making minor modifications, so we will compare lQuery to transformation languages, specifically the path expressions that are used in transformation languages, and to EMF Model Query library [7].

Transformation languages, as the name suggests, are optimized for matching patterns in source model and creating corresponding patterns in target model. Because navigation is not the most important problem in those tasks, transformation languages have either only one-step navigations through role names [8], or navigation expressions that have been inspired by OCL, like in languages ATL [9] and QVT [10]. But none of these

languages treat navigation expressions as first-class values, and thus it is impossible to build or change navigation expressions at runtime, or pass them as arguments to other functions. This makes them less usable in situations where the task at hand requires a construct that the language designers did not anticipate. For example, if lQuery did not have the `closure` combinator built-in as a primitive, it would be possible to add it as a user defined function, and later use it just as if it was a language primitive. Also, this ability allows a programmer to define a new higher-level selector language that will be tailored for his domain and thus abstract away from the specific details of the metamodel structure. This has two advantages: firstly, the code becomes more readable because the selectors are tailored for the problem, and secondly, if the structure of the metamodel changes, we only need to update our domain-specific selectors but all the logic can remain the same, because it is built on top of custom selectors.

EMF Model Query is a model query library that is part of Eclipse Modeling Framework [11]. It treats selectors as objects and can build them at runtime. But the resulting queries are in the style of SQL, i.e. `select-from-where`, where *from* and *where* clauses accept a structure that is similar to a lQuery selector. However, we think that XPath-like navigation paths, where navigation and filtering can be intermixed, are more readable.

4 Conclusions

We have shown how to build a query and transformation library for an EMOF-like data store in the Lua language. The library has just as readable selector expressions as transformation languages and in addition it can be easily extended with custom selectors using the full power of the host language. Also, the new custom selectors are indistinguishable from the ones that came with the library. In addition, the whole library, including the parser for shorthand notation, took less than 1000 lines of Lua code to implement. It shows that the library can be easily ported and that the amount of work could be comparable to writing a wrapper for an existing transformation language to work on a new repository.

Although the library is implemented in Lua, it could be just as easily ported to any other language that has first-class functions. The same can be said about data store: although the current library is implemented for an EMOF-like data store, it could be similarly implemented for a different structure, e.g. XML. The only thing that would change is the primitive selectors.

For the future work, we plan to explore how to make the compiler for shorthand notation extendable, so that shorthand can also be added for custom selectors and selector combinators.

Acknowledgements

This work has been supported by the European Social Fund within the project «Support for Doctoral Studies at University of Latvia».

References

1. Hutton, G.: Higher-Order Functions for Parsing. In: Journal of functional programming, Cambridge Univ Press (1992)
2. Ierusalimschy, R.: Programming in Lua, second edition (2006)
3. Meta Object Facility (Mof™) Core 2.0, http://www.omg.org/spec/MOF/2.0/, January 2006
4. Sproģis, A., Liepiņš, R., Bārzdiņš, J., Čerāns, K., Kozlovičs, S., Lāce, L., Rencis, E., Zariņš, A.: GRAF: a Graphical Tool Building Framework. In: ECMFA 2010 Tools and Consultancy track, France (2010)
5. XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath/, November 1999
6. Object Constraint Language (Ocl) 2.2, http://www.omg.org/spec/OCL/2.2, February 2010
7. EMF Model Query Developer Guide, http://help.eclipse.org/helios/index.jsp?nav=/22, May 2011
8. Kalnins, A., Barzdins, J., Celms., E.: Model Transformation Language MOLA. In: Proceedings of MDAFA 2004, 14–28 (2004)
9. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: ATL: a QVT-like transformation language. In: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, New York, NY, USA (2006)
10. Meta Object Facility (MOF) 2.0 Query/View/Transformation, V1.1, http://www.omg.org/spec/QVT/1.1, January 2011
11. EMF: Eclipse Modeling Framework, http://www.eclipse.org/emf/, May 2011

Appendix: Model Repository API

Object Collection Method	Description
getEntity(entityId)	returns the schema entity with the given id
allObjects()	returns a list of all objects that are in the repository
isKindOf(object, class)	returns true if the object is a member of the given class or a member of any class in the class‘ subclass chain
createObject(class)	returns a new object that is a member of the given class
deleteObject(object)	deletes the given object from the repository together with all its attribute values and link assertions
getAttrValue(object, attr)	returns the attribute value string that is connected to the given object and the given attribute
setAttrValue(object, attr, value)	replaces the string value of attribute value of the given object
getLinkedObjects(object, link)	returns a list of objects that are connected to the given object through the given link
existsLink(src_obj, link, trg_obj)	returns true if there exists the given link type between the source object and the target object
createLink(src_obj, link, trg_obj)	creates a new link assertion between the source object and the target object with the given link type; if the link type has an inverse, another link assertion is created in the opposite direction
deleteLink(src_obj, link, trg_obj)	deletes the link assertion that was between the source object and the target object; if there is an inverse link assertion in the opposite direction, it is also deleted

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2011. Vol. 770
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 46–55 P.

Testing Computer Skills In Productive Environment: Solution And Know-How

Andris Paikens, Zane Bicevska, Janis Bicevskis
Datorikas Instituts DIVI, Avotu 40-34, Riga, Latvia
Andris.Paikens@di.lv

The research discusses issues of training the computer skills and testing them in productive environment. Based on outcomes of the research project, authors describe recommendable solution for mastering office software applications skills and aspects of putting this solution into practice. Main conclusions of the research are based on data acquired and analyzed during approbation of solution in practice.

Keywords: digital literacy, testing of IT skills, eLearning.

1 Introduction

The eLearning initiative of the European Commission seeks to mobilize the educational and cultural communities, as well as the economic and social players in Europe, in order to speed up changes in the education and training systems for Europe's move to a knowledge-based society [1].

Modern, effective education and training systems are vital to everything from economic competitiveness to social inclusion [2]. Information and communications technologies (ICTs) are part of the answer, improving classical education and providing flexible learning solutions to people throughout their life. Vivianne Reding, Member of the European Commission responsible for Information Society and Media, announced: "ICTs can help improve education, life-long learning and social inclusion" [3].

Widespread digital literacy, however, is vital. To participate and take advantage, citizens must be digitally literate – equipped with the skills to benefit from and participate in the Information Society. Today, almost all workers need to be able to use ICTs, so training in using ICTs is both a key part of Europe's inclusion strategy and essential to reaping the benefits ICTs bring to education. The digital literacy includes both the ability to use new ICT tools and the media literacy skills to handle the flood of images, text and audiovisual content that constantly pour across the global networks.

Digital literacy is therefore one element in the i2010 Strategy's emphasis on Inclusion, better public services and quality of life. But this is not just about Inclusion –

ICT-related skills are vital for the competitiveness and innovation capability of the European economy.

In order to ensure internationally recognized computer skills certification in Europe, European Computer Driving Licence (ECDL) Foundation [4] was set up. ECDL provides international certification of computer skills and issues European Computer Driving License. ECDL Foundation's certification programmes have been delivered to over 10 million people, in 41 languages, across 148 countries, through a network of over 24,000 test centres. The certification programmes are designed to be accessible to all citizens, irrespective of age, gender, status, ability or race.

ECDL Foundation declares vendor neutrality [5]: "ECDL Foundation's certification programmes are vendor neutral. This means that our programmes are not tied to any one brand of software and are designed to enable people to gain skills that can be used in any appropriate software environment. This vendor neutrality gives people the freedom and flexibility to complete the training and testing, and demonstrate skills in a range of software applications."

However, complete vendor neutrality in computer skills testing process can be hardly observed. For instance, a project "Training in Computer and Internet Usage of Unemployed in Latvia" [6] implemented within the framework of European Community EQUAL by the Latvian Information and Communication Technologies Association (LIKTA), a member of ECDL Foundation issuing ECDL licences in Latvia, included training material „Basics for text processing". This material as recommended training material can be found on the Latvian ECDL certification homepage [7] includes Microsoft text processing tool Word as the only example of text processing environment. The situation remains the same in spreadsheet application training etc.

The open-source software products have an increasing market share in Europe and worldwide. For instance, the European portal www.webmasterpro.de published the results of its latest research [8] in February 2010. The study showed that in some EU countries market share of OpenOffice products reaches almost 22% (Poland, Czech Republic). In old EU member states, market share of OpenOffice products is: Germany – 21%, France – 19%.

This is another proof of increasing vendor neutrality in computer skills testing demanding adaptation of solutions for computer skills training and ECDL certification support tools to the changed market situation without causing unnecessary pressure and costs.

The research focuses on the following issues:

- Automated testing of computer skills used in a particular software environment;
- Applying the work assessment principles to the tests of computer skills;
- Testing the computer skills considering vendor neutrality principle.

2 Testing in Productive Environment

Bloom's Taxonomy [9] divides educational objectives into three "domains": Affective, Psychomotor, and Cognitive, each having domain specific skills to be developed.

Skills in the affective domain describe the way people react emotionally and their ability to feel another living thing's pain or joy. Affective objectives typically target the awareness and growth in attitudes, emotion, and feelings. Skills in the psychomotor domain describe the ability to physically manipulate a tool or instrument like a hand or a hammer. Psychomotor objectives usually focus on change and/or development in behaviour and/or skills.

Traditional education tends to emphasize the skills in the cognitive domain. These skills revolve around knowledge, comprehension, and critical thinking. There are six levels in the taxonomy, moving through the lowest order processes to the highest: knowledge, comprehension, application, analysis, synthesis, evaluation.

Thus, the knowledge is the very basic level of expressing skills – an exhibit memory of previously-learned materials by recalling facts, terms, basic concepts and answers. Developing skills at a higher level requires penetration, ability to analyze and evaluate and ability to apply the acquired knowledge in practice.

Computer skills refer to the level of cognitive domain application at least, since usage of a computer to the full extent is not possible without the ability to use specific applications. The process of learning computer skills including training is not possible without performing actions in certain environments. In such trainings, new knowledge is added to the existing knowledge of student through accomplishing tasks in practice.

Unfortunately, most tests of computer skills still involve testing techniques characteristic to the assessment of theoretical knowledge. Traditionally, it is organized as a test or exam where students provide answers to test questions in a free or structured or partly structured form. This approach, however, does not provide any significant proof to students' ability to use the particular environment. Moreover, this approach induces assumption that computer skills are derived from knowledge about computer skills that is generally not correct.

Testing of practical skills, on the other hand, shows student's ability to work with the particular application or in the particular environment where skills are learned. Controlling of these skills, as a rule, is difficult to automate since this would imply that testing software controls user's actions in a productive environment. This approach is called productive knowledge testing. Mostly recognized commercial product that supports this approach is EnlightKS [10].

Testing of practical skills includes various techniques and methodologies, and often combinations of them:

- Questions combined with images involves classic testing approach, however possible answers to test questions contain graphical information close to examples in real environment;
- Semi-practical tasks contain in advance prepared images of the environment where skills are to be tested. Correctness of the answers is determined by verifying relative or absolute position of computer mouse cursor in a given image against pre-defined mouse cursor position of the correct answer. Task images can be grouped or combined in series or presentations;
- Simulation of work environment is implemented by means of special software visually resembling the environment where skills are to be tested but being technically less sophisticated letting the user to perform limited set of actions

necessary to accomplish the task. Interactive videos are one of such possible ways of work environment simulation;

- Verification of work results is performed in a productive environment where specialised software solution verifies the result of performed actions against the correct answer;
- Assessment of user's actions is performed in a productive environment where specialised solution continuously verifies user's actions in order to determine not only correctness of these actions but also sequence and way how these actions were performed. Implementation of this approach requires very close integration between the testing software and the environment where the test is performed.

Verification of work results was considered as determinative testing methodology within the given research and experimental development since:

- the technical implementation is not restricted by a single version of productive environment – the test uses productive environment that is available on user's computer;
- the primary task of the test mechanism is to verify user's ability to perform the task, however, does not limit methodology or the way how the user reaches results within the given environment (it is acceptable not to perform the task in an optimal way);
- the automated verification of test results decreases the risk of human errors present in manually processed test results.

3 Implementation

The solution was implemented within the framework of experimental development applying findings of scientific research into practice. The developed prototype was approbated for two office desktop application packages – Microsoft Office and OpenOffice – mostly used Office software programs within the European Union.

The solution supports operations with different office desktop application packages, thus extending application area of the project outcomes, at the same time increasing complexity of technical aspects. Support is provided for OpenOffice version 3.2; Microsoft Office receives support for versions 2000 to 2007. Significant aspect of the project outcome – the solution supports also different operating systems such as MS Windows XP, MS Windows Vista and MS Windows 7. Enabling support for operating systems was one of the most sophisticated problems, mainly, due to different security settings and other technical nuances.

3.1 Methodology

Implementation of solution is based on generalized automated testing methodology design for universal application, respectively, independent from office software vendor. Moreover, a self-evident request was the independence from different product versions from the same vendor.

The main methodological task was to define a set of simple, measurable, separable and autonomous units that would cover the area of testable knowledge and skills.

Knowledge and skills were divided into smaller units. Each such unit includes a small set of knowledge and skills that undergo no further detailing. Each of these units can be used separately or in combination for testing and training purposes accordingly to the difficulty level and knowledge area.

It was important that the standard specification for computer skills of an office desktop application without specifying a particular product is already considered in the design phase. Thus, the solution had to function equally well and independently from the chosen application.

Such an approach complies with the requirement to provide a possibility to design test tasks without taking interest in which environment tests will be performed. Students would receive equivalent evaluation independently of the environment they took test in.

During the research process, several potential problems and ambiguously interpretable opportunities were identified. These problems mainly comprised cases when skills for different products or different product versions have to be tested by different methods.

Purposely not restricting probable outcomes of the research each probable problem situation was evaluated separately – eventually some of them were excluded from test unit set, but, however, some were implemented. As a result, the developed testing software provides more features to end users, though a partial responsibility for the outcome lies on the person who defines the tasks. Within the development phase of solutions, test units were described and developed. With the help of these test units, sets of practical tasks for testing computer skills were defined. Each of these units comprises logically simple tests, and each unit corresponds to a specific ECDL requirement, e.g., a skill to apply a specific font to text or to change its colour or size.

3.2 Technical platform

Since office applications are usually made as desktop applications, the productive testing environment should also have a form of a desktop application.

The development was performed in VisualBasic.Net for Windows environment. Good compatibility with MS Office applications was a strong argument to choose this development environment. However, this methodology can be successfully implemented also with other application development tools.

Each of the test units was developed as a separate VisualBasic.Net function whose tasks included test object and its properties to be checked. The system returns true, if the property named by the argument exists and its value matches one defined as function's argument, and false, if there is no property or property value does not match the defined desirable outcome. Specification, development and testing of such test units comprised a very significant portion of project time resources.

Considering autonomic nature of each unit within the given implementation, unit testing method was used to test software units. This method provides comprehensive overview of status of completion and quality of developed skills evaluation units.

3.3 Architecture

Although the selected solution is not complex and it has homogeneous implementation logic, the practical implementation involved several technically sophisticated issues.

Data gathering in format that would enable testing of unit logic presented the most significant problem during implementation.

For instance, if logical test *IsFont* includes simple test or an element with a specific font, then the implementation should include testing the object itself and accessing its components and properties. In a particular case, if the test would imply a MS Word document as a result, then in order to check fonts in a part of the document, the test mechanism should know how to access this part of the document and its properties.

The above-described access to objects has been implemented by means of interpreter. The interpreter has three interconnected components:

- Data model – a structure where processed document data are filled;
- Application interface – a subsystem providing data gathering and usage;
- Test units – the implementation of the logic of the test.

Architecture of the interpreter is shown in Figure 1.

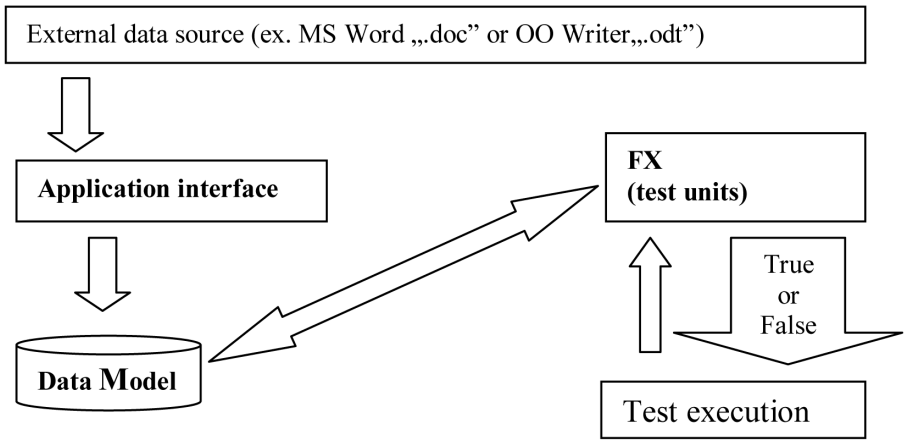


Fig. 1 Architecture of interpreter

Data model provides a possibility to store information needed for testing logic of units. Moreover, data structures are general by nature, namely, independent from source and manner of use.

Data loading is organized via help of application programming interface (API) classes. Initially, general classes were defined for description of general interface. Further, significantly different API classes were implemented for specific applications in MS Office and OpenOffice.

The component of test units derives values form the data model and executes the test logic in units.

This kind of architecture enables complementing the API without significant changes in data model and testing units in order to add support for other office software products in the future.

4 Implementation of Application Programming Interface

Due to specifically different applications, the development of application interface was the most sophisticated part in the implementation of testing interpreter.

Implementations were different in both cases, MS Office and OpenOffice:

- Significant discrepancies in applications' measurement units;
- Even within one program package such operations as program start and storage of results were different in each application. For instance, Open Office regards diagram saved in MS PowerPoint format as a picture. However, a diagram created in OpenOffice is not recognized by MS PowerPoint at all;
- Different conception in other significant matters, for instance, MS Word and OpenOffice Writer, have different algorithms for word counting in a document.

Next chapters deal with comparative overview of technical implementation.

4.1 Processing methods for MS Office objects

For processing MS Office documents Microsoft recommends using Primary Interop Assemblies [11].

A primary interop assembly is a unique, vendor-supplied assembly that contains type definitions (as metadata) of types implemented with COM. There can be only one primary interop assembly, which must be signed with a strong name by the publisher of the COM type library. A single primary interop assembly can wrap more than one version of the same type library.

A COM type library that is imported as an assembly and signed by someone other than the publisher of the original type library cannot be a primary interop assembly. Only the publisher of a type library can produce a true primary interop assembly, which becomes the unit of official type definitions for interoperating with the underlying COM types.

Publishers of COM components produce primary interop assemblies and distribute them to developers for use in .NET Framework applications.

To use the features of a Microsoft Office application from an Office project, the primary interop assembly (PIA) must be used for the application. The PIA enables managed code to interact with a Microsoft Office application's COM-based object model [12].

4.2 Processing methods for OpenOffice objects

UNO Project provides processing of OpenOffice documents [13]. UNO stands for Universal Network Objects. UNO is one of the accepted projects of OpenOffice.org.

UNO is the component model of OpenOffice.org. UNO offers interoperability between programming languages, other component models and hardware architectures, either in process or beyond process boundaries, in the Intranet as well as in the Internet. UNO components may be implemented in and accessed from any programming language for which a UNO implementation (AKA language binding) and an appropriate bridge or adapter exists.

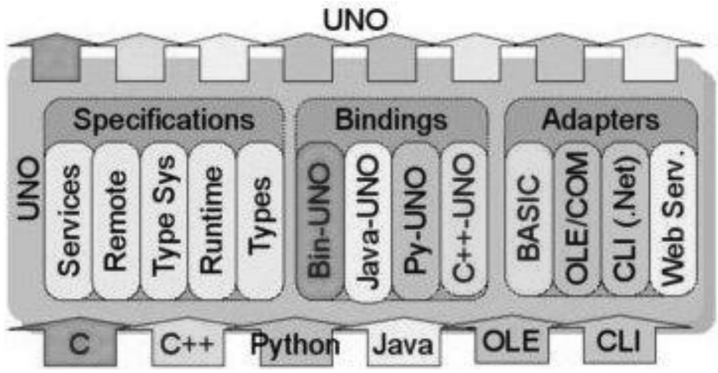


Fig. 2 OpenOffice possible usages using UNO project

4.3 Comparative evaluation

The implementation process contained a sophisticated challenge to overcome differences among PIA and UNO in order to develop solution according to specification. Moreover, most UNO documentation is written for Java programming language. Samples for usage with other languages were scarce or there were hardly any.

Within last years, OpenOffice has brought in several changes in object structure by extending UNO features, thus facilitating access to object information. However, these changes caused ambiguity regarding the volume of available information, namely, the documentation is not brought up-to-date and huge volume of available information still refers to outdated interface versions, thus hindering the research efforts.

By using PIA, it is possible to auccess different object properties and data units easily. However, PIA enables working with „com" [14] objects only; therefore, it does not provide a possibility to review „com" object errors in problem cases.

UNO has a different framework, however its discrepancies with PIA due to complex access to data details are only perplexing development. Moreover, in comparison to PIA, in UNO, the data set requires additional processing efforts in order to obtain the needed data. In the case of PIA, it was comparatively easy to obtain object's properties and process its values. In the case of UNO, additional efforts were required to both identification of the needed property among others in the set and obtaining the needed value form all available values of a property.

Some difficulties were caused by the necessity to use proper data objects in order to access further properties. False objects did not have appropriate properties, but finding the proper objects was hindered by the scarce and incomplete documentation.

5 Results and Conclusions

The ideas developed during the project implementation were transformed into prototype, which was approbated for testing purposes of computer skills in office software environments of two different vendors.

During development phase, 140 units of skill testing were specified. Implementation was accomplished for 120 such units. Due to various logical inconsistencies between office software from two different vendors discovered during the prototype approbation phase, an implementation of an interpreter (a practical implementation of the prototype design) could not be done equally for different office software.

The results of the project are both ideological and technological by nature.

Benefits of automated testing of computer skills in productive environment are the following:

- Optimized testing process of computer skills – significant reduction in time needed for testing and gathering the results, significant reduction (in most cases even total absence!) of evaluation errors. Unified and neutral attitude towards every student is ensured;
- Complete identity between software product versions used in test and as a working environment is endured – if the student is able to perform the action within the skill testing environment, then it can be asserted that she/he will be able to master these skills in a real working environment as well;
- Accumulated data base of computer skill testing – tasks, examples, good practice – that can be used in long term period and provide good basis for the improvement of teaching and training methodology;
- Automatically accumulated valuable statistics – valuable data on tests, test results both on general and detailed level (e.g. dynamics of a student's development) can be obtained without significant efforts and additional resources;
- Computer skill testing materials (sets of tasks, etc.) can be used for training purposes as long as solution can technically „replay" testing process step-by-step.

However, a few restrictions and challenges for the use of solution exist:

- The leading software applications are subjects to permanent improvement. Although the newest product versions are supposed to be compatible with the previous ones, technical implementation of the newer versions can significantly differ from the previous versions, thus causing necessity to continuously maintain automated testing solution's consistency with market development;
- In order to comply with the principle of vendor neutrality, testing the computer skills would have to use only generalized and standardized cluster of features characteristic to most environments, where skills are tested;
- Generally, every environment eligible for testing skills needs a technically individual solution in order to automatically verify students' test results against correct answers.

Within the framework of this research project, the following objectives were accomplished to a full extent:

1. Specification of testing skill units;
2. Development and prototypes of interpreters;
3. Testing the correct operation of a prototype during its testing phase;
4. Approbation and testing the end version of interpreter prototype.

6 Acknowledgement

The research is supported by the European Regional Development Fund (ERDF).

References

1. http://ec.europa.eu/education/archive/elearning/index_en.html
2. http://ec.europa.eu/information_society/tl/edutra/index_en.htm
3. Reding, V.: i2010: A new start for Lisbon and for European Information Society and Media policies. The i2010 Conference, London, 6 September 2005
4. http://www.ecdl.org
5. http://www.ecdl.org/index.jsp?p=94&n=928
6. http://www.latvijapasaule.lv/
7. http://www.ecdl.lv
8. http://www.webmasterpro.de/portal/news/2010/02/05/international-openoffice-market-shares.html
9. Bloom, B., Englehart, M. D., Furst, E. J., Hill, W. H., Krathwohl D. : The Taxonomy of Educational Objectives, The Classification of Educational Goals, Handbook I: Cognitive Domain, 1958
10. http://www.knowledgeuk.com/index.php/main/show/0/5/Home_Page
11. http://msdn.microsoft.com/en-us/library/aax7sdch.aspx
12. http://msdn.microsoft.com/en-us/library/15s06t57.aspx
13. http://udk.openoffice.org/
14. http://en.wikipedia.org/wiki/Component_Object_Model

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2011. Vol. 770
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 56–67 P.

File Transfer Protocol Performance Study for EUMETSAT Meteorological Data Distribution

Leo Truksans, Edgars Znots, Guntis Barzdins
Institute of Mathematics and Computer Science
University of Latvia
leo.truksans@lu.lv, edgars.znots@lumii.lv, guntis.barzdins@lumii.lv

The described study provides the experimental results and their analysis for selection of the file transfer protocol to be used in the upcoming Meteosat Third Generation Programme requiring sustained dissemination data rate in the range of 300-400Mbps over heterogeneous networks. This dissemination speed cannot be easily achieved with default TCP protocol settings and file transfer applications with significant round trip time (RTT) and packet loss typical to large heterogeneous networks. The designed test lab allowed finding the optimal TCP protocol and file transfer application settings reaching the target data rate at 70ms RTT and 10^{-6} packet loss, typical to terrestrial networks. Meanwhile, none of the surveyed applications were able to reach the target data rate at 700ms RTT, typical to satellite distribution networks.

Keywords: computer networks, data dissemination, satellite communications.

1 Introduction

In January 2010, the European meteorological union (EUMETSAT) commissioned IMCS to perform a detailed study on currently available open standards file transfer protocols for TCP/IP networks. The purpose of the study was to provide the background experimental material for the selection of a file transfer architecture in the upcoming next generation Meteorological weather satellite system to be launched in 2014 – EUMETSAT.

The results obtained in this study could be of interest to much wider audience, as there are much ungrounded myths about the performance of underlying TCP protocol and data transfer applications built on top of it. Under permission from EUMETSAT, in this paper we provide a condensed version of the original technical report.

The purpose of the study was to perform a multi-dimensional survey of four file transfer protocols (FTP, UFTP, bbFTP, GridFTP, RSYNC) under widely varying conditions characteristic to various network conditions. Namely, performance 70ms and 700ms RTT characteristic to intercontinental terrestrial Internet and geostationary

satellite communications were studied. Additionally, various packet loss patterns were examined.

The measurements were conducted in controlled laboratory environment, which was meticulously fine-tuned and validated to ensure that the lab setup itself could not be the cause of negative artifacts during measurements. The lab itself was built from open-source components rather than from closed commercial network emulators. This enabled full tunability of the network simulator performance characteristics and parameters (e.g. insertion of various packet loss patterns: random packet loss, packet loss in random bursts, etc.) – we were not limited by the constraints of the given test platform.

2 Test Lab Description

As depicted in Fig. 1, a single test bed (two identical test bed sets were used during this study) consisted of a file transfer server connected via LAN switch with one of the clients and a network simulator. Second client was placed behind a network simulator. Switch port connected to the server was mirrored and all traffic originated from or sent to the file transfer server was copied to the traffic monitoring server. In case of unicast file transfer scenarios, data was sent between the server and client behind the network simulator. For multicast scenarios, data was sent from server to both clients. All machines had at least dual 1GbE NICs, and each machine had a separate interface used for management purposes only.

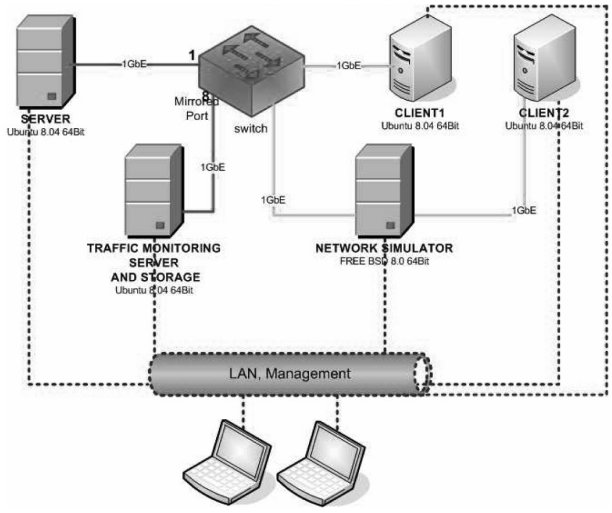


Fig. 1 Test lab topology

2.1 Hardware

All servers used within the test bed achieved or surpassed the necessary performance levels to ensure that results obtained in the study were not biased due to performance bottlenecks in equipment used.

Network simulator had AMD Opteron 148, 2.2 GHz single core CPU, 2GB RAM, dual Broadcom NetXtreme 1GbE network interfaces (BCM5704), 160GB SAMSUNG HD160JJ WU100-33 HDD.

File transfer server, clients and traffic monitoring server had two AMD Opteron 275, 2.2 GHz, dual core CPUs, 8GB RAM, dual Broadcom Tigon3 1GbE network interfaces (BCM95704A7), 80GB WDC WD800JD-00LSA0 HDD.

A small but capable HP ProCurve Switch 1800-8G was used for LAN connectivity.

2.2 Software

Network simulator operating system was FreeBSD 8.0-RELEASE. Network simulator software: ‚ipfw' and ‚dummynet' subsystems in the default system kernel.

File transfer server, clients and traffic monitoring server had Ubuntu Linux 8.04.4 LTS, 64-bit operating system. Usage of 64-bit kernel was essential for optimal memory addressing and necessary for large TCP buffers. File transfer applications: ProFTPd 1.3.1, vsftpd 2.0.6, bbFTP 3.2, GridFTP 4.2.1, UFTP 2.10.3, RSYNC 2.6.9. Traffic logging software: ‚tcpdump', default version provided with distribution.

2.3 Network tuning

After the default server installation, TCP parameters for all machines were tuned for better TCP throughput.

The following system configuration variables were tuned in all Ubuntu servers in accordance with best current practice [1]:

```
#Tuning Linux TCP settings
sysctl net.ipv4.tcp_window_scaling=1
sysctl net.ipv4.tcp_timestamps=1
sysctl net.ipv4.tcp_sack=1
sysctl net.ipv4.tcp_moderate_rcvbuf=1
sysctl net.ipv4.tcp_syncookies=0
sysctl net.ipv4.tcp_no_metrics_save=1
sysctl net.ipv4.tcp_ecn=1
sysctl net.ipv4.tcp_adv_win_scale=7

#Tuning Linux TCP buffers
sysctl net.core.rmem_max=16777216
sysctl net.core.wmem_max=16777216
sysctl net.ipv4.tcp_rmem=»4096 16000000 180000000»
sysctl net.ipv4.tcp_wmem=»4096 16000000 180000000»

#Tuning network interface parameters
ifconfig eth1 txqueuelen 10000
```

After initial server distribution installation on the network simulator, FreeBSD kernel was recompiled to enable Dummynet network simulator functionality, as well as to increase kernel time resolution to 40000Hz for more precise and consistent RTT

simulation. After kernel recompilation, both network interfaces were configured in single virtual bridge, so that traffic was transparently passed through FreeBSD network simulator between Ubuntu server and client machines on both interfaces.

2.4 Test bed validation

In order to validate test bed host capability to execute all test cases and produce correct measurements for scenarios specified in the study, several baseline performance measurements were performed. Initially, raw TCP and UDP throughput was measured for test bed hosts connected in a back-to-back configuration. After initial measurements, another set of test runs was performed with addition of switch between test bed hosts. Also, baseline RTT measurements for back-to-back and switched cases were performed for later comparison with test bed configuration accommodating network simulator.

Back-to-back, test bed hosts achieved raw TCP throughput of 941Mbps and raw UDP throughput of 957Mbps. RTT of 73µs – 7.8ms (avg. 76µs) in a back-to-back configuration was measured. The highest RTT was observed for the first packet, and was caused by necessity to perform the ARP request. The standard deviation of 26µs shows reasonably timed and predictable network interface operation.

Addition of a switch between test bed hosts did not have any significant effect on achievable TCP/UDP throughput or RTT. Switched performance was exactly the same as in case of back-to-back configuration – 941Mbps TCP and 957Mbps UDP. RTT was a little lower – 61µs-103µs (avg. 66µs) with a low standard deviation of 11µs.

Rapid sending of 100,000 ICMP ECHO requests (further – ‚ping flood‘) was used for measuring consistency of introduced RTT at network simulator. Iperf TCP and UDP tests were carried out to measure raw TCP and UDP throughput. These tests were run for one hour. RTT measurements were performed for first 100,000 ICMP ECHO (ping) packets.

At no packet loss and no introduced delay the raw TCP and UDP performance was again 941Mbps and 957Mbps, respectively. The RTT slightly increased to the range of 175-411µs (avg. 330µs) with a standard deviation of 44µs. These numbers demonstrated network simulator performance as stable and low impact on packet flow [2].

Results from measurements of RTT consistency at pre-configured RTT of 70ms and 700ms showed that maximum observed deviation from specified RTT was 1.9ms, average deviation from specified RTT was 0.58ms at 70ms RTT and 1.2ms at 700ms RTT. Since this falls well below 1% error margin relative to specified value, network simulator RTT simulation could be considered as consistent.

3 Testing Methodology

To understand and demonstrate the practical limitations of selected applications and protocols, a plan of test scenarios was created. The scenarios fall into six categories as detailed in the Table 1: small unicast, medium unicast, large unicast, mixed unicast, mixed multicast, large multicast. All the tests were run for one hour, except test 19 which was run for 5 hours. The 5th category used a mix of all file sizes with all RTT variants but with only two packet loss rates (10^{-6}, 10^{-3}). The 6th category used just 2GB large files and the worst packet loss (10^{-3}).

Table 1

All test scenarios

Scenarios	Category: file sizes	RTT	Packet loss rate
1,2,3,4,5,6	Cat1: 10kB	1,2,3: 70ms 4,5,6: 700ms	1,4: 0 2,5: 10^{-6} 3,6: 10^{-3}
7,8,9,10,11,12	Cat2: 5MB	7,8,9: 70ms 10,11,12: 700ms	7,10: 0 8,11: 10^{-6} 9,12: 10^{-3}
13,14,15,16,17,18	Cat3: 2GB	13,14,15: 70ms 16,17,18: 700ms	13,16: 0 14,17: 10^{-6} 15,18: 10^{-3}
19,20	Cat4: mix of 10kB, 500kB, 5MB, 50MB, 2GB	19: 70ms 20: 700ms	0
21,22,23,24	Cat5: mix of 10kB, 500kB, 5MB, 50MB, 2GB	21,22: 70ms 23,24: 700ms	21,23: 10^{-6} 22,24: 10^{-3}
25,26	Cat6: 2GB	25: 70ms; 26: 700ms	10^{-3}

During each test, the tcpdump utility on traffic monitoring server was used to capture first 68 bytes of each packet. After each test, raw captured data was uploaded to a separate system with large storage for off-line analysis. The analyses allowed to account data packets separately from protocol control messages and was protocol specific. All performance indicators in this study are for actual data without protocol overhead.

To better observe behavior of the protocols and applications, two histograms were generated for each scenario:

- 1st histogram: 5 second period for the 70ms RTT scenarios; 50 second period for the 700ms RTT scenarios;
- 2nd histogram: 15 minute period for all scenarios.

The reason why the first histogram represents the longer period of 700ms RTT scenarios is that in most such scenarios little data throughput was observed during the first 5 seconds. These seconds were mostly used for session initiation and negotiation. It can be seen in the actual histograms that traffic pattern of 5 second period at 70ms RTT was very similar to that of 50 second period at 700ms RTT, given other parameters equal. This confirms earlier observations that TCP throughput is inversely proportional to RTT [3].

Although all the tests were run for at least 1 hour, the 15 minute interval was determined to demonstrate at least one full session, yet be short enough to distinguish session cycles.

Graphs are drawn in logarithmic throughput scale to better illustrate performance patterns in presence of highly disproportionate absolute values.

4 File Transfer Applications and Test Results

The following applications performance results were obtained based on data gathered during this project. For every application, the impact of different file sizes and packet drop rates at fixed RTT was demonstrated. We chose to base observations on fixed RTT because from all the condition variables particularly RTT seemed outstanding for two reasons:

- Both RTT values (70ms and 700ms) represent different real life usage scenarios. The first is a representative RTT for a global terrestrial network. The second is extremely high RTT appropriate for geostationary satellite communication;
- All the protocols and applications tested in this project demonstrated substantial performance decrease in 700ms RTT link. The best case at this latency was GridFTP in scenario 16 reaching throughput of about 38MBps (304Mbps).

4.1 70ms RTT

Fig. 2 shows seven FTP scenarios: 1, 2, 7, 8, 13, 14, 19. They represent all three file sizes for 70ms RTT and 0 or 10^{-6} loss. The graphs have been grouped in 3 pairs – each for a different file size. The performance differs dramatically: the large files (2GB) are transferred at about 100MBps while small files (10kB) are transferred at about 20kBps. It can also be concluded that low or no packet loss does not impact the average performance much due to Fast Retransmit [4]. Transfer of mixed file set in scenario 19 shows varying average performance as it increased during the transfer of larger files and decreased during the transfer of small files.

Fig. 3 shows the same set of scenarios for GridFTP. Scenarios 13 and 14 show that the full benefits of GridFTP protocol were achieved when transferring large files over parallel streams.

Usage of multiple streams and TCP window scaling proved to be most advantageous for medium and large files on links with no packet loss, since GridFTP has the opportunity to accelerate and reuse each data connection. But, this same approach seems to be less effective in presence of packet loss. The larger is the file transferred over a TCP connection, the more likely is that some packets will be dropped for that file. Thus, the longer a TCP connection is used on a lossy link, the more likely this connection will experience degrading throughput over time due to several close packet drops that cause decrease of TCP window. Hence, re-use of existing data connections in case of high packet loss is undesirable. It is much more effective to use a connection only for a lifetime of singe file transfer, and then reopen new connection with the highest possible initial TCP window.

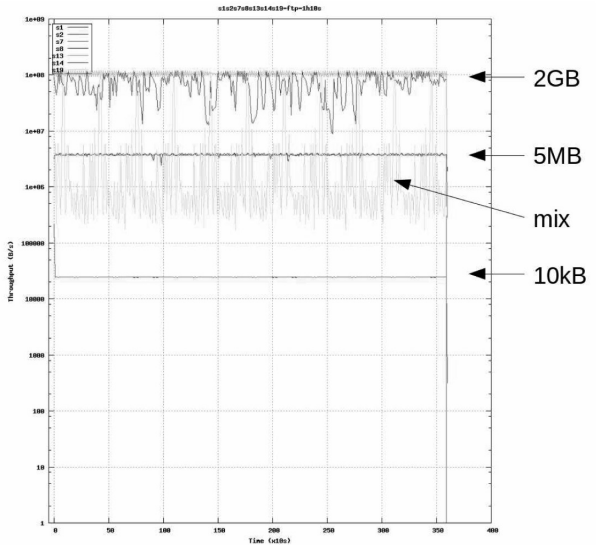


Fig. 2 FTP protocol, little or no packet loss, 70ms RTT

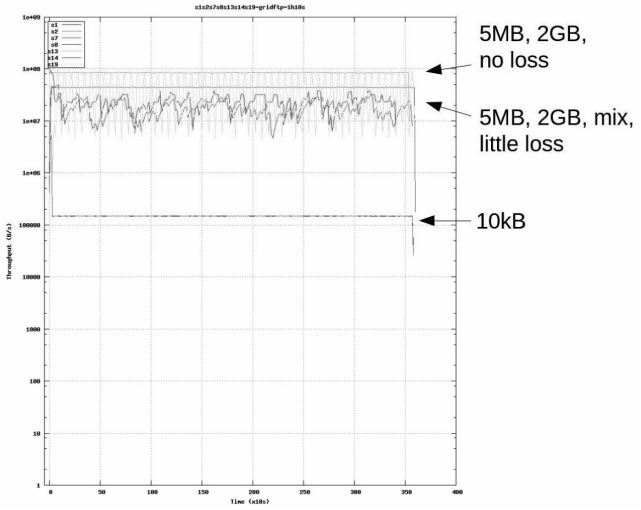


Fig. 3 GridFTP protocol, little or no packet loss, 70ms RTT

This explains why throughput curves are almost identical for scenarios 8 and 14 – because if the same data connection(s) are reused for the whole duration of file transfer session, the TCP window in both cases will converge to the same size depending on

packet loss rate, and irrespective of whether medium or large files are being transferred. As it can be seen, in case of high-throughput of large files packet loss rate is much more important than RTT.

4.2 700ms RTT

Fig. 4 shows seven other scenarios for FTP: 4, 5, 10, 11, 16, 17, 20. They represent all three file sizes for 700ms RTT and 0 or 10^{-6} loss. It can be easily seen that the graphs are grouped in 3 pairs again – each for a different file size. The performance also differs dramatically but the throughput is significantly lower. Rare packet loss impacted average performance of the large file transfer in scenario 17. Occasional packet drops close one to another made TCP to reduce window and drop performance in few of the transfers. This is indicated by the "ladders" in the graph. Comparing Fig. 4 to the Fig. 2 (FTP, 70ms) an observation can be made that traffic patterns are "stretched" about 10 times as RTT becomes 10 times longer. At the same time, performance decrease was about 10 times as the RTT was increased 10 times. As mentioned previously, it complies with findings in [3].

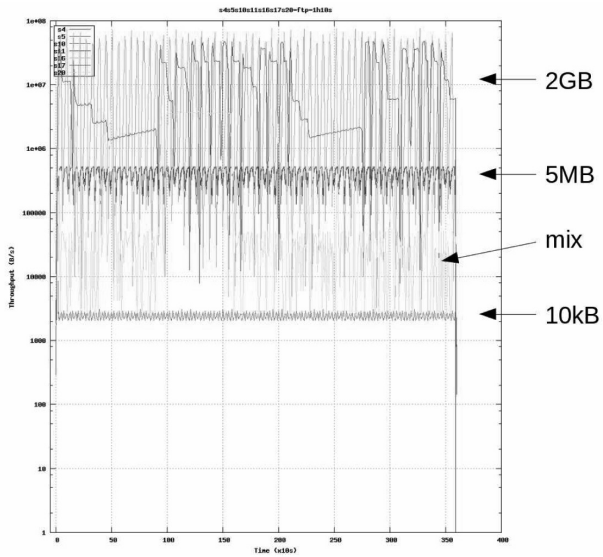


Fig. 4 FTP protocol, little or no packet loss, 700ms RTT

Fig. 5 shows all three file sizes for 700ms RTT and 10^{-3} packet loss. At this packet drop rate, large and medium file sizes show degraded performance while small files show no difference. Larger RTT decreased performance even more. As a result, 2GB file transfer at worst conditions was possible only at average throughput of 50kBps.

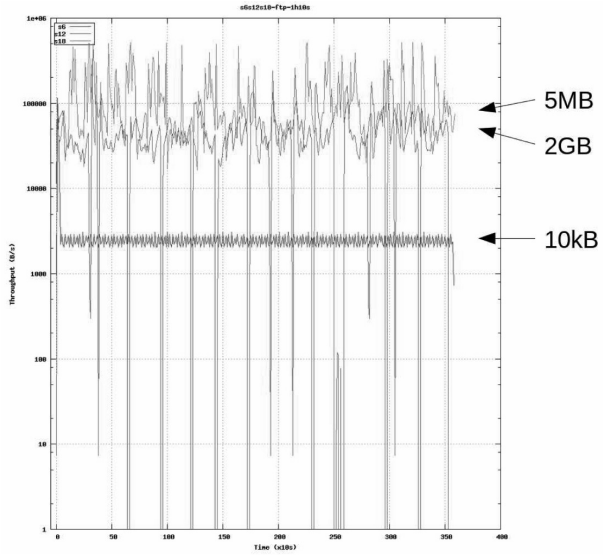


Fig. 5 FTP protocol, high packet loss, 700ms RTT

As in case with 70ms RTT, also here performance with large files was slightly lower than performance with medium files. The reason is the same – fast starting TCP could transfer most 5MB files with few or no lost packets and did not need to decelerate [5].

5 Conclusions and Recommendations

5.1 Conclusions on application suitability

The criterion of application suitability in this study was the earlier defined *target* performance. The target performance level was specified by the EUMETSAT as the minimum throughput of 350Mbps (43,7MBps) for the next generation real time content delivery.

5.2 FTP suitability

FTP protocol mandates a separate TCP connection for control session and a new TCP connection for every data stream. A data stream is either a data file or a directory listing. FTP commands get sent over single permanent control connection. Inefficiency of the multiple file sending process can be clearly seen in the analyzed data and histograms given earlier in this document. It took at least two round-trips to initiate a new file download in the presence of an already open session. In case of 70ms RTT, it meant at least 140ms lost in protocol "chat" for every file regardless of its size.

The analyses show that FTP reached *target* only in scenarios 13 and 14, which were sending 2GB files at 70ms RTT and no or rare packet loss. None of the other FTP scenarios reached *target* as a consequence of either smaller files, higher RTT, more frequent packet loss, or a combination of these.

5.3 UFTP suitability

UFTP application spent at least 4 seconds for every session initiation in our tests. The sender was always explicitly provided with IP addresses of both receivers. Session initiation took even longer time in case of open client participation. UFTP had to start a new session for every file it sends. As a result, UFTP application is unsuitable for sending files of small or medium size. Those are even out of scope in a UFTP design and performance study [6] that focuses only on large files.

UFTP reached *target* only in scenario 25, which was sending only large (2GB) files. The other parameters were: 70ms RTT and high packet loss. One of the reasons why this scenario was added to the test plan was to show at least one multicast scenario when UFTP reaches *target*. None of the other UFTP scenarios (mixed file sizes) reached *target* as a consequence of either smaller files, higher RTT, or a combination of both.

Scenario 25 shows another interesting point. UFTP does not suffer much from packet drops. Its delivery process does not retransmit a lost packet immediately following a NACK. Instead it continues to transmit file at the given rate (900Mbps in all tests) and collects NACKs for the next phase. During a subsequent phase, it transmits only the lost fragments at the same given rate. It repeats phases until every receiver has received a complete file. Another welcome feature of UFTP phased delivery process is that any receiver that has received a complete file finishes the session with the sender while other receivers may continue with more phases in case of NACKs.

Scenario 25 was run in presence of worst packet loss. UFTP was the only application that reached *target* at worst packet loss rate (10^{-3}).

5.4 bbFTP suitability

Although bbFTP protocol allows sending large files over multiple parallel streams, it has the same protocol limitations as standard FTP. bbFTP reopens data connections for transfer of each subsequent file, thus it was not possible to send files of small or medium size at a high throughput. Moreover, bbFTP uses fixed TCP window size, and has several implementation restrictions on stable settings for the number of parallel streams and TCP window sizes used. The usage of fixed TCP window size may have advantages only in high packet loss scenarios. Even then, bbFTP was unable to achieve high enough throughput.

bbFTP did not reach *target* throughput in any scenario, it performed worse than even standard FTP in all scenarios except ones with packet loss ratio of 10^{-3}. Best case throughput for bbFTP – scenario 13: 36MBps (288Mbps).

Also, bbFTP was poorly implemented and crashed periodically during deployment, configuration and execution of test scenarios. If properly implemented, ensuring more stable and reliable operation, as well as allowing usage of more than ten parallel streams and optimal performance with non-default TCP buffer parameters, bbFTP could possibly be considered for use in large file transfer in high packet loss cases. But, considering the state of bbFTP at the time of this study, it was more perspective to research on the possibilities of achieving the same benefits of using fixed TCP windows during transfers with high packet loss by tuning GridFTP operation specifically for high packet loss scenarios.

bbFTP cannot be suggested based on the data gathered in this study.

5.5 GridFTP suitability

GridFTP protocol opens permanent data connection(s) that can be reused to transfer multiple files. This feature resembles protocols like RSYNC and clearly allows achieving higher throughput with small files. For files large enough (bigger than what can be sent within one TCP window), GridFTP was able to utilize parallel transfer of single file over several streams – a feature common with bbFTP. However, we could not confirm how scalable GridFTP was in capability to send multiple files simultaneously over the open parallel connections, as it was outside the scope of this project.

Due to GridFTP capability to reuse open data connections, utilize TCP window scaling provided by operating system, as well as parallel transfer of large files through several streams, GridFTP is highly suited for medium and large file transfer on WAN networks with no packet loss. But in case of packet loss on the network, GridFTP will experience dramatically decreased throughput depending on packet loss rate, amount of parallel streams used, and duration of file transfer session.
GridFTP reached *target* only in scenarios 7 and 13, which were sending 5MB and 2GB files at best conditions (70ms RTT and no packet loss). None of the other GridFTP scenarios reached *target* as a consequence of either smaller files, higher RTT, more frequent packet loss, or a combination of these.

5.6 RSYNC suitability

RSYNC protocol opens single TCP connection for the duration of whole session. This connection carries all the control commands and file data, including multiple file transfer. Upon starting the session, RSYNC application compares the given local directory with the given remote directory, calculates differences and only then starts to send actual files. This initial comparison makes the average throughput of the tests lower than the performance that can be observed during actual file transfer.

Still, single TCP connection process gives good results with all tested file sizes in case of no severe packet loss. RSYNC reached *target* in scenarios 1 and 13, when sending the smallest and the largest files (10kB and 2GB, respectively) at best conditions (70ms RTT and no packet loss). RSYNC was the only application that reached *target* with 10kB files. The other RSYNC tested scenarios (3, 15, 18) had worse conditions and did not reach *target*. This limitation on achievable throughput at high RTT or packet loss is common for all tested applications that rely on TCP window scaling provided by the operating system (FTP, GridFTP, RSYNC).

5.7 Conclusions on Applications and Protocols

Exceptional results produced within the study:

- Highest performance – FTP scenario 13: 105MBps (840Mbps);
- For small files (10kB) only RSYNC reached *target* (scenario 1: 55MBps (440Mbps));
- At worst packet loss (10^{-3}) only UFTP reached *target* (scenario 25: 45MBps (360Mbps));
- At 700ms RTT only GridFTP came close to *target* (best case – GridFTP in scenario 16: 38MBps (304Mbps)).

Only FTP application could surpass 2x *target* mark (700Mbps, 87,5MBps) in scenario 13. GridFTP and RSYNC were close to that mark in that scenario reaching 84MBps and 87MBps, respectively.

It can be concluded that four of the five applications tested have shown their best performance at some specific scenarios. Any of them may be considered for use depending on the anticipated file sizes and infrastructure or dissemination process constraints. Only bbFTP was unable to reach *target* and can be excluded from further consideration.

Several recommendations were given considering various possible assumptions about the infrastructure. Summarizing all recommendations at different assumptions the following general but not strict recommendation was made: use UFTP for multicast or if packet loss is high, otherwise use RSYNC or *tar+nc*.

The tar+nc as a very simple recommendation emerged as an afterthought after the detailed tests contracted by Eumetsat and that was described here. Tar+nc is a combination of archiving tool ‚tar' and network session tool ‚nc'. It packs together a given set of files in a single network session. We believe the performance patterns of such solution to be similar to rsync, but without the need to compare directories at session beginning. These tools are present in any mature network operating system and have evolved to be very powerful, yet simple and achieving same top throughput rates.

References

1. K. Sataki, B. Kaskina, G. Barzdins, E. Znots, M. Libins: BalticGrid-II project final report on Network Resource Provisioning, 2010. URL: http://www.balticgrid.org/Deliverables/pdfs/BGII-DSA2-9-v1-2-FinalReport-IMCSUL.pdf
2. M. Carbone, L. Rizzo: Dummynet Revisited, SIGCOMM CCR, Vol. 40, No. 2, April 2010.
3. Lee J., Cha H., Ha R.: A Two-Phase TCP Congestion Control for Reducing Bias over Heterogenous Networks, In: Proceeding of Information networking: convergence in broadband and mobile networking : international conference, ICOIN 2005, Jeju Island, Korea, January 31-February 2, 2005, LNCS Vol.3391, Springer, 2005.
4. M. Allman, V. Paxson, W. Stevens, RFC 2581: TCP Congestion Control, April 1999.
5. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. RFC 2018: TCP Selective Acknowledgment Options, October 1996.
6. J. Zhang, R. D. McLeod: A UDP-Based File Transfer Protocol (UFTP) with Flow Control using a Rough Set Approach, submitted to IEEE Transactions on Networking, 2002.

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2011. Vol. 770
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 68–89 P.

Computing Relations in the Quantum Query Model [1]

Alina Vasilieva, Taisia Mischenko-Slatenkova
Faculty of Computing, University of Latvia
Raina blvd. 29, LV-1459, Riga, Latvia
Alina.Vasiljeva@gmail.com, Taisia.Miscenko@gmail.com

Query algorithms are used to compute mathematical functions. Classical version of this model is also known as decision trees. Quantum counterpart of decision trees – quantum query model – has been actively studied in recent years. Typically, query model is used to compute Boolean functions. In this paper, we consider computing mathematical relations instead of functions. A relation is a set of ordered pairs and the difference from a function is that each element from a domain set may be mapped to multiple elements from a range set. We demonstrate that quantum query model is well suited for computing relations. We present examples of quantum query algorithms that are more efficient than the best possible classical algorithms for computing specific relations.

Keywords: quantum computing, quantum query algorithm, algorithm complexity, mathematical relation.

1 Introduction

Query model is a popular, elegant and rather simple model of computation. The goal is to compute the value of a well-known function for an arbitrary hidden input. The complexity of a query algorithm is measured by the number of questions it asks about the input variables on the worst-case input. The classical version of this model is known as *decision trees* [1].

Quantum computing is an alternative way of computation based on the laws of quantum mechanics. Quantum algorithms can solve certain problems faster than classical algorithms. The most exciting examples are Shor's [2] and Grover's algorithms [3]. This branch of computer science is developing rapidly; various computational models exist and we consider one of them. Many impressive quantum query algorithms have been developed in a query model in recent years [4-8]. An important task in complexity theory is to find examples with a large gap between classical and quantum algorithm complexity of the same computational problem.

[1] This work has been supported by the European Social Fund within the project „Support for Doctoral Studies at University of Latvia".

Most often query model is used to compute Boolean functions. However, it is possible to apply query model to functions with larger domain and range as well. In this paper, we consider even more uncommon case – computing *mathematical relations* instead of functions. A binary relation is more general type of problem than a function. A relation is a set of ordered pairs that associates values from a domain set with values from a range set. Difference from a function is in element mapping: each element from a domain set may be mapped to multiple elements from a range set. So, a function is simply a special case of a relation, where each value from a domain set is mapped to no more than one value from a range set. Alternative way is to consider relations as multi-valued functions.

The study of query complexity of relations has been inspired by the book on communication complexity by Kushilevitz and Nisan [9]. The main part of this book discusses communication complexity of functions, but Chapter 5 is devoted exactly to the communication complexity of relations.

We apply traditional query model to compute relations. In classical deterministic settings, however, it does not seem to be possible to employ the difference between a relation and a function to obtain new interesting results. A deterministic decision tree always follows one and the same fixed path for each certain input and outputs one and the same value each time. The situation is different in the quantum case. Quantum state before the measurement is in a superposition of the basis states, so it is not determined to which exactly basis state quantum system collapses after the measurement.

Various computational problems may be represented in terms of relations. Let us consider, for instance, an online reservation system for a large renting company. Company provides various products for rent, for example, cars, flats, TV-sets etc. User fills in a reservation form on the Web page and submits it. According to user's request parameters (relation input) system has to find a set of satisfying and available items (value set for that input) and display them to user for further selection or even perform selection automatically. By designing an efficient algorithm for computing this kind of relation we are able to speed-up processing significantly. Nowadays, in heavy-loaded systems with huge amount of concurrent requests, a lot of resources could be saved by performance improvement at the moment of selecting appropriate value set.

Significant difficulty in designing quantum query algorithm is making it exact (i.e. make it output correct result always with probability $p = 1$). The largest complexity separation between classical deterministic and quantum exact query algorithm complexity for the same total function known for today is N versus $N/2$. However, in the case of relation, we are allowed to output values from a fixed set instead of one fixed value for a certain input. We assert that in such case the task of designing a non-trivial exact quantum query algorithm is achievable more easily. That could help to construct examples, where number of queries required by quantum algorithm is more than two times less than required by classical algorithm.

In this paper, we adapt the query model for computing relations. First, we give the definitions related to mathematical relations. We define several types of query algorithms that may compute relations in different manners. Then we demonstrate examples of computing relations in classical and quantum query models, where quantum algorithm achieves a speed-up comparing to classical algorithm. Finally, we discuss the prospects of achieving good results in enlarging the complexity gap between classical and quantum query complexity for relations.

2 Preliminaries

This section contains definitions and provides theoretical background on the subject. First, we define classical decision tree. Next, we provide a brief overview of the basics of quantum computing. Finally, we describe the quantum query model.

2.1 Classical Decision Trees

The classical version of the query model is known as *decision trees* [1]. Definition of Boolean function is known to everybody, but the input $X = (x_a, x_2, ..., x_n)$ is hidden in a black box, and can be accessed by querying x_i values. Algorithm must be able to determine value of the function correctly for arbitrary input. Complexity of the algorithm is measured by number of queries on the worst-case input. For more details, see the survey by Buhrman and de Wolf [1].

Deterministic decision tree is a tree with internal nodes labeled with variables x_i, arrows exiting internal nodes labeled with possible variable values and leafs labeled with function values. Deterministic decision tree always follows the same path for each input and produces the correct result with probability $p = 1$. Deterministic complexity of a function f is denoted by $D(f)$.

Probabilistic (randomized) decision tree may contain internal nodes with probabilistic branching, i.e., multiple arrows exiting from the same node, each one labeled with a probability for algorithm to follow that way. The total probability to obtain the result r after execution of an algorithm on certain input X equals to the sum of probabilities for each leaf labeled with r to be reached. Total probability of an algorithm to produce the correct result is the probability on the worst-case input.

2.2 Quantum Computing

We briefly outline basic notions of quantum computing here that are necessary to define the quantum query model. For more details, see [5, 6, 10].

An n-dimensional quantum pure state is a unit vector in a Hilbert space. Let $|0\rangle,|1\rangle,.,|n\text{-}1\rangle$ be an orthonormal basis for $\mathbb{C}^n$. Then, any state can be expressed as $|\psi\rangle=\sum_{i=0}^{n-1} a_i |i\rangle$ for some $a_i \in \mathbb{C}$. The norm of $|\psi\rangle$ is 1, so we have $\sum_{i=0}^{n-1} |a_i|^2 = 1$. States $|0\rangle,|1\rangle,\ldots,|n\text{-}1\rangle$ are called *basis states.* Any state of the form $\sum_{i=0}^{n-1} |a_i|^2 = 1$ is called a *superposition* of $|0\rangle,\ldots,|n\text{-}1\rangle$. The coefficient a_i is called the *amplitude* of $|i\rangle$.

The state of a system can be changed by applying *unitary transformation.* Unitary transformation U is a linear transformation on $\mathbb{C}^n$ that maps each vector of unit norm to a vector of unit norm. The *transpose* of an $m \times n$ matrix A is the $n \times m$ matrix $A_{ij}^T = A_{ji}$ for $1 \le i \le n$, $1 \le j \le m$. H denotes the Hadamard gate.

We use the simplest case of quantum measurement: the full measurement in the computational basis. Performing this measurement on state $|\psi\rangle=a_0|0\rangle+\ldots a_{n\text{-}1}|n\text{-}1\rangle$ produces outcome i with probability $|a_i|^2$. Measurement changes the state of the system to $|i\rangle$ and destroys the original state.

2.3 Quantum Query Model

The quantum query model is the quantum counterpart of the decision tree model and is intended for computing Boolean functions. For a detailed description, see [4-6].

A quantum computation with T queries is a sequence of unitary transformations:

$$U_0 , Q_0 , U_1 , Q_1 , ... , U_{T-1} , Q_{T-1} , U_T$$

U_i's can be arbitrary unitary transformations that do not depend on input bits. Q_i's are query transformations. Computation starts in the initial state $|\vec{0}\rangle$. Then we apply U_0, Q_0, ..., Q_{T-1}, U_T and measure the final state.

We use the following definition of a query transformation: if the input is a state $|\psi\rangle = \sum_i a_i |i\rangle$, then the output is:

$$|\gamma_i\rangle = \sum_i (-1)^{\varphi_i} a_i |i\rangle, \text{ where } \varphi_i \in \{x_1, ..., x_N, 0,1\}.$$

For each query, we may arbitrarily choose a variable assignment φ_i for each basis state. If the value of the assigned variable $\varphi_i \in \{x_1, ..., x_N\}$ is "1", then the sign of the i-th amplitude a_i changes to the opposite.

Each quantum basis state corresponds to an algorithm's output. We assign a value of the function to each output. The probability of obtaining the result j after executing the algorithm on input X equals to the sum of squared modulus of all amplitudes that correspond to outputs with value j.

Definition 1 [1]. *A quantum query algorithm* computes *f* exactly *if the output equals f(x) with probability p = 1, for all* $x \in \{0,1\}^N$. *Complexity is* $Q_E(f)$.

3 Mathematical Relations

The main object which is studied in this paper is mathematical relations.

Definition 2 [11]. *A* relation R *from a set A to a set B is a subset of Cartesian product A × B – a collection of ordered pairs (a, b) with first components from the set A (domain) and second components from the set B (range).*

In other words, relation associates each value from the domain set with a subset of values from the range set. We call each value from the domain set – an input $X = (x_1, x_2, \ldots, x_N)$. We call each x_i – a variable. We call a set of associated values from the range set – a result set for input X and denote it by $R(X)$. We consider *left-total* relations only, when the result set is not empty for each domain set element. Relation can actually be considered a multi-valued function.

A function is a special case of relation and it uniquely associates each value from the domain set with one value from the range set. Fig. 1 graphically demonstrates this difference.

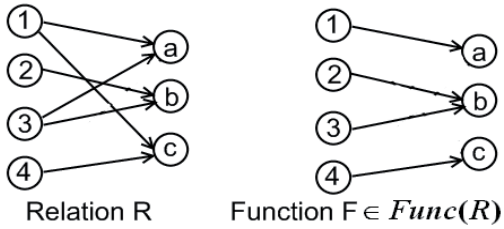


Fig. 1. Example of a relation and a function

Various functions can be selected in such a way from a single relation. We denote by *Func*(*R*) the set of all total functions that can be selected from relation *R*.

Example. The graph on the left side of Fig. 1 defines the relation:

$$R = \{ (1,a),(1,c),(2,b),(3,a),(3,b),(4,c) \}.$$

The set *Func*(*R*) consists of four total functions that may be selected as a subset of the relation:

$$Func(R) = \{ f_1 = \{ (1,a),(2,b),(3,a),(4,c) \}, f_2 = \{ (1,a),(2,b),(3,b),(4,c) \},$$
$$f_3 = \{ (1,c),(2,b),(3,a),(4,c) \}, f_4 = \{ (1,c),(2,b),(3,b),(4,c) \}\}.$$

4 Computing Relations in a Query Model

It is well known how to compute functions in a query model. Algorithm simply has to output the function value with certain probability. But what does it mean to compute a relation in a query model? We propose three different options to describe that a query algorithm computes a relation and define three types of query algorithms based on these options.

Definition 3. *Query algorithm computes relation R in a* definite manner*, if for each X it outputs one certain correct value from a result set with probability p = 1. Classical query complexity is denoted by* $C_D(R)$*. Quantum query complexity is denoted by* $Q_D(R)$*.*

The type of classical decision tree that computes a relation in a definite manner is deterministic decision tree. In the quantum version, corresponding algorithm type is an exact quantum query algorithm.

Definition 4. *Query algorithm computes relation R in a* randomly distributed manner*, if for each X it outputs arbitrary values from a result set with arbitrary probabilities (for each value, such probability has to be positive) and never outputs an incorrect value. Classical query complexity is denoted by* $C_{RD}(R)$*. Quantum query complexity is denoted by* $Q_{RD}(R)$*.*

This definition is more natural and takes into account the essence of relation as a mathematical object. In a classical query model, probabilistic decision trees should be used to produce the described behavior. Quantum query algorithms seem to be better suited for computing relations in a distributed manner because of the superposition principle. To achieve the goal, we need to bring quantum system in such a superposition, where only basis states associated with values from the result set have non-zero amplitude values. After the measurement, quantum system collapses to one of these basis states with a probability determined by its amplitude value.

Definition 5. *Query algorithm computes relation R in uniformly* distributed manner*, if for each X it outputs each value from a result set with equal probability and never outputs an incorrect value. Classical query complexity is denoted by* $C_{UD}(R)$*. Quantum query complexity is denoted by* $Q_{UD}(R)$*.*

This definition adds a serious constraint to design of a query algorithm. However, in our opinion, this definition is the most reasonable in a sense of computing a relation.

Each definition may be applied for solving specific real-world computational problems. We are most interested in comparing complexity of computing relations in the same manners in classical and quantum query models. Our goal is to analyze special features and differences of algorithm implementation to produce examples with large difference between classical and quantum query complexity.

5 Examples of Computing Relations

In this section, we present examples of computing relations in both classical and quantum query models. In all our examples, we achieve a speed-up in quantum algorithm complexity comparing to the best possible classical analogue.

5.1 First Example of Computing a Relation

Let us consider an online banking client service system. To receive specific kind of bank's services, client sends a request to the system. System has to analyze client's request, determine a set of appropriate agents and assign a request to some agent from this set.

In our example, we assume four agents: Alice (id = 1), Bob (id = 2), Carol (id = 3) and Daren (id = 4). There are three factors that determine a set of appropriate agents for each client – location, client status and loan history.

Table 1 describes these parameters. Second column contains a reference to the system function that has to be invoked to calculate parameter value. Invocation of each function can be interpreted as querying a black box and internal calculations may involve various database requests and other costly operations. Third column contains possible parameter values; fourth column contains corresponding numeric value returned by each function.

Table 2 defines the three-variable relation with Boolean domain and four-valued range - $\Re 1 : \{0,1\}^3 \to \{1, 2, 3, 4\}$.

Table 1

Parameters that determine an agent that is able to serve client's request

Parameter		Value	
Description	**System function**	**Actual**	**Numeric**
Client location	`getLocation(client_id)`	Saldus	0
		Ventspils	1
Client status	`isVIP(client_id)`	Normal	0
		VIP	1
Does client have an active loan?	`hasLoan(client_id)`	No	0
		Yes	1

Rows of Table 2 have to be interpreted as the following statements:

- If a request is received from an ordinary client from Saldus, which does not have an active loan ($X = 000$), then a request should be served by either Alice or Carol;
- If a request is received from an ordinary client from Saldus, which has an active loan ($X = 001$), then a request should be served by either Alice or Daren;
- If a request is received from a VIP client from Saldus, which does not have an active loan ($X = 010$), then a request should be served by either Bob or Daren;
- etc.

Table 2

Definition of the relation $\Re 1$

X	$\Re 1$ (X)	X	$\Re 1$ (X)
000	{ 1 , 3 }	100	{ 2 , 4 }
001	{ 1 , 4 }	101	{ 2 , 3 }
010	{ 2 , 3 }	110	{ 1 , 4 }
011	{ 2 , 4 }	111	{ 1 , 3 }

Now, let us discuss the computational complexity of relation $\Re 1$.

5.2 Definite Query Complexity of Relation $\Re 1$

When computing a relation in definite manner, algorithm has to output one certain correct value from a result set with probability $p = 1$. It means that we are just aware of that client's request is not forwarded to incompetent agent, but we do not care about the work distribution among the competent agents.

Theorem 1. $C_D(\Re 1) = 2$.

Proof. It is easy to see that one query is not enough to compute this relation classically in a definite manner. However, two queries are sufficient to reach the goal – we only need to know the values of the first two variables. Deterministic decision tree is shown in Fig. 2.

Actually, what we need is to compute *XOR* of the first two bits. If $XOR(x_1, x_2) = 0$, algorithm outputs "1". Otherwise, algorithm outputs "2". □

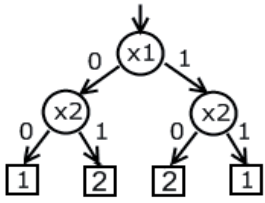


Fig. 2. Deterministic decision tree that computes $\Re 1$ in a definite manner

□

Theorem 2. $Q_D(\Re 1) = 1$.

Proof. It is well known that *XOR* of two bits can be computed exactly in the quantum query model by asking one query. It immediately implies that relation $\Re 1$ can

be computed with one query in a quantum query model in a definite manner. Quantum algorithm is shown in Fig. 3 and described below.

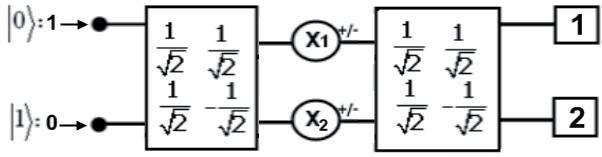


Fig. 3. Quantum query algorithm that computes $\Re 1$ in a definite manner

Algorithm uses one-qubit quantum system. Each horizontal line corresponds to the amplitudes of the basis states $|0\rangle$ and $|1\rangle$. Large rectangles correspond to the 2×2 Hadamard matrices. Single query Q_0 is defined by the unitary matrix:

$$Q_0 = \begin{pmatrix} (-1)^{x_1} & 0 \\ 0 & (-1)^{x_2} \end{pmatrix}$$

Query matrix specifies how the signs of amplitudes of basis states change depending on variable values. Measurement is performed after the last unitary transformation. Finally, two small squares at the end of each horizontal line define the output value for each basis state. □

The problem with such implementation of work distribution algorithm is that all requests will be forwarded to Alice and Bob only, but Carol and Daren will be bored without work.

5.3 Uniformly Distributed Query Complexity of Relation $\Re 1$

Now, let us consider computing $\Re 1$ in uniformly distributed manner, which seems to be much more practical. This time algorithm has to output each value from the result set with equal probability and should never output incorrect value.

Obviously, one query is not enough in the classical case. However, this time again, two queries suffice.

Theorem 3. $C_{UD}(\Re 1) = 2$.

Proof. Classical probabilistic decision tree that computes $\Re 1$ in uniformly distributed manner is shown in Fig. 4. □

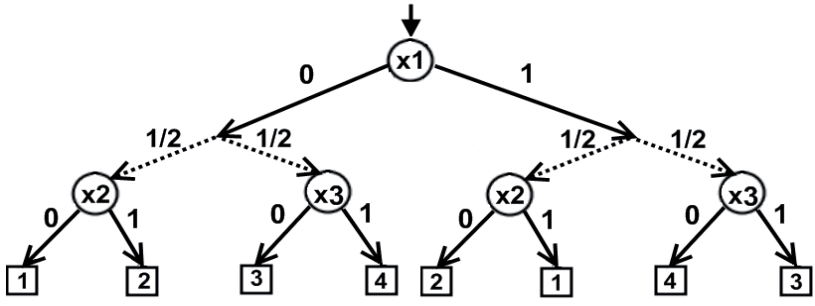


Fig. 4. Probabilistic decision tree that computes $\Re 1$ in uniformly distributed manner

Theorem 4. $Q_{UD}(\Re 1) = 1$.

Proof. Quantum query algorithm $Q1$, which computes $\Re 1$ in the same uniformly distributed manner with one query, is presented in Fig. 5 and described below.

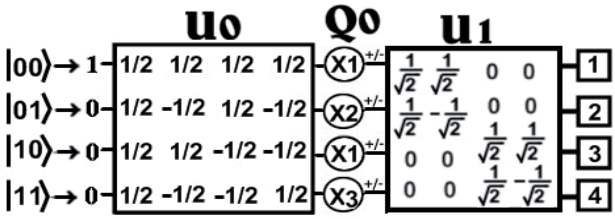


Fig. 5. Quantum query algorithm $Q1$ for computing $\Re 1$ in uniformly distributed manner

Algorithm $Q1$ uses two-qubit quantum system. Each horizontal line corresponds to the amplitude of the basis state. Large rectangles correspond to the 4 × 4 unitary matrices. Four small squares at the end of each horizontal line define the output value for each basis state.

Single query Q_0 is defined by the unitary matrix:

$$Q_0 = \begin{pmatrix} (-1)^{x_1} & 0 & 0 & 0 \\ 0 & (-1)^{x_2} & 0 & 0 \\ 0 & 0 & (-1)^{x_1} & 0 \\ 0 & 0 & 0 & (-1)^{x_3} \end{pmatrix}$$

Computational process for each input X is shown in Table 3. □

Table 3

Computation process of the quantum query algorithm $Q1$

X	State after the query	State before the measurement	Output
000	$\left(\frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("1")=1/2 Pr("3")=1/2
001	$\left(\frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("1")=1/2 Pr("4")=1/2
010	$\left(\frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("2")=1/2 Pr("3")=1/2
011	$\left(\frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("2")=1/2 Pr("4")=1/2
100	$\left(-\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("2")=1/2 Pr("4")=1/2
101	$\left(-\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("2")=1/2 Pr("3")=1/2
110	$\left(-\frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("1")=1/2 Pr("4")=1/2
111	$\left(-\frac{1}{2}\ -\frac{1}{2}\ -\frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("1")=1/2 Pr("3")=1/2

This time all work items are equally distributed among agents.

With this basic example we have demonstrated query algorithms for computing relations in action. We have shown that even in such a simple case of relation with three Boolean variables it is possible to obtain a gap between classical and quantum query complexity. In the next subsection, we demonstrate how to enlarge the complexity gap in uniformly distributed case.

5.4 Generalizations of the Relation $\Re 1$

In this subsection, we demonstrate two extensions of the relation $\Re 1$ with a bigger number of variables and more impressive complexity separation between classical and quantum algorithms.

Definition 6. *Relation* $\Re 2 : \{0,1\}^N \rightarrow \{1, 2, ..., 2(N–1)\}$ *associates each input element from the domain set with (N-1) output elements from the range set according to the following rule:*

$$\forall\ 1 < i \leq N : \text{if } (x_1 \oplus x_i = 0), \quad \text{then } (X, 2(i–1)–1) \in \Re 2$$
$$\text{otherwise } (X, 2(i–1)) \in \Re 2$$

It turns out that it is possible to compute relation $\Re 2$ classically in uniformly distributed manner using two queries.

Theorem 5. $C_{UD}(\Re 2) = 2$.

Proof. Classical probabilistic decision tree is demonstrated in Fig. 6. □

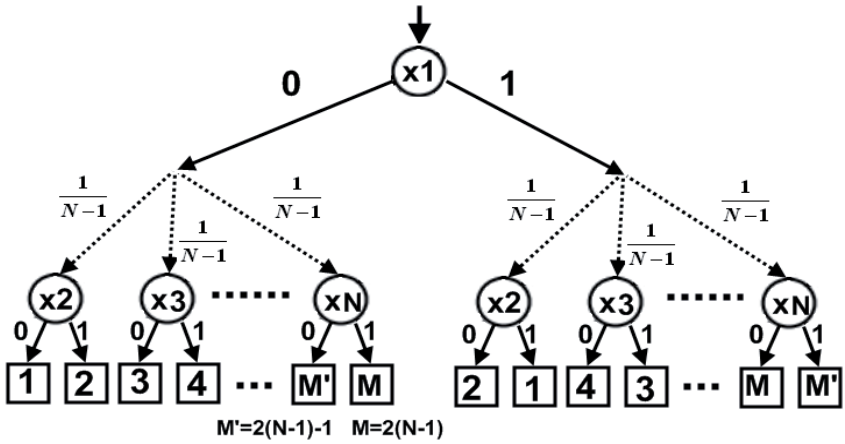


Fig. 6. Classical query algorithm for computing $\Re 2$ in uniformly distributed manner

Theorem 6. $Q_{UD}(\Re 2) = 1$.

Proof. To compute relation $\Re 2$ in a quantum query settings, we extend algorithm $Q1$ to query all N relation variables in a single query. To be able to handle all variables,

we extend quantum system to have $2(N–1)$ basis states. Fig. 7 shows quantum algorithm $Q2$, which is an extended version of the algorithm $Q1$. H is the 2×2 Hadamard transformation, $\oplus$ denotes matrix tensor product operation. Quantum system consists of A qubits, where $A = \lceil \log_2(2(N-1)) \rceil$.

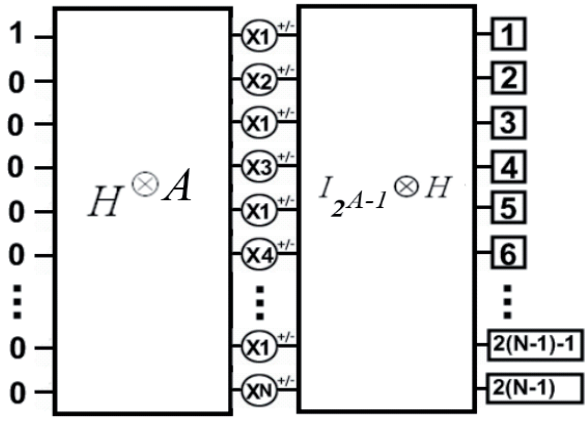


Fig. 7. Quantum query algorithm $Q2$ for computing $\Re 2$ in uniformly distributed manner

Important moment is that variable x_1 is assigned to all odd amplitudes, but remaining variables x_2, ..., x_N are sequentially assigned to even amplitudes. □

In this example, we enlarged the number of relation variables, but did not succeeded yet in enlarging the gap between classical and quantum query complexity.

Next, we demonstrate another generalization of the relation $\Re 1$. This time we achieve a gap $2N$ versus N between classical and quantum query complexity.

Definition and behavior of relation $\Re 3$ is similar to relation $\Re 2$ – it associates each input element with $(N\text{-}1)$ output elements from the range set. But this time more variables are involved in the condition of the rule, which defines the relation.

Definition 7. *Relation* $\Re 3 : \{0,1\}^{N^2} \rightarrow \{1, 2, ..., 2(N-1)\}$ *associates each input element from the domain set with (N-1) output elements from the range set according to the following rule:*

$$\forall\ 1 < i \le N : \text{if } ((x_1 \oplus x_2 \oplus ... \oplus x_N) \oplus (x_{(i-1)N+1} \oplus x_{(i-1)N+2} \oplus ... \oplus x_{(i-1)N+N}) = 0)$$
$$\text{then } (X, 2(i-1)-1) \in \Re 3$$
$$\text{otherwise } (X, 2(i-1)) \in \Re 3$$

To compute relation $\Re 3$ in a classical query model, $2N$ queries are required.

Theorem 7. $C_{UD}(\Re 3) = 2N$.

Proof. In order to determine which range set element to include into the result set, it is necessary to know values of all $2N$ variables involved into condition of the rule. A part of classical decision tree is depicted in Fig. 8. All sequentially queried variables are joined into one common query represented in the diagram by ellipses. Multiple arrows corresponding to common query outcomes are exiting these ellipses. □

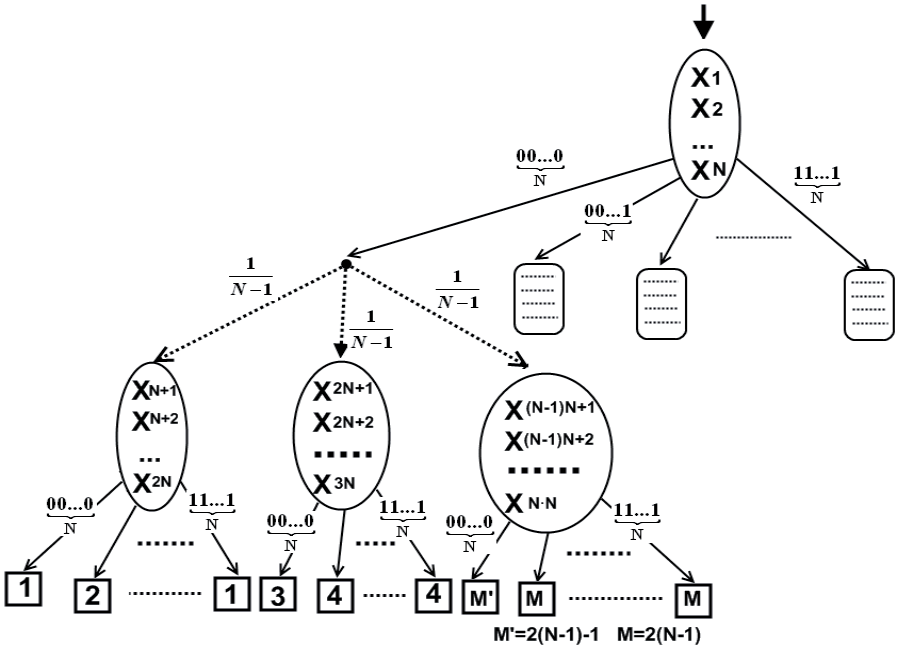


Fig. 8. Classical query algorithm for computing $\Re 3$ in uniformly distributed manner

Theorem 8. $Q_{UD}(\Re 3) = N$.

Proof. General structure of the algorithm remains the same, but we add more queries. Algorithm $Q3$ is presented in Fig. 9. Again, odd amplitudes all have the same set $(x_1, ..., x_N)$ of queried variables assigned. Remaining variables are sequentially assigned to even amplitudes. □

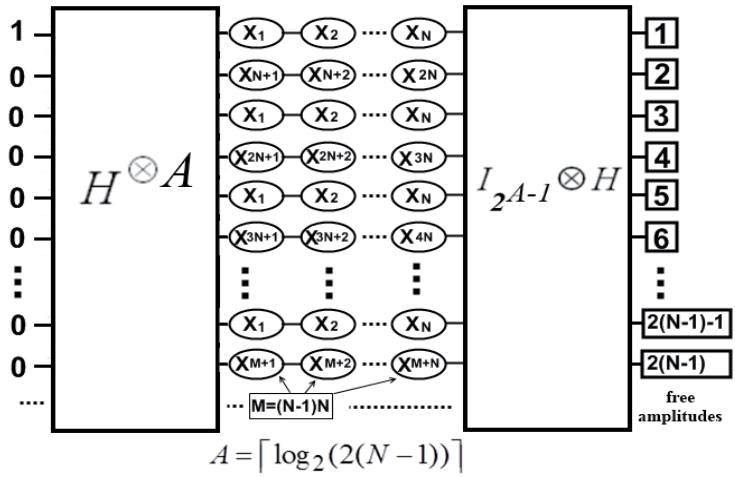


Fig. 9. Quantum query algorithm $Q3$ for computing $\Re 3$ in uniformly distributed manner

In this subsection, we demonstrated approach for extending relations to a larger number of variables. As a result we obtained a complexity separation N versus $2N$, which is the same as the largest separation between quantum exact and classical deterministic query algorithm for total functions known for today. During computing relations correct result is obtained with probability $p = 1$ as well (algorithm always outputs some correct value from the result set). However, the structure of considered relations is based on *XOR* operation. All examples of N versus $2N$ separations for functions, that we are aware of, are directly based on *XOR* as well. We are interested to find different cases, where *XOR* is not involved in obtaining a speed-up.

5.5 Second Example of Computing a Relation

Let us consider some TV company that offers minimal package and four more supplementary packages: movies, sports, social talk-shows and cartoons. Every client is free to choose any number of supplementary packages he is interested in. Company is willing to make a present for each client according to client's choice of packages. There are four different types of gift, let us mark them "1", "2", "3", "4".

Rule 1. If a client has one or three packages besides minimal package, company has to choose one from "1", "2", "3", "4" (probability to choose any gift from the scope has to be equally distributed between options, each having $p = ¼$ to be selected).

Rule 2. If a client has only the minimal package or all four supplementary packages, company presents a gift of type "1".

Rule 3. If a client has chosen movies and social talk-shows or sports and cartoons, company presents a gift of type "2".

Rule 4. If a client has chosen movies and sports or social talk-shows and cartoons, company presents a gift of type "3".

Rule 5. If a client has chosen movies and cartoons or social talk-shows and sports, company presents a gift of type "4".

Table 4 defines relation with Boolean domain and four-valued range: $\Re 4 : \{0,1\}^4 \rightarrow \{1, 2, 3, 4\}$. Let us assign an index to each type of packages: 1 for movies, 2 for sports, 3 for social talk-shows and 4 for cartoons. Each bit in the input string X gives the information whether i-th package is chosen by the client. 0000 means that only the minimal package is chosen, 1111 – full and so on.

Table 4

Definition of the relation $\Re 4$

X	$\Re 4$ $\{X\}$	X	$\Re 4$ $\{X\}$
0000	{1}	1000	{1,2,3,4}
0001	{1,2,3,4}	1001	{4}
0010	{1,2,3,4}	1010	{2}
0011	{3}	1011	{1,2,3,4}
0100	{1,2,3,4}	1100	{3}
0101	{2}	1101	{1,2,3,4}
0110	{4}	1110	{1,2,3,4}
0111	{1,2,3,4}	1111	{1}

5.6 Uniformly Distributed Query Complexity of Relation $\Re 4$

Now, let us discuss the complexity of relation $\Re 4$. We consider computing $\Re 4$ again in the same uniformly distributed manner.

Theorem 9. $C_{UD}(\Re 4) = 3$.

Proof. Proof of this fact consists of two steps. First, we show that it is not possible to build a classical randomized decision tree of depth $d = 2$, which computes $\Re 4$ in uniformly distributed manner. Second, we present a tree, which computes $\Re 4$ using three queries.

Lemma 1. *It is not possible to build a classical randomized decision tree of depth $d = 2$, which computes $\Re 4$ in uniformly distributed manner.*

Proof. Let us assume there exists a tree where all paths from root to leaves contain no more than two variables. When executing algorithm on input $X = 0000$ result "1" should be output with probability $p = 1$. It means that there exists a path from root to leaf with value "1", which goes through some two variables: $x_A = 0$ and $x_B = 0$. This path is depicted in Fig. 10. The fact is that it is not possible to select A and B to avoid contradictions with other inputs. Table 5 shows all possible selections of A and B, together with such input Y, which has the same values in positions A and B as $X = 0000$. For these inputs, algorithm goes the same path as for $X = 0000$ and finishes in a leaf with value "1", which is incorrect for Y, thus causing a contradiction with a correct output value for Y. So, it is not possible to build a classical randomized decision tree of depth $d = 2$, which computes $\Re 4$ in uniformly distributed manner. □

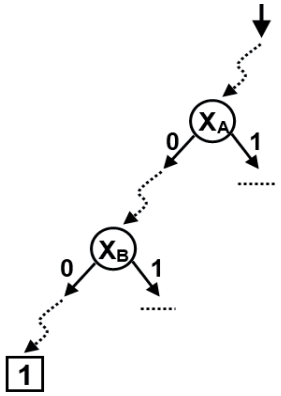


Fig. 10. Path for input X=0000 in a potential classical randomized decision tree of depth $d = 2$ for computing $\Re 4$ in uniformly distributed manner

Table 5

All possible selections of A and B, each causing a contradiction

A	B	Input *Y,* for which algorithm goes through the same path (Fig. 10), which contradicts with X=0000 in output value	$\Re 4$ (X)
1	2	0011	{3}
1	3	0101	{2}
1	4	0110	{4}
2	3	1001	{4}
2	4	1010	{2}
3	4	1100	{3}

Lemma 2. *There exists a classical randomized decision tree, which computes* $\Re 4$ *in uniformly distributed manner using three queries.*

Proof. Classical probabilistic decision tree that computes $\Re 4$ in uniformly distributed manner is shown in Fig. 11.

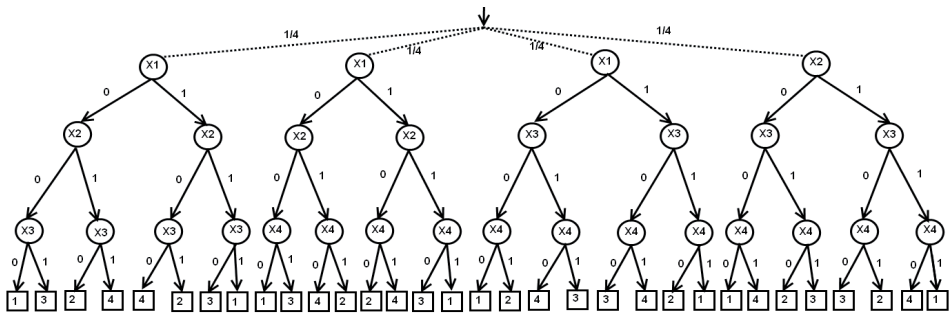


Fig. 11. Probabilistic decision tree that computes $\Re 4$ in uniformly distributed manner

Theorem 10. $Q_{UD}(\Re 4) = 1$.

Proof. Quantum query algorithm $Q4$, which computes $\Re 4$ in the same uniformly distributed manner with one query, is presented in Fig. 12. □

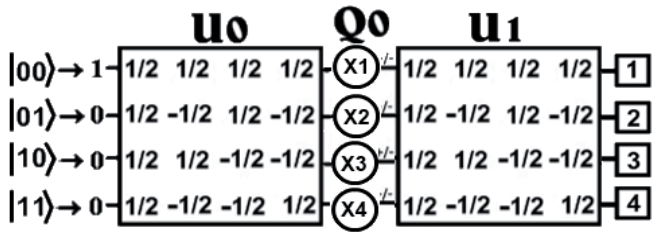


Fig. 12. Quantum query algorithm $Q4$ for computing $\Re 4$ in uniformly distributed manner

We would like to note that definition of the relation $\Re 4$ and algorithm $Q4$ in some sense look similar to the definition and solution of the well-known Deutsch-Jozsa problem [12,13]. Careful reader could figure out this similarity by oneself. However, as we demonstrate further, generalization of that relation is not of that kind anymore.

5.7 First generalization of the relation $\Re 4$

Let us define the relation $\Re_{4N}: \{0,1\}^{4N} \to \{1, 2, 3, 4\}$. Imagine that $4N$ variables are put on four vertical lines (v-lines) in such a way that:

$\forall i \in \{0, ...N - 1\}, \forall k \in \{1,2,3,4\} : x_{4i+k}$, belongs to v-line number k.

For example, x_1, x_5, x_9, x_{13}, ... are placed on the 1st v-line, x_2, x_6, x_{10}, x_{14}, ... – on the 2nd, and so on (see Fig. 13 for illustration).

The result set for each input X of the relation is defined as follows:

1. $\Re_{4N}(X) = \{1\}$, if all four v-lines of X contains either odd or even number of ”1”s. For example, for the next input strings, the relation’s result set is $\{1\}$:
   – input string 00000000 has zero ”1”s on each v-line
   – input 00010001 has zero ”1”s on the first, second and third v-line and two ”1”s on the fourth v- line
   – input 00001111 has one ”1” on each v-line
   – input 11111111 has exactly two ”1”s on each v-line
2. $\Re_{4N}(X) = \{2\}$, if 1st and 3rd v-lines of X have odd number of ”1”s and 2nd and 4th have even number of ”1”s, or vice versa: 1st an 3rd – even and 2nd and 4th – odd. For example, input strings 00000101, 00001010, 01011111, 11011000 have the result set $\{2\}$.
3. $\Re_{4N}(X) = \{3\}$, if 1st and 2nd v-lines of X have odd number of ”1”s and 3rd and 4th have even number of ”1”s, or vice versa: 1st and 2nd – even and 3rd and 4th – odd. For example, input strings 00000011, 00001100, 00111111, 10001011 have the result set $\{3\}$.
4. $\Re_{4N}(X) = \{4\}$, if 1st and 4th v-lines of X have odd number of ”1”s and 2nd and

3rd have even number of ”1”s, or vice versa: 1st and 4th – even and 2nd and 3rd – odd. For example, input strings 00000110, 00001001, 00111010, 10011111 have the result set {4}.

5. In all other cases, $\Re_{4N}(X) = \{1,2,3,4\}$.

Theorem 11. $Q_{UD}(\Re_{4N}) = N$.

Proof. Quantum algorithm that computes relation in the uniformly distributed manner is presented in Fig. 13. Each quantum query Q_i is defined by the following unitary matrix: □

$$Q_i = \begin{pmatrix} (-1)^{x_{4i+1}} & 0 & 0 & 0 \\ 0 & (-1)^{x_{4i+2}} & 0 & 0 \\ 0 & 0 & (-1)^{x_{4i+3}} & 0 \\ 0 & 0 & 0 & (-1)^{x_{4i+4}} \end{pmatrix}, \; i \in \{0,\ldots,N-1\}.$$

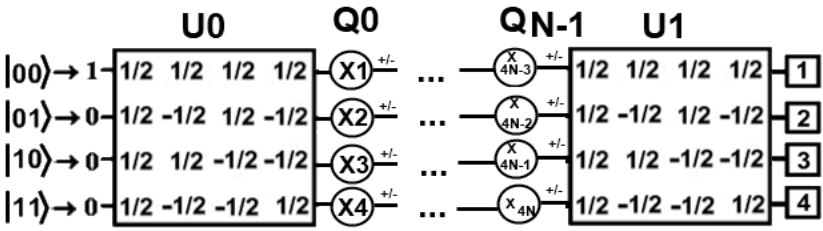


Fig. 13. Quantum query algorithm for computing $\Re_{4N}$ in uniformly distributed manner

Theorem 12. $Q_{UD}(\Re_{4N}) \geq 3N$.

Proof. Let us assume there exists a classical decision tree that computes relation $\Re_{4N}$ by asking $3N-1$ questions. We use all zeros input $X = \vec{0}$ to demonstrate a contradiction. Suppose we queried arbitrary $3N-1$ variables, $N+1$ variables remain unquestioned.

On $4N$-zeros input $X = \vec{0}$ algorithm has to output value ”1” because all v-lines contain zero number of ”1”s. Then, we consider only such inputs that have ”0” in all queried $3N-1$ variables and exactly two ”1”s among remaining unquestioned variables. For all such inputs, algorithm will follow the same path and will finish in the same leaves with output value ”1”.

However, all $N+1$ unquestioned variables cannot be located on one v-line, simply because each v-line consists of N variables. So, there is an input for which two ”1”s among unquestioned variables are located on different v-lines. As we know, the result set in such case is {2} or {3} or {4}. Thus, algorithm outputs incorrect value for this input, this fact contradicts with the initial assumption and implies $Q_{UD}(\Re_{4N}) \geq 3N$. □

5.8 Second generalization of the relation ℜ4

Suppose we are given a relation of N variables $\Re_N : \{0,1\}^N \to \{1,2,...,N\}$, where N is power of 2. This time, we do not provide full definition of the relation; it follows from properties of quantum algorithm described below. We would only like to demonstrate that such generalization is technically possible.

This time, we consider computing relation in a randomly distributed manner. Algorithm is allowed to output any value from the result set with arbitrary probability, but probability for each value has to be positive: $p > 0$.

Theorem 13. *There is a quantum query algorithm computing specific relation* $\Re_N$ *in a randomly distributed manner asking one question only:* $Q_{RD}(\Re_N) = 1$.

Proof. We add more qubits and sequentially assign variables to amplitudes. Given $N = 2^k, k \in \mathbb{N}$, quantum algorithm starts with k-qubit zero state $|0\rangle$, then applies $N \times N$ Hadamard matrix, N-variable query and finally applies $N \times N$ Hadamard matrix once again. Algorithm is depicted in Fig. 14. □

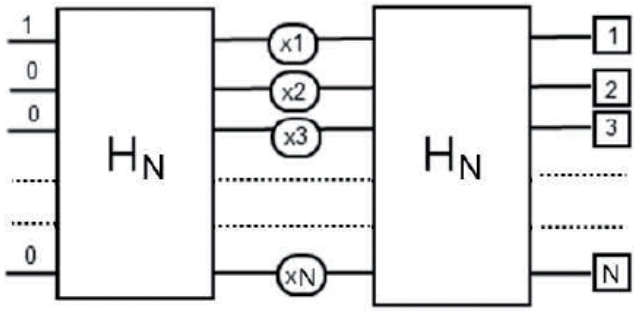


Fig. 14. Generalization of the quantum query algorithm for computing $\mathbf{R_N}$

Theorem 14. $\frac{N}{2} + 1 \le C_{RD}(\Re_N) \le N$.

Proof. Let us analyze the relation that is computable by the extended quantum algorithm. Imagine the first element of the quantum algorithm result vector (amplitude of the quantum basis state $|0\rangle$) right before the quantum measurement. It can be described by the formula:

$$\alpha_1 = \frac{(-1)^{x1} + (-1)^{x2} + ... + (-1)^{xN}}{N}.$$

If all $x_i = 0$, then $\alpha_1 = 1$, so for the input $X = 00\ldots0$, algorithm outputs ”1” with probability $p = 1$. Let us suppose exactly $N/2$ variables are ”1”s and $\frac{N}{2}$ are ”0”s. In this case, α_1 is precisely zero for all possible combinations. It means that probability to observe result value ”1” for any such input is $p = 0$.

Classical algorithm has to behave in the same way: for input $X = 00\ldots0$, value "1" has to be produced with probability $p = 1$, but for all inputs with exactly $N/2$ "1"s, result value "1" is not allowed to be output at all. This implies we are unable to recognize relation classically by asking only $N/2$ variable values, at least $N/2+1$ queries are required. □

5.9 Third Example of Computing a Relation

In this section, we demonstrate our last example of computing a relation. We present a quantum query algorithm that computes the relation asking two queries in uniformly distributed manner; while classically at least five queries are necessary to compute the same relation.

Important fact is that the structure of a relation and the algorithm for computing it are not based on *XOR* operation. In the area of quantum query algorithms for computing total functions, all examples, that we are aware of at the moment, where quantum query complexity is two times less than classical query complexity are directly based on utilization of *XOR* operation.

Another important moment is that in this example the result set for each input consists of two elements and there is no input for which the result set consists of all possible output values.

Relation $\Re 5 : \{0,1\}^6 \rightarrow \{1,2,3,4\}$ is defined by the following set of rules:

- if $x_1 = x_2$ & $x_3 = 0$ & $x_1 = x_5$, then $\Re 5(X) = \{1,2\}$;
- if $x_1 = x_2$ & $x_3 = 0$ & $x_1 \neq x_5$, then $\Re 5(X) = \{3,4\}$;
- if $x_1 = x_2$ & $x_3 = 1$ & $x_1 = x_6$, then $\Re 5(X) = \{2,3\}$;
- if $x_1 = x_2$ & $x_3 = 1$ & $x_1 \neq x_6$, then $\Re 5(X) = \{1,4\}$;
- if $x_1 \neq x_2$ & $x_3 = 0$ & $x_1 = x_4$ & $x_5 = 0$, then $\Re 5(X) = \{1,4\}$;
- if $x_1 \neq x_2$ & $x_3 = 0$ & $x_1 = x_4$ & $x_5 = 1$, then $\Re 5(X) = \{2,3\}$;
- if $x_1 \neq x_2$ & $x_3 = 0$ & $x_1 = x_4$ & $x_5 = 0$, then $\Re 5(X) = \{2,3\}$;
- if $x_1 \neq x_2$ & $x_3 = 0$ & $x_1 \neq x_4$ & $x_5 = 1$, then $\Re 5(X) = \{1,4\}$;
- if $x_1 \neq x_2$ & $x_3 = 1$ & $x_1 = x_4$ & $x_6 = 0$, then $\Re 5(X) = \{3,4\}$;
- if $x_1 \neq x_2$ & $x_3 = 1$ & $x_1 = x_4$ & $x_6 = 1$, then $\Re 5(X) = \{1,2\}$;
- if $x_1 \neq x_2$ & $x_3 = 1$ & $x_1 \neq x_4$ & $x_6 = 0$, then $\Re 5(X) = \{1,2\}$;
- if $x_1 \neq x_2$ & $x_3 = 1$ & $x_1 \neq x_4$ & $x_6 = 1$, then $\Re 5(X) = \{3,4\}$.

Theorem 15. $5 \leq C_{UD}(\Re 5) \leq 6$.

Proof. Let us assume there exists a classical randomized decision tree that computes relation $\Re 5$ by asking four queries. We analyze algorithm behavior for the certain input X=010000. According to the definition of the uniformly distributed algorithm, decision tree must output correct values from the result set for each input with equal probability. It means that there has to be a path in the tree, which goes through at most four variable nodes, follows arrows with variable values corresponding to values of X=010000 and finishes in a leaf with the output value «4». On the other hand, for any choice of four

variables for that path, there exists another input X, which equals X in selected four variables values, but does not have «4» among the result set values according to the definition of relation $\Re 5$. All such contradicting inputs are listed in Table 6. For any such input, computation goes through the same path in the decision tree as for X=010000 and finishes in a leaf with incorrect value «4». It is a contradiction, so assumption is wrong and classical randomized decision tree that computes relation $\Re 5$ using only four queries does not exist. □

Table 6

Proof of Theorem 15: contradictions in result set values

Path variables	Input X' contradicting with X=010000	$\Re 5(X')$
x_1,x_2,x_3,x_4	010010	{2,3}
x_1,x_2,x_3,x_5	010100	{2,3}
x_1,x_2,x_3,x_6	010010	{2,3}
x_1,x_2,x_4,x_5	011001	{1,2}
x_1,x_2,x_4,x_6	010010	{2,3}
x_1,x_2,x_5,x_6	010100	{2,3}
x_1,x_3,x_4,x_5	000000	{1,2}
x_1,x_3,x_4,x_6	000000	{1,2}
x_1,x_3,x_5,x_6	000000	{1,2}
x_1,x_4,x_5,x_6	000000	{1,2}
x_2,x_3,x_4,x_5	000000	{1,2}
x_2,x_3,x_4,x_6	000000	{1,2}
x_2,x_3,x_5,x_6	000000	{1,2}
x_2,x_4,x_5,x_6	000000	{1,2}
x_3,x_4,x_5,x_6	000000	{1,2}

Theorem 16. $Q_{UD}(\Re 5) = 2$.

Proof. Quantum query algorithm that computes $\Re 5$ with two queries is presented in Fig. 15. Sign "+" inside question circle signifies that none variable impacts the value of corresponding amplitude. □

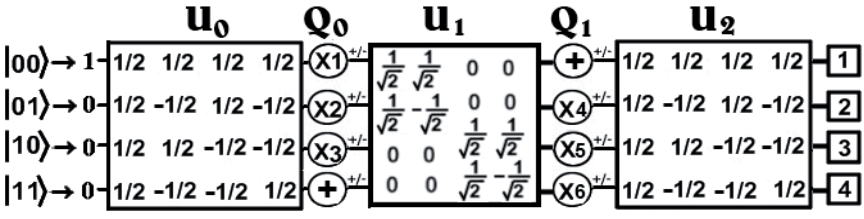


Fig. 15. Quantum query algorithm $Q5$ for computing $\Re 5$ in uniformly distributed manner

6 A Note on Computing Relations in a Definite Manner

In this section, we discuss the first type of query algorithms for relations, which compute relations in a definite manner. Are there prospects to obtain a large separation between classical and quantum complexity?

According to the definition, for each input X, such algorithm always outputs one definite value. The only condition is that this value should be from the result set assigned to that input by relation $\Re$. It actually means that a definite query algorithm for relation $\Re$ computes a function, which is a subset of relation.

When designing a query algorithm to compute relation $\Re$ in a definite manner, we may choose some arbitrary function from a set $Func(\Re)$, which is better suited for computing in a query model, and construct an algorithm for that function. So, classical and quantum query complexities for computing relation definitely are expressed by formulas:

$$D(\Re) = \min_{f \in Func(\Re)} (D(f)) \qquad Q_E(\Re) = \min_{f \in Func(\Re)} (Q_E(f))$$

It appears that the task of enlarging the gap between classical and quantum query complexity to compute relations in a definite manner is completely the same as when computing usual functions in a query model. Even more, the interesting moment is that the functions selected from the set $Func(\Re)$ for computing in classical and quantum cases may also be different. Unfortunately, it does not give us additional tool to enlarge the complexity gap when computing relations instead of functions, quite contrary. For that reason, computing relations in a distributed manner looks much more interesting.

7 Conclusion

In this paper, we considered computing mathematical relations instead of Boolean functions in a query model. Various general computational problems and tasks of certain type in software engineering may be represented in terms of relations. We proposed three types of a query algorithm for computing relations with different output behavior.

We demonstrated several examples of computing relations in classical and quantum versions of a query model.

In the first example, the definition of relation is based on *XOR* operation. We generalized the basic relation and obtained an example, when quantum query algorithm computes relation with N^2 variables using N queries, while $2N$ queries are required in the classical case. This result repeats the largest separation between quantum exact and classical deterministic query complexity for functions that is known for today.

In the second example, a quantum query complexity for relation is more than two times less than classical query complexity for the same relation. However, the considered relation has a property of having inputs for which the result set consists of all range set elements.

In the third example, we considered finite six-variable relation, which is not based on *XOR* operation and there are no inputs for which result set consists of all range set elements. These properties make this relation very interesting. For this relation, quantum query complexity is also more than two times less than classical query complexity.

Finally, we discussed the specifics of computing relations in a definite manner and concluded that the task of computing relations in a distributed manner is more promising for enlarging the gap between classical and quantum query complexity.

Results presented in this paper build a foundation for further investigation. The main goal, which we are looking to achieve, is to construct examples with larger complexity separation between classical and quantum query algorithm complexity. The most important work direction is to develop a technique for proving complexity lower bounds for computing relations in a classical query model.

8 Acknowledgments

We would like to thank our supervisor Rūsiņš Freivalds for familiarizing us with quantum computation and for permanent support and advising.

This work has been supported by the European Social Fund within the project „Support for Doctoral Studies at University of Latvia".

References

1. Buhrman, H., de Wolf, R.: Complexity Measures and Decision Tree Complexity: A Survey. Theoretical Computer Science, v. 288(1): 21-43 (2002)
2. Shor, P. W.: Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing, 26(5):1484-1509 (1997)
3. Grover L.K.: From Schrödinger‘s equation to quantum search algorithm, American Journal of Physics, 69(7): 769-777 (2001)de Wolf, R.: Quantum Computing and Communication Complexity. University of Amsterdam (2001)
4. Ambainis, A.: Quantum query algorithms and lower bounds (survey article). In Proceedings of FOTFS III, Trends on Logic, vol. 23, pp. 15-32 (2004)
5. Kaye, R., Laflamme, R., Mosca, M.: An Introduction to Quantum Computing. Oxford (2007)
6. Ambainis, A., Childs, A., Reichardt, B., Spalek, R., Zhang, S.: Any AND-OR formula of size N can be evaluated in time O(N^{1/2+epsilon}) on a quantum computer. SIAM J. Comput. Volume 39, Issue 6, pp. 2513-2530 (2010).
7. Vasilieva, A., Mischenko-Slatenkova, T.: Quantum Query Algorithms for Conjunctions. Proc. of the UC 2010, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, vol. 6079/2010, ISBN: 978-3-642-13522-4, pp. 140-151 (2010)
8. Kushilevitz, E., Nisan, N.: Communication complexity. Cambridge University Press, (1997)
9. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
10. Weisstein, E. W.: Relation. From MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/Relation.html
11. D. Deutsch and R. Jozsa: Rapid solutions of problems by quantum computation. In Proceedings of the Royal Society of London, volume A 439, pp. 553-558 (1992)
12. R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca: Quantum algorithms revisited. In Proceedings of the Royal Society of London, volume A 454, pp. 339–354 (1998)

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2011. Vol. 770
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 90–111 P.

Fuzzified Algorithm for Game Tree Search with Statistical and Analytical Evaluation

Dmitrijs Rutko[1]
Faculty of Computing, University of Latvia
Raina blvd. 19, Riga, LV-1586, Latvia
dim_rut@inbox.lv

This paper presents a new game tree search algorithm which is based on the idea that the exact game tree evaluation is not required to find the best move. Therefore, pruning techniques may be applied earlier resulting in faster search and greater performance. The experiments show that applied to an abstract domain, the presented algorithm outperforms the existing ones such as PVS, Negascout, NegaC*, SSS*/ Dual* and MTD(f). This paper also provides improvements for algorithm such as statistical and analytical game tree evaluation.

Keywords: game tree search, alpha-beta pruning, fuzzified search algorithm, performance.

1 Introduction

Games are usually represented with the help of a game tree which starts at the initial position and contains all the possible moves from each position. Classical game tree search algorithms such as Minimax and Negamax operate using a complete scan of all the nodes of the game tree and are considered to be too inefficient. The most practical approaches are based on the Alpha-beta pruning technique, which seeks how to reduce the number of nodes to be evaluated in the search tree. It is designed to completely stop the evaluation of a move if at least one possibility is found, the one that proves the current move to be worse than the previously examined move. Such moves do not need to be evaluated further.

The examples of more advanced algorithms that are even faster while still being able to compute the exact minimax value, are PVS, Negascout and NegaC*. The other group of algorithms like SSS* / Dual* and MTD(f), use best-first strategy, which can potentially make them more time-efficient, however, typically at a heavy cost of space-efficiency.

Through analyzing and comparing these algorithms it can be seen that in many cases the decision about the best move can be made before the exact game tree minimax value is obtained. The author introduces a new approach which allows finding the best move faster while visiting less nodes.

The paper is organized as follows: the current situation in the game tree search is discussed; then the idea that allows performing game tree search in a manner

[1] This research is supported by the European Social Fund project No. 2009/0138/1DP/1.1.2.1.2/09/IPIA/VIAA/004.

based on the move that leads to the best result is proposed; the algorithm structure and implementation details are explained. Thereafter, improvements to the algorithm, such as statistical self-learning and analytical evaluation, are discussed. Then, the experimental setup and empirical results on the search performance obtained in abstract domain are shown. The paper is concluded with future research directions.

2 State of the Art

Classical game tree search algorithms are based on the Alpha-beta pruning technique. Alpha-beta is a search algorithm which tries to reduce the number of nodes to be evaluated in the search tree by the Minimax algorithm. It completely stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined one. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision [12].

The illustration of the Alpha-beta approach is given in *Fig. 1*.

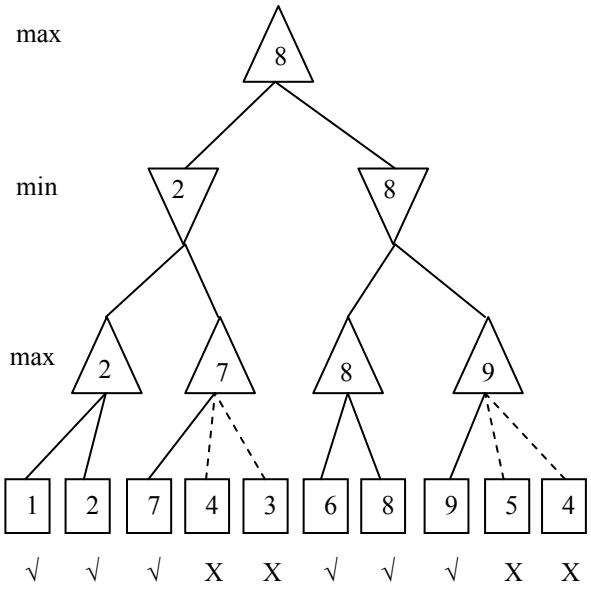


Fig. 1 Traditional Alpha-Beta approach

The game tree in *Fig. 1* has two branches with minimax values 2 and 8 for the left and right sub-trees respectively. In order to find the best move, the Alpha-beta algorithm is scanning all the sub-trees from the left to the right and is forced to evaluate almost each node. The possible cut-offs are depicted with a dashed line (at

each step, the previous evaluation is smaller than the value of currently checked node).

When all the nodes are checked, the algorithm compares the top-level sub-trees. The evaluation of the left and the right branches are 2 and 8 respectively; the highest outcome is chosen, and the best move goes to the right sub-tree.

The benefit of alpha-beta pruning lies in the fact that branches of the search tree can be eliminated. The search time can in this way be limited to the 'more promising' subtree, and a deeper search can be performed in the same time.

Since the minimax algorithm and its variants are inherently depth-first, a strategy such as iterative deepening is usually used in conjunction with alpha-beta so that a reasonably good move can be returned even if the algorithm is interrupted before it has finished execution. Another advantage of using iterative deepening is that searches at shallower depths give move-ordering hints that can help produce cutoffs for higher-depth searches much earlier than would otherwise be possible [11].

More advanced algorithms are the following:

- PVS (Principal Variation Search) is an enhancement to Alpha-Beta based on null or zero window searches of none PV-nodes to prove whether a move is worse or not than the already safe score from the principal variation [1][10].
- NegaScout, which is an Alpha-Beta enhancement. The improvements rely on a Negamax framework and some fail-soft issues concerning the two last plies which did not require any re-searches [3] [4].
- NegaC* – an idea to turn a Depth-First to a Best-First search like MTD(f) to utilize null window searches of a fail-soft Alpha-Beta routine and to use the bounds that are returned in a bisection scheme [5].
- SSS* and its counterpart Dual* are search algorithms which conduct a state space search traversing a game tree in a best-first fashion similar to that of the A* search algorithm and retain global information about the search space. They search fewer nodes than Alpha-Beta in fixed-depth minimax tree search [2].
- MTD(f), the short name for MTD(n, f), which stands for Memory-enhanced Test Driver with node n and value f. MTD is the name of a group of driver-algorithms that search minimax trees using null window alpha-beta with transposition table calls. In order to work, MTD(f) needs a first guess as to where the minimax value will turn out to be. The better than first guess is, the more efficient the algorithm will be, on average, since the better it is, the less passes the repeat-until loop will have to do to converge on the minimax value [6] [7] [8] [9].

Transposition tables are another technique which is used to speed up the search of the game tree in computer chess and other computer games. In many games, it is possible to reach a given position, which is called transposition, in more than one way. In general, after two moves there are 4 possible transpositions since either player may swap their move order. So it is still likely that the program will end up analyzing the same position several times. To avoid this problem transposition tables store previously analyzed positions of the game [11].

3 Fuzzy Approach

The author proposes a new approach, which is based on the attempt to implement a human way of thinking adapted to logical games. A human player rarely or almost never evaluates a given position precisely. In many cases, the selection process is limited to rejecting less promising nodes and making certain that the selected option is better than others. The important moment is that we are not interested in the exact position evaluation but in the node which guarantees the highest outcome.

Let the given problem be explained in details.

We could look at our game tree from a relative perspective like "is this move better or worse than some value X" (*Fig. 2*). At each level, we identify if a sub-tree satisfies "greater or equal" criteria. So passing search algorithm, for instance, with argument 5, we can obtain the information that the left branch has value less than 5 and the right branch has value greater or equal than 5. We do not know exact sub-tree evaluation, but we have found the move, which leads to the best result.

In this case, different cut-offs are possible:

- at max level, if the evaluation is greater (or equal) than the search value;
- at min level, if the evaluation is less than the search value.

In the given example, reduced nodes are shown with dashed line. Comparing to *Fig. 1* it can be seen that not only more cut-offs are possible, but also pruning occurs at higher level which results in better performance.

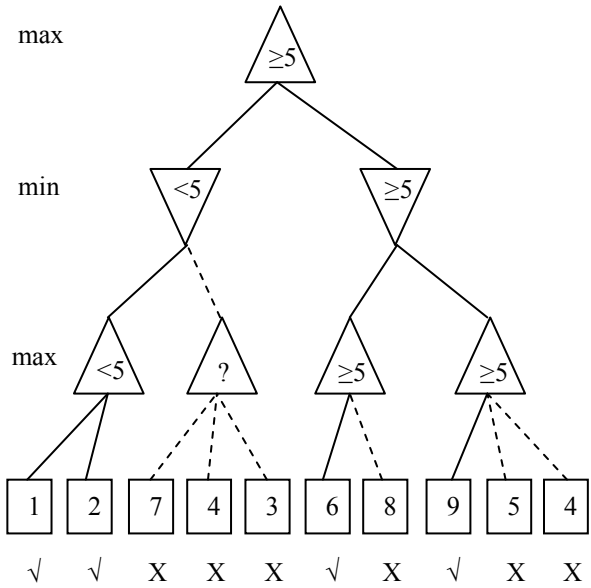


Fig. 2 Fuzzy best node approach

In this approach, the best/worst cases are the same as for alpha-beta pruning: $O(w^{d/2})$ for the best case as only one branch should be checked at cut-off level, and $O(w^{d})$ for worst case as all nodes should be checked (w is width, d is depth of the tree). But in the presented approach, cut-offs are more often possible in general.

If we use geometric interpretation and put our sub-tree minimax values on coordinate axis, then our task is to separate/divide branches so that only one branch would have higher value than the test value. *Fig. 3* illustrates our previous example. Alpha-beta window is initially set to leaf node range $\alpha = 0$, $\beta = 10$; then the following test values are used X_1, X_2, X_3. If value X_2 is chosen, then the successful separation is obtained after the first iteration – we know that the second sub-tree has higher estimation. If values X_1 or X_3 are chosen, then no separation is possible at this point – both values are on the same side of the test value. In this case, the algorithm continues with reduced alpha-beta search window: 1) $\alpha = X_1$ in the first case; or 2) $\beta = X_3$ in the second.

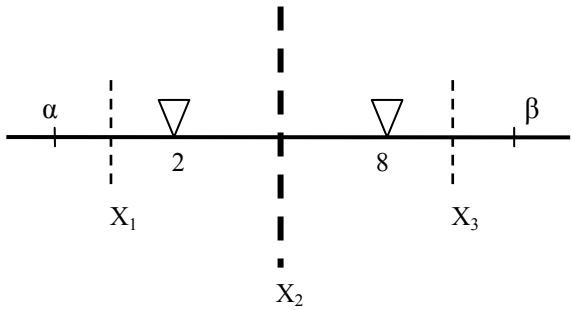


Fig. 3 Geometric interpretation of separation in the fuzzified game tree search

In a game tree with three or more sub-trees, the algorithm workflow remains the same. Our task is to separate sub-trees in a way that only one branch has higher estimation than the test value. However, more cases are possible – 0, 1, 2, 3 branches fall in on one side of separation line. In this case, alpha-beta window is reduced correspondingly and the algorithm proceeds with the next iteration.

Comparing to existing algorithms such as MTD(f) in order to work, it needs the first guess as to where the minimax value will turn out to be. If you feed MTD(f) the minimax value to start with, it will only do two passes, the bare minimum: one to find an upper bound of value x, and one to find a lower bound of the same value.

In the presented algorithm, it is possible to find the best move with a single iteration and we are not limited to the accurate first guess. For the presented example, any value from interval 3..7 (inclusive) would apply.

4 Fuzzified Search Algorithm

Best Node Search (BNS) is a new game tree search algorithm based on the idea described in the previous section. The main difference between the classical approach and the proposed algorithm is that BNS does not require the knowledge of the exact game tree minimax value to select a move. We only need to know which sub-tree has higher estimation. By iteratively performing search attempts the algorithm can obtain information about which branch has higher estimation without knowing the exact value. So less information is required and, as a result, the best move can be found faster – total number of searched nodes is smaller and total

algorithm execution time is reduced comparing to the algorithms based on the exact game tree evaluation.

The presented algorithm uses a standard call of Alpha-Beta search with 'zero window'. The proposed implementation relies on the transposition tables but variation without memory (transposition tables) usage is also possible. While scanning a game tree, algorithm checks all sub-trees and returns node which leads to the best result. In general, BNS is expected to be more efficient comparing to the classical algorithms in terms of number of nodes checked as it does not obtain additional information which is not required in many cases – the exact game tree minimax value.

BNS algorithm is given in *Fig. 4* which makes use of the following functions:

1. NextGuess() – returns next separation value tested by algorithm;
2. AlphaBeta() – alpha-beta search with Zero Window (Null Window) performs a boolean test whether a move produces a worse or better score than the passed value.

All sub-trees are tested with separation values (this information is stored in the transposition tables and reused in subsequent iterations). If exactly one branch exceeds test value, then the best node is found. If all branches have smaller estimation, then the number of sub-trees that exceeds separation test value remains the same, beta value is reduced. If several nodes exceed test value, then `subtreeCount` is updated correspondingly, and alpha value is updated to test value, and algorithm continues with the next iteration. If a single sub-tree that exceeds test value cannot be found and alpha-beta range is reduced to 1, it means that several sub-trees have the same estimation and we can choose any of them.

```
function BNS(node, α, β)
    subtreeCount := number of children of node
    do
        test := NextGuess(α, β, subtreeCount)
        betterCount := 0
        foreach child of node
            bestVal := -AlphaBeta(child, -test, -(test - 1))
            if bestVal ≥ test
                betterCount := betterCount + 1
                bestNode := child
        update number of sub-trees that exceeds separation
  test value
        update alpha-beta range
    while not((β - α < 2) or (betterCount = 1))
    return bestNode
```

Fig. 4 The BNS algorithm

One of the main parts of this algorithm is the method `NextGuess(α, β, subtreeCount)` which returns the next value to be checked by the algorithm. In the simplest case, it could be a formula based on linear distribution – alpha-beta range is proportionally divided into sections according to the sub-tree count:

```
NextGuess = α + (β - α) * (subtreeCount - 1) / subtreeCount;
```

where alpha and beta are the lower and the upper bounds of the search window respectively; `subtreeCount` is the number of sub-trees which satisfies the previous test call (the branches that have higher estimation than the test value). However, the best algorithm performance is achieved after its statistical training or analytical game tree evaluation resulting in non-linear distributions. These methods are described in the following chapters.

5 BNS Enhancement through Statistical Training

Some algorithms, such as MTD(f) benefit from accurate "first guess" – as to where the minimax value will turn out to be. The better than first guess is, the more efficient the algorithm will be, on average.

The BNS algorithm can greatly benefit from good separation value as well. The better separation value is, the faster the best node will be found, on average. So self-training becomes an important part of the BNS algorithm as it helps us to tune separation test values used by algorithm during consecutive search attempts and results in reduced search space and improved performance [12].

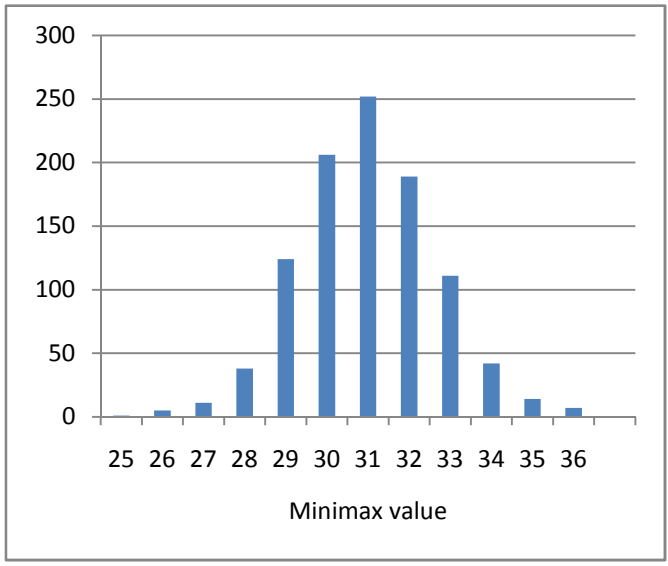
In this section, the author proposes a new multi-dimensional statistics approach which is developed to work in conjunction with BNS algorithm.

It is possible to collect this statistics before the game starts analyzing multiple test data or on-line during the game process reusing previous estimations.

Table 1

Game tree minimax value distribution over 1000 trees

Minimax value	Tree count
25	1
26	5
27	11
28	38
29	124
30	206
31	252
32	189
33	111
34	42
35	14
36	7
	1000



The statistical approach for finding initial value (first guess) can be demonstrated in the following example. 1000 game trees were generated with fixed

structure and randomly assigned values for leaf nodes in a specific range (for given example, the following values were used – width 2, depth 14, leaf node values are in interval [0; 80]). For these game trees, statistics was collected and results are shown in *Table 1*

It can be seen that there are 252 trees with minimax value 31 and there is only one tree out of one thousand with minimax value 25. These statistics results are used, for example, to determine the first guess in MTD(f) algorithm, and in all tests it was called with argument f = 31 showing its best results.

However, this information does not provide additional benefits. So new approach is proposed – single-dimension statistics is extended into two-dimension statistics meaning collecting all possible pair info – for each sub-tree in our binary tree. As a result, we have a matrix showing a number of trees having respectively one sub-tree value (columns) and other sub-tree value (rows) – *Table 2* Due to symmetry reasons (according to the main diagonal) one half is shown. Tree count column has summed up matrix values in the row resulting in the previous single-dimension statistics (*Table 1*).

It can be seen that there are 78 trees which have sub-trees correspondingly with branch values 31 and 29 (in this case, tree minimax value is 31).

Table 2

Two dimensional game sub-tree distribution over 1000 trees

	23	24	25	26	27	28	29	30	31	32	33	34	35	36	Tree count
23	0														0
24	0	0													0
25	0	1	0												1
26	0	0	2	3											5
27	0	0	5	3	3										11
28	0	1	0	12	12	13									38
29	0	0	2	10	35	43	34								124
30	1	2	6	9	26	58	71	33							206
31	0	0	6	10	27	41	78	57	33						252
32	0	1	3	13	17	30	32	41	38	14					189
33	0	0	1	2	8	12	26	28	21	11	2				111
34	0	0	0	1	3	5	13	8	6	2	2	2			42
35	0	0	0	0	0	2	4	3	2	3	0	0	0		14
36	0	0	0	0	0	0	1	2	2	1	1	0	0	0	7
															1000

BNS algorithm divides search interval into parts and verifies if sub-tree values stay in different parts or not. If one branch value is less than separation value and another branch value is higher, then algorithm immediately returns better move and stops its work. If the branch values lay in the same part, then the interval is reduced and the algorithm continues with an updated alpha-beta window. So, the algorithm

becomes more efficient with the accurate first guess when the most of the game trees get separated into parts after the first iteration.

The grayed-out rectangle in *Table 2* gives us separation distribution for X = 30. All marked cells represent trees with one branch greater or equal than 30 (by row) and the other branch less than 30 (by column). It means that all these trees will be separated into parts after the first iteration. To calculate the number of trees for separation value X = 30, we need to sum up all the marked cells. For the given example, 509 trees will get separated.

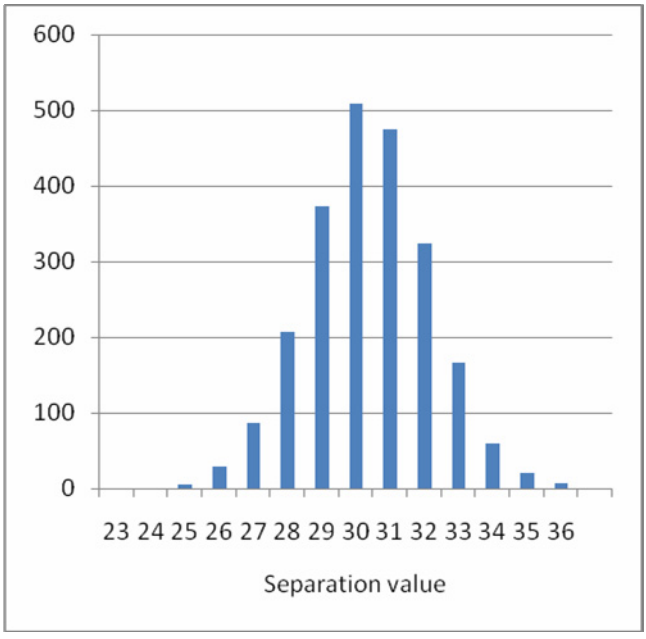
So, to find such value X when the highest number of trees will be divided, we need to build remaining rectangles along the main diagonal for each X value and sum up the cells bounded by X along axis as it is done in the previous example. The resulting table is shown in *Table 3*

As it can be seen from *Table 3*, the best results are given with X = 30, meaning that if we call BNS algorithm with argument 30, then 509 game trees will be divided into two parts and the best node will already be found after the first iteration. So trained BNS is more efficient and if we continue this idea we can find the best X value for the second, third etc. iteration, until the best node is found.

Table 3

Statistical sub-tree separation over 1000 trees

Separation value	Tree count
23	0
24	1
25	6
26	30
27	88
28	208
29	374
30	509
31	475
32	325
33	167
34	61
35	21
36	7
	2272



Note: the total count of the game trees is higher than 1000 as many values are overlapping – the same X value could divide different trees and the same tree could be divided by different X values.

If we take a look at the tree with branching factor 3, we can apply similar techniques for finding the best separation value. In this case, we have triplets [x, y, z] defining minimax value of each sub-trees, so we can build corresponding 3D matrix displaying the total number of the game trees with the given triplet.

While searching this matrix, we look for such separation value X, so one sub-tree would be greater or equal with X, and two other sub-trees would have smaller estimation. And, therefore, we maximize the number of trees which would be separated after the first method call, so the best move is found after the first iteration.

6 Game Tree Analytical Evaluation

In the previous chapter (BNS enhancement through self-training), statistical analysis, which can improve BNS algorithm performance by calculating and applying "good" separation values, was discussed. Therefore, in the development of this idea, the author offers a new approach which is based on fully analytical determination of best successful separation value generally for any type of tree with various structures (alpha-beta range, tree width, depth, etc).

As it was stated before, we use abstract domain search in our experiments – meaning tree generation with fixed structure (width / depth) and randomly assigning leaf values based on uniform distribution within a given range.

In *Fig. 5*, leaf nodes are noted as probabilistic function F_X. Here, our task is to calculate resulting function starting from the lowest level (leaf nodes) up to the top level (root node).

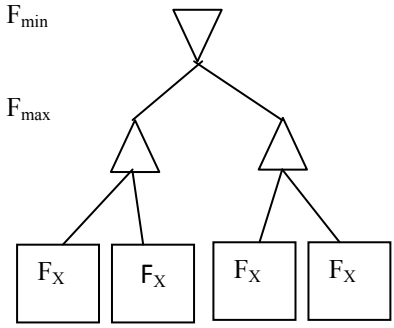


Fig. 5 Application of probabilistic function to maximum and minimum levels

In this case, the following functions demonstrate the behavior of leaf nodes:

- *Probability density function* describes the relative likelihood for this random variable to occur at a given point. For our example (leaf node values are in interval [0; 80]), this likelihood is given in Fig. 6;
- *Cumulative distribution function* describes the probability that a real-valued random variable X with a given probability distribution will be

found at a value less than or equal to X. For our example, it is given in Fig. 7.

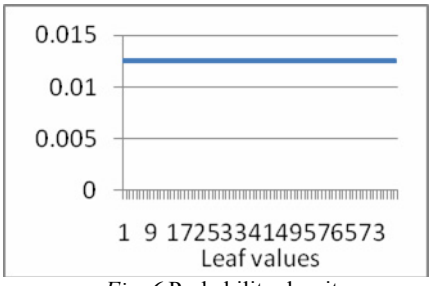


Fig. 6 Probability density

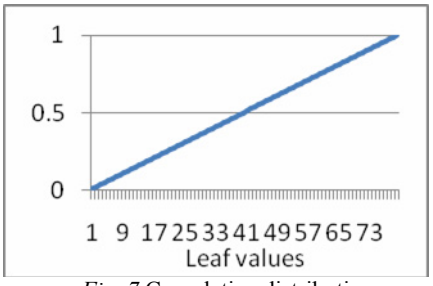


Fig. 7 Cumulative distribution

To calculate probabilistic values correspondingly at maximum and minimum levels, the author proposes the following formulas which are applicable for binary tree (square of probabilistic function) – for max level, it is probability that both sub-trees are less than our cumulative distribution function; for min level, that not both elements are greater than our cumulative distribution function:

$$F_{max} = (F_x)^2 \quad (1)$$

$$F_{min} = 1 - (1 - F_x)^2 \quad (2)$$

For trees with larger branching factor, the following general formula should be used, where *w* is the width of the tree:

$$F_{max} = (F_x)^w \quad (3)$$

$$F_{min} = 1 - (1 - F_x)^w \quad (4)$$

Correspondingly, if we apply this formula to our example with binary tree with leaf nodes in the given range [0..80], we receive the following equations:

$$F_{max} = \left(\frac{x}{80}\right)^2 \quad (5)$$

$$F_{min} = 1 - \left(1 - \frac{x}{80}\right)^2 \quad (6)$$

By using these formulas we can build up the following matrix (*Table 4.*) with iteration results and iteration values for each minimum and maximum level up to level of depth 14 (actually, we start from the lowest level with leaf nodes and go up to the highest level – the root node).

Table 4

Calculated cumulative distribution for binary tree with leaf node values from interval [0; 80] and depth 14

Leaf values		*Level*				
		1 – min	2 – max	3 – min	…	14 – max
x	F_x	$1-(1-F_x)^2$	$(F_x)^2$	$1-(1-F_x)^2$	…	$(F_x)^2$
1	1 / 80	0,02484375	0,00061721	0,00123404	…	0
2	2 / 80	0,049375	0,00243789	0,00486984	…	0
3	3 / 80	0,07359375	0,00541604	0,01080275	…	0
…	…	…	…	…	…	…
80	80 / 80	1	1	1	…	1

Fig. 8 demonstrates the progress of cumulative probability function bottom up changing its slope and coming nearer to vertical. Correspondingly, the transformed probability density function is displayed in *Fig. 9* with higher and higher peaks at each next level where the highest peak corresponds to level 14.

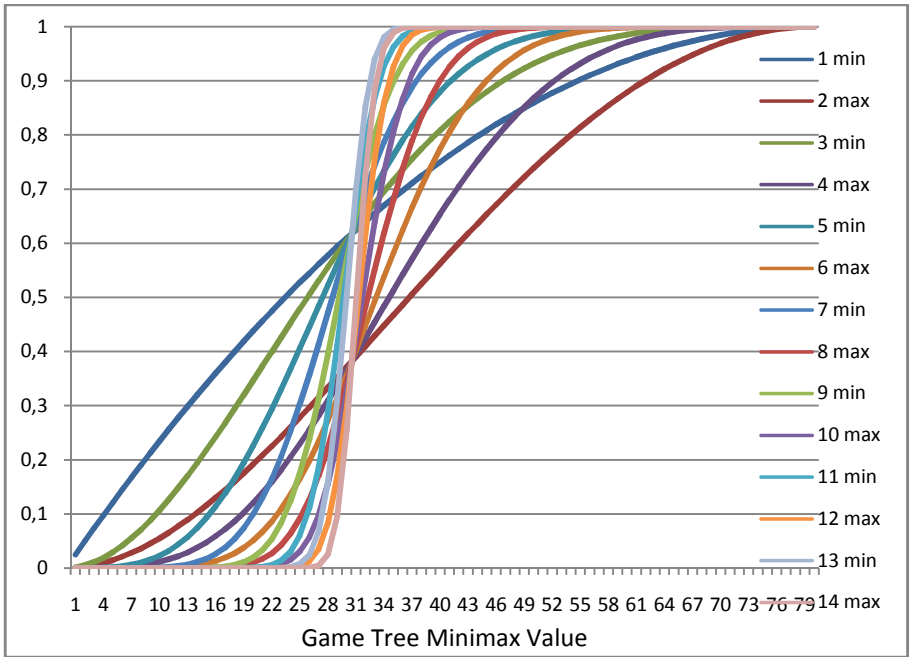


Fig. 8 Cumulative probability function by level for depth 14

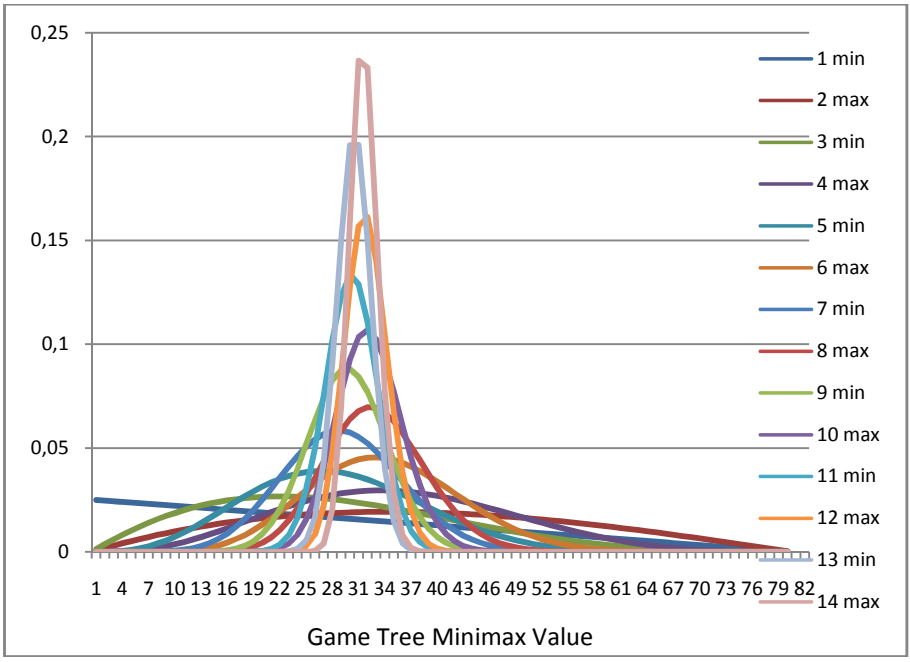


Fig. 9 Probability density function by level for depth 14

In the conducted experiments, statistical information is collected to prove the correctness of analytical game tree evaluation. The difference between analytically received data and statistical experiments is shown in *Fig. 10*. The error rate is relatively low meaning that analytical estimation is really close to experimentally received results.

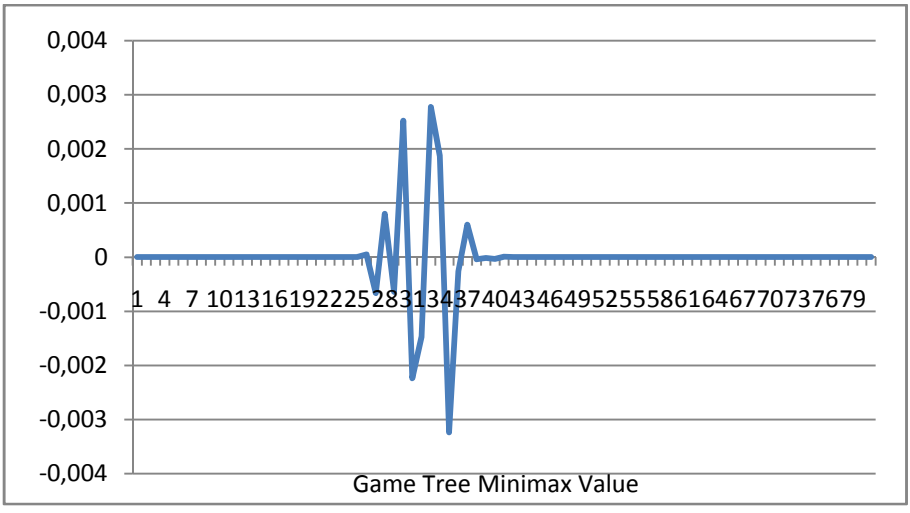


Fig. 10 Error function between analytical estimation and experimentally received results

Resulting probability density function is given in *Fig. 11*. These results correspond to the statistically received results in previous section.

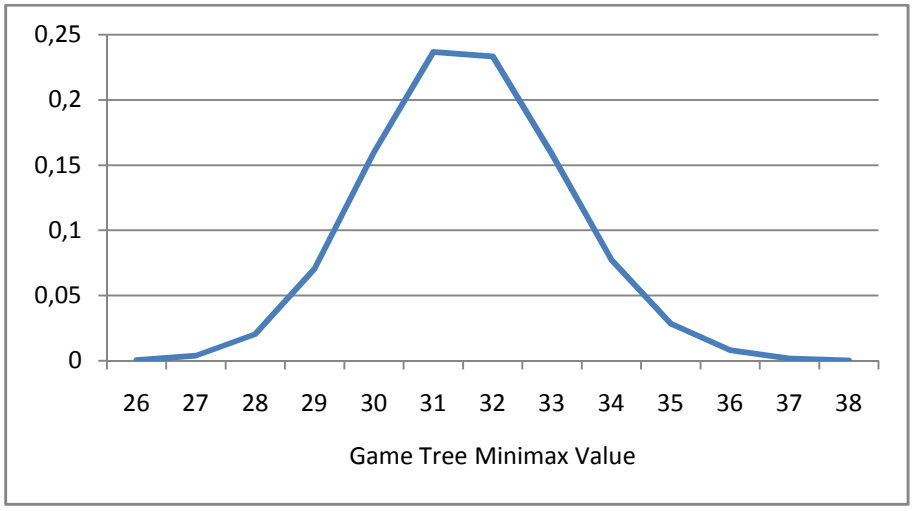


Fig. 11 Resulting zoomed-in function

Given the probability density function we can predict the most probabilistic outcome of the game tree. Thus, we can choose the best separation value for our BNS algorithm – such value X that the greatest number of trees will be separated / divided after the first iteration of the algorithm.

These are the same values as in statistical evaluation we have been used before, except that analytically we could improve precision and make calculations much faster without performing long-running experiments.

We are querying our tree with some separation value X. So, given density function, we can calculate probability, that the tree value is less than our test value, or the tree value is greater. So, our task is to maximize our chances to separate tree with the given value X.

The entropy, H, of a discrete random variable X is a measure of the amount of uncertainty associated with the value of X.

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

Having probabilistic outcome when tree is separated with probability P, and its counterpart outcome when tree is not separated with probability 1-P, results in Binary entropy function, H_b. The entropy is maximized at 1 bit per trial when the two possible outcomes are equally probable, as in an unbiased coin toss.

$$H_b(p) = -p \log_2 p - (1-p) \log_2 (1-p)$$

So, we should find such separation value that maximizes amount of information received after querying the tree. For the first iteration, we receive value 30. For the second iteration, we do the same way – if separation is not obtained after the first query, that means all sub-trees are either less (fall down) or greater (fall up). So, we chose the next separation value in the given range maximizing probability of successful separation. Correspondingly, the separation values for the second iteration are 29 and 31 respectively.

In *Fig. 12*, separation value X_1 is shown for the first iteration. If no successful separation is obtained after the first query, then we use the next group of separation values X_2 going to the left or to the right.

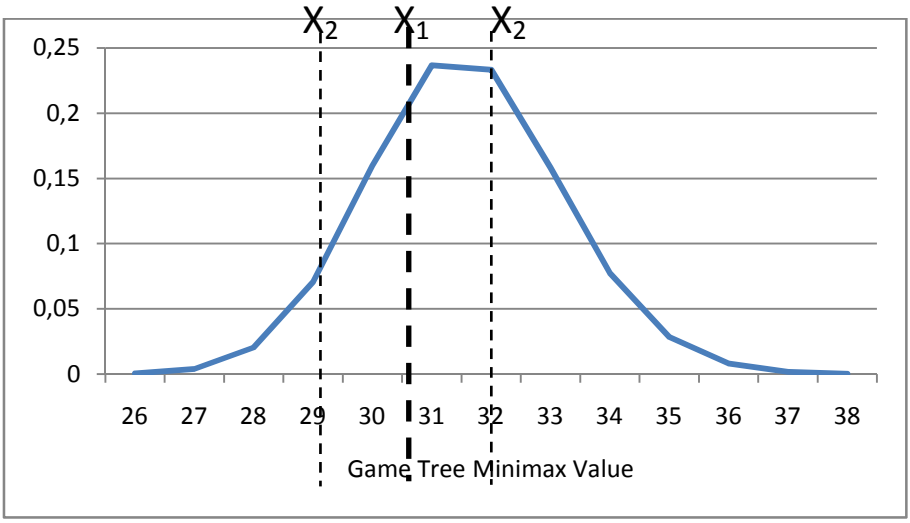


Fig. 12 Separation usage by resulting density function

Similarly, we seek for separation values for the third, the fourth etc. iterations until the best value is found. At each step, we reduce alpha-beta window. This process is similar to binary search, except for the selection separation coefficients where we use probability density function.

7 Experimental Results

More than 10 algorithms were implemented and over 40000 test runs were conducted during this research. Both versions with transposition tables and without them were used in our setup.

These algorithms were tested in an abstract domain – generating the game tree test set with fixed structure (width / depth) and randomly assigning leaf node values from the given range. Then, these experiments extended to trees with a different branching factor starting from 2 to 5 and full alpha-beta window (not limited range [-infinity, +infinity]).

All the algorithms were run on the same game tree test set (each consisting of 10000 generated samples) to compare algorithm efficiency under the same

conditions. For each algorithm, visited leaf nodes count (evaluation function call) and total visited node count was measured and average count per tree was calculated. In most cases, the first parameter is more important as in real games evaluation functions are usually complex enough and require some computing resources. The second parameter is usually less important but for some algorithms total node count increases dramatically and should be considered while comparing algorithm efficiency. In the algorithms with reiterative techniques based on transposition tables when node is visited multiple times, the total node count is increased and leaf node count remains the same as this info is stored in TT.

In the chart in *Fig. 13*, MTDF performance is taken as the base point (treated as 100%) and the performance of other algorithms is measured as a ratio to it, so a result greater than 100% means larger number of search iterations and respectively only BNS was able to show results less than 100%. It is a combined graph showing trends increasing width of the search tree – from binary tree to a tree with 5-width structure at each node. In this section, number of visited leaf nodes is counted.

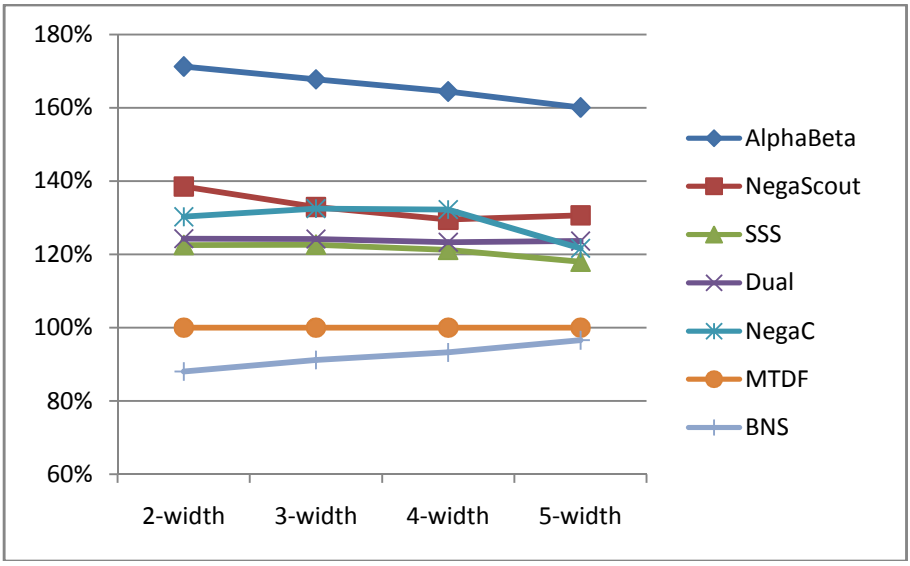


Fig. 13 Algorithm relative performance across different tree widths (leaf nodes visited)

BNS algorithm shows progress from 88% at 2-width stage to 96% at 5-width stage.

Fig. 14 demonstrates the same data slice, but here, the total number of visited nodes is measured. It can be seen that BNS performance still remains at the level of approximately 80% comparing to MTDF algorithm across all branching factors. Note: SSS and Dual algorithms show low results of 700% and 300% correspondingly and fall outside of diagram range.

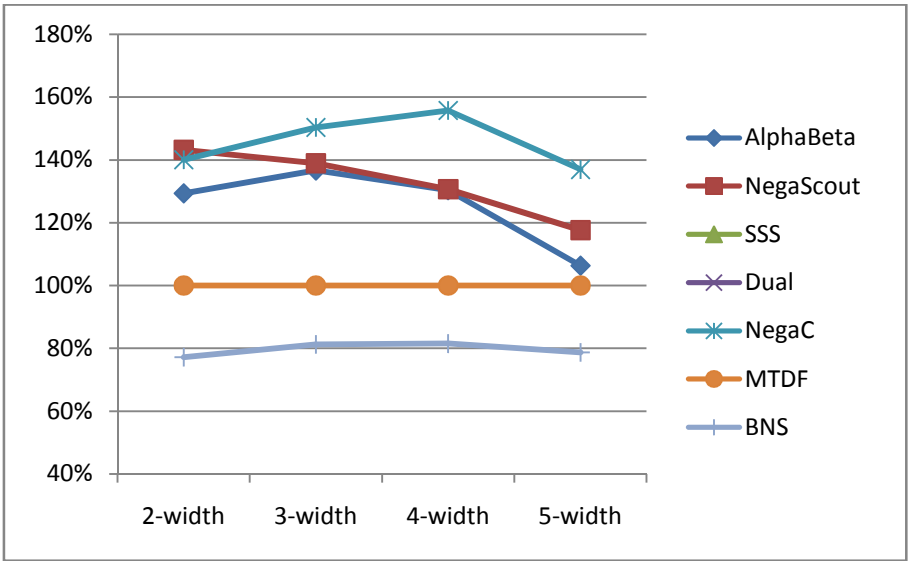


Fig. 14 Algorithm relative performance across different tree widths (total nodes visited)

8 Conclusions and Future Work

The main goal of this paper was to show that it is possible to find the best move without the exact tree minimax value. After self-training based on multi-dimension statistics, the proposed BNS algorithm was able to demonstrate better results than other existing algorithms. Game tree analytical evaluation gives additional improvement allowing us to use this algorithm as general purpose approach for different tree types.

Having analyzed the results we can conclude the following:

- Among algorithms without Transposition Tables (TT) BNS shows competitive results. Both leaf node count and total node count is less comparing to other algorithms;
- The algorithms based on TT approach show different performance in different conditions. Currently, MTD(f) is more preferable choice providing the highest performance. But in the current experiments, BNS demonstrated itself to be more efficient comparing both scanned leaf node count and total node count;
- Considering leaf nodes visited, BNS algorithm demonstrates an improvement in a range from 12% (for binary trees) to 4% (for 5-width trees) comparing to MTD(f);
- Regarding total nodes visited, BNS algorithm demonstrates a stable improvement up to 20% across different branching factors comparing to MTD(f);

The current results are based on experiments in abstract domain and additional research is needed to verify the behavior of the algorithm for wider trees (with

branch factor larger than 15-20). Interesting results may be obtained in testing non-regular trees with asymmetrical structure. Future experiments should also consider analyzing algorithm performance in real games, but it is believable that proposed approach could be successfully applied for real domain games as well.

9 Acknowledgments

The author would like to thank Nikolajs Nahimovs for valuable ideas in the field of game tree analytical evaluation.

References

1. T. A. Marsland, M. Campbell. Parallel Search of Strongly Ordered Game Trees. ACM Comput. Surv., 1982
2. Judea Pearl. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. Communications of the ACM, 1982
3. Reinefeld, A. An Improvement to the Scout Tree-Search Algorithm. ICCA Journal, 1983, Vol. 6, No. 4, pp. 4-14
4. A. Reinefeld. Spielbaum-Suchverfahren. Informatik-Fachbericht 200, Springer-Verlag, 1989
5. Jean Christophe Weill. The NegaC* search. ICCA Journal, March 1992
6. Plaat, A., Schaeffer, J., Pijls, W., and Bruin, A. de. A New Paradigm for Minimax Search, Technical Report EUR-CS-95-03, 1994
7. Plaat, A., Schaeffer, J., Pijls, W., and Bruin, A. de. Best-First and Depth-First Minimax Search in Practice, Proceedings of Computing Science in the Netherlands, 1995
8. Plaat, A., Schaeffer, J., Pijls, W., and Bruin, A. de. An Algorithm Faster than NegaScout and SSS* in Practice, Computer Strategy Game Programming Workshop at the World Computer Chess Championship, 1995
9. Plaat, A., Schaeffer, J., Pijls, W., and Bruin, A. de. Best-First Fixed-Depth Minimax Algorithms, Artificial Intelligence, volume 87, 1996
10. Yngvi Björnsson. Selective Depth-First Game-Tree Search. Ph.D. thesis, University of Alberta, 2002
11. Russell, Stuart J.; Norvig, Peter, Artificial Intelligence: A Modern Approach (3rd ed.), Upper Saddle River, New Jersey: Pearson Education, Inc., 2010
12. Dmitrijs Rutko, Fuzzified Algorithm for Game Tree Search. Second Brazilian Workshop of the Game Theory Society, BWGT 2010

Appendix

The following section contains the performance results of algorithms implemented during the current research for different tree structures and leaf node ranges.

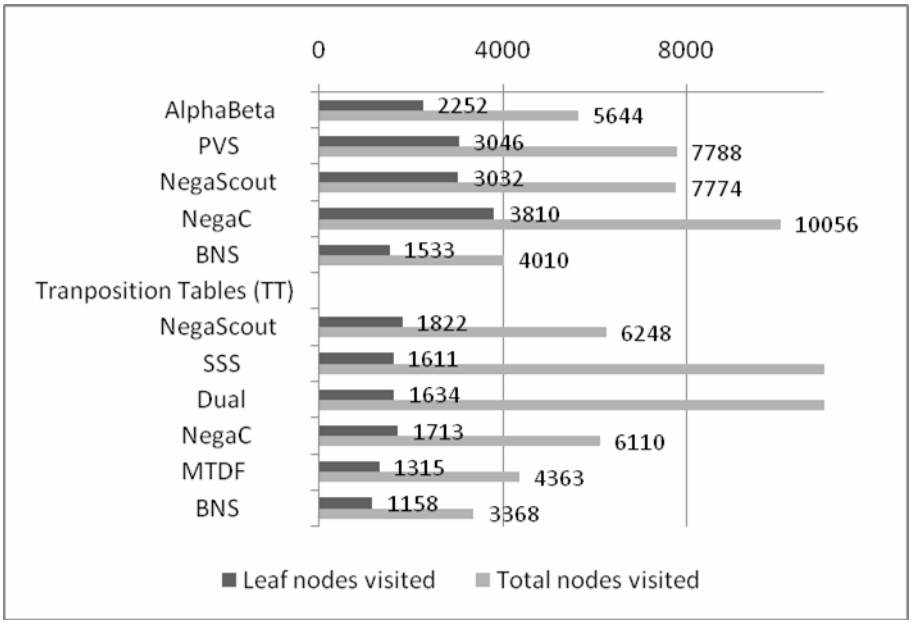


Fig. 15 Tree width – 2, depth – 14

Leaf node range 0..80; full alpha-beta window

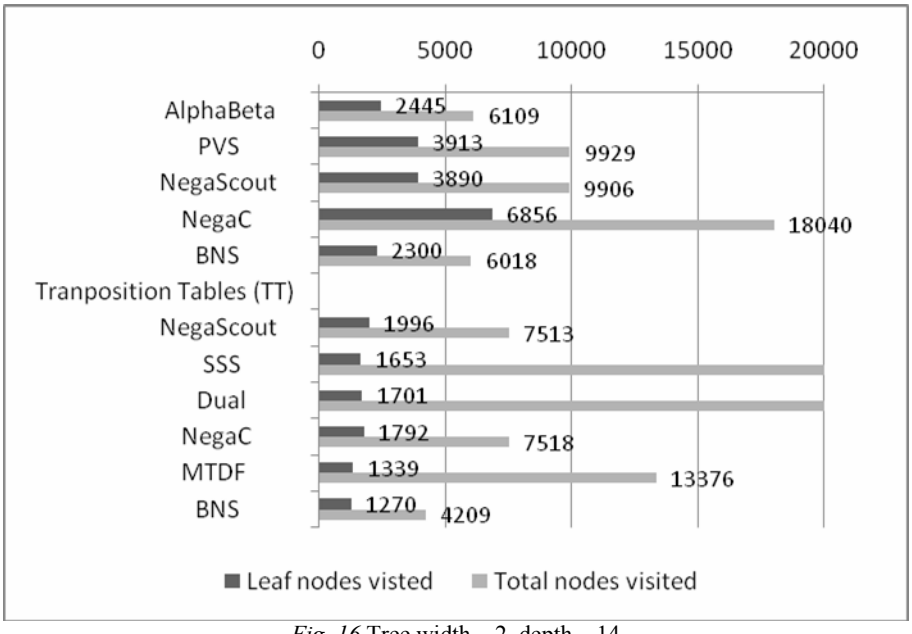


Fig. 16 Tree width – 2, depth – 14

Leaf node range 0..800; full alpha-beta window

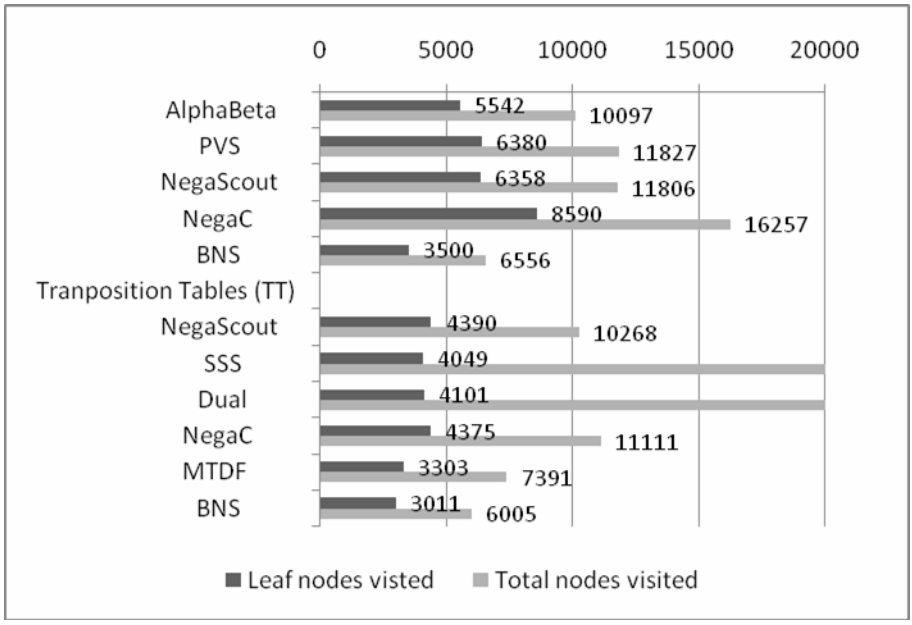


Fig. 17 Tree width – 3, depth – 10

Leaf node range 0..80; full alpha-beta window

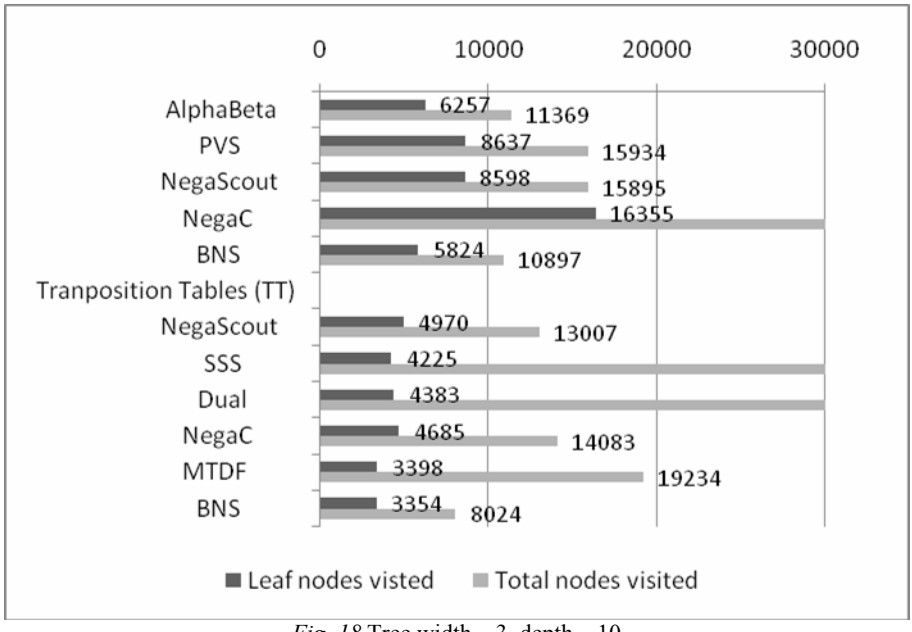


Fig. 18 Tree width – 3, depth – 10

Leaf node range 0..800; full alpha-beta window

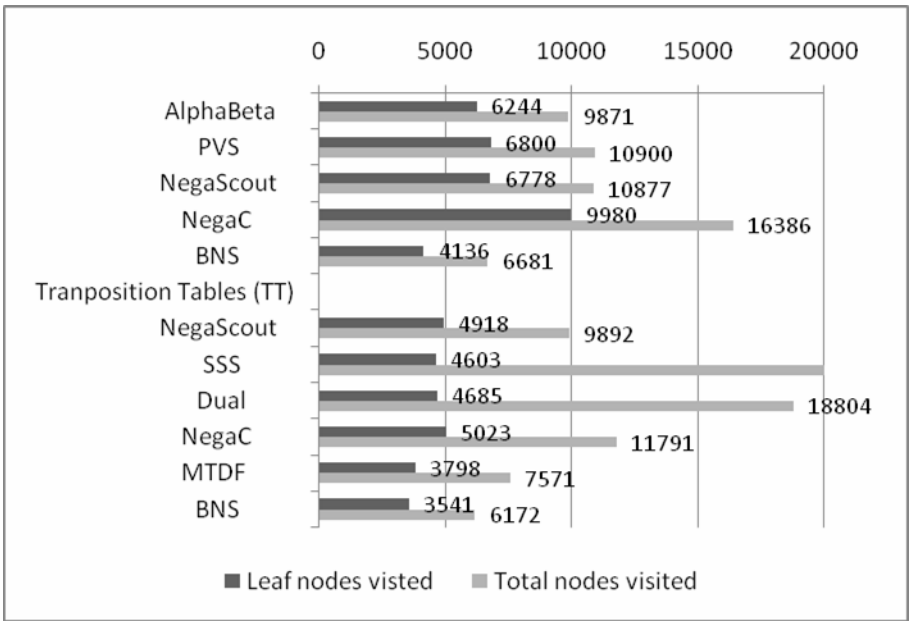


Fig. 19 Tree width – 4, depth – 8

Leaf node range 0..80; full alpha-beta window

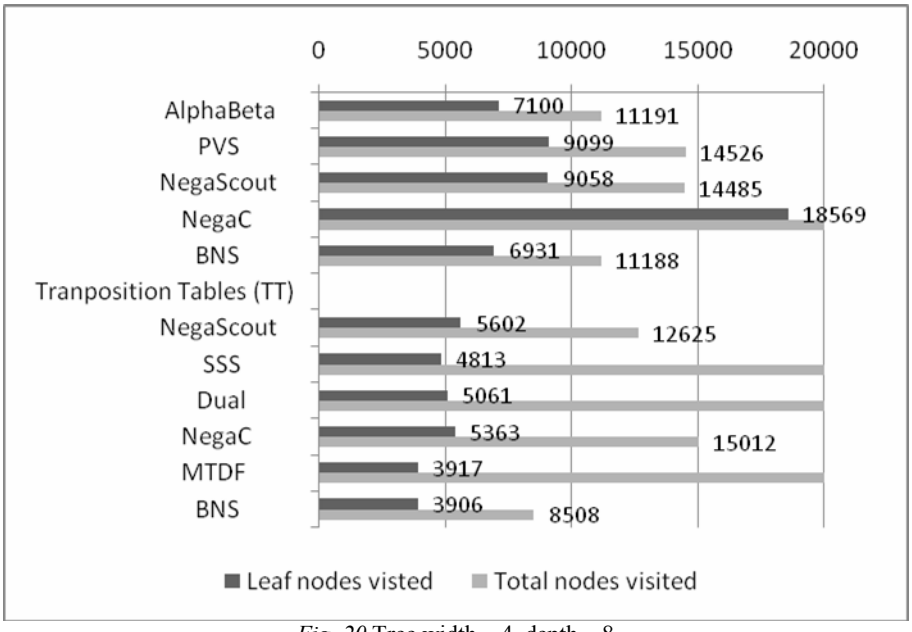


Fig. 20 Tree width – 4, depth – 8

Leaf node range 0..800; full alpha-beta window

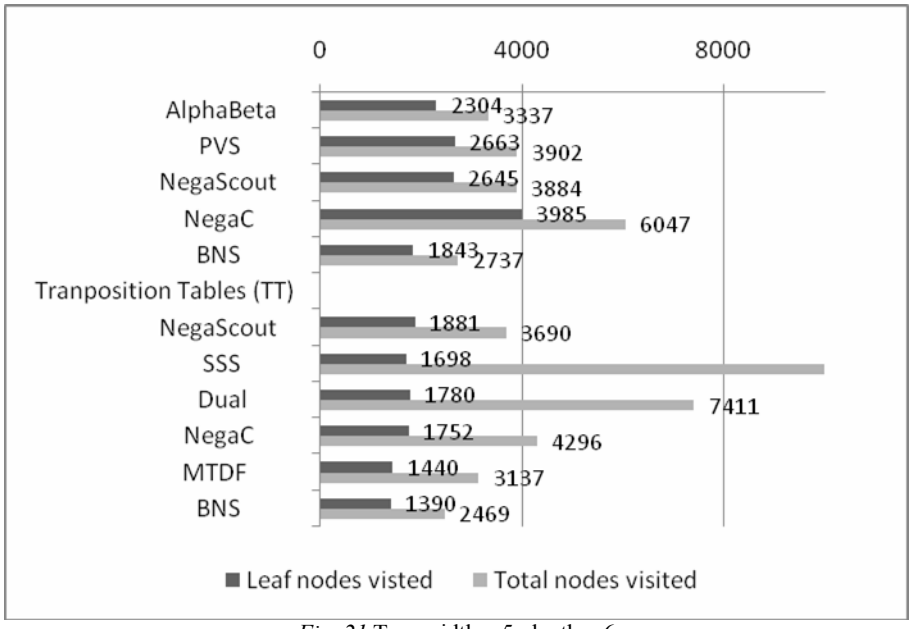


Fig. 21 Tree width – 5, depth – 6

Leaf node range 0..80; full alpha-beta window

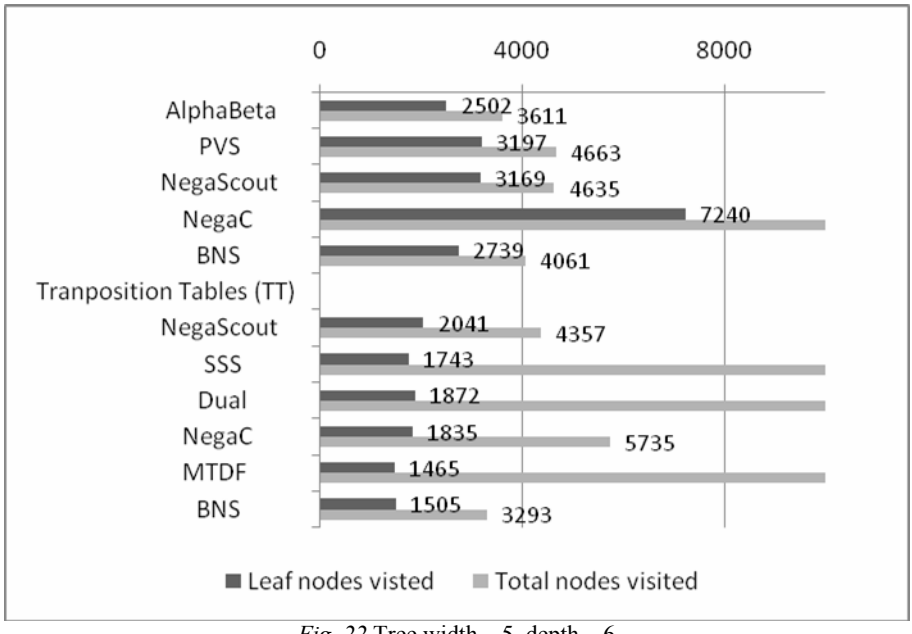


Fig. 22 Tree width – 5, depth – 6

Leaf node range 0..800; full alpha-beta window

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2011. Vol. 770
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 112–123 P.

Computer Scientists as Early Digital Artists

Solvita Zariņa
Institute of Mathematics and Computer Science, University of Latvia
29 Raina blvd., Riga, LV-1459, Latvia
solvita.zarina@lumii.lv

Computer graphics and early computer art are practically of the same age as computers. Since the moment when graphical output devices became available people started to use them for experiments in art. This paper attempts to analyze the early computer art in context of its authors' opinions. There are outstanding examples of collaboration between computer scientists, software programmers and 20th century artists. Some recent media art events in Latvia are mentioned from this viewpoint.

Keywords: computer graphics, computer art, digital art.

1 Introduction

The first visual examples created by using an analogue computer appeared in the mid-1950s in the United States and Germany. We can find some experiments with oscilloscope images even before the stage of electronic graphics. The point of interest is why anybody dares to call these pieces the art. We can prove it from the viewpoint of the history of art of the 20th century. Art forms, such as cubism, Dada, futurism, naïve art, primitivism, constructivism, suprematism and kinetic art, had already emerged before and were widely considered to be the art forms in the middle of the century. Abstract expressionism was one of the major stylistic approaches in the US at that time. Therefore, three things characterising the first examples of computer art are: art as a process not only as a result (Dada), non-professional artists (naïve and primitive art) and abstract, non-representative art forms (suprematism, constructivism, abstractionism) have been already accepted by art theoreticians, gallery and museum curators, as well as by general public in the Western world.

The first authors of the computer images are mathematicians, computer scientists and engineers. It seems to be self-understandable considering the availability of computer time being extremely expensive at that time and necessity for high-level technical skills. It has to be mentioned that the first computer artists had their own theoretical viewpoint on the style and aesthetics of their newly created art form.

Almost ten years later, the informal communities consisting of professional artists (visual artists, as well as musicians, filmmakers, etc.) of the one part and scientists, engineers of the other part began to spread out in order to combine the creative potential of art with computer technologies. Some well-known examples of cooperation in such countries as the United States, Japan and the United Kingdom will be analyzed. Despite the political system and official politics in Soviet Latvia, some kinetic and even conceptual art examples emerged in 1970ties. They were often marked as design proposals and implemented in cooperation with engineers and programmers.

The very first examples of computer drawings in Soviet Latvia can be found in the late 1960s. Undoubtedly, this had been made possible thanks to the appearance of better output devices and more appropriate printing possibilities for graphics. The political background dictates a completely different approach to the style and aesthetics of this art form. Nobody, including the authors of computer drawings, accepted them as art. They are anonymous, and only a very small part of them has survived till nowadays.

Nevertheless, in the Soviet Union, particularly in Soviet Russia, papers covering the topics of computer aesthetics and computer music appeared in the 1970s and 1980s.

After 1990, new media art (now computer art is often a part of it) began to develop in Latvia. Some Latvian contemporary artists had successful cooperation with our computer scientists and physicists in developing their art projects.

2 Computer Graphics and Computer Art Pioneers

The term "computer art" was coined in BOEING Company by its designer William Fetter (1928-2002) who first used it in 1960 to describe his 3D renditions (computer-generated orthographic view) of aircraft cockpits and pilots.

Benjamin Francis Laposky (1914-2000) is being considered as a pioneer of computer art. Shortly after World War II, he began working with oscilloscopes due to his longstanding interest in geometry, curves and Lissajous figures. At the very beginning, he even did not use a computer in his artworks which he called "Oscillons". We can note his works as some kind of predecessors to the computer art because of the use of algorithmic signals to control artwork producing devices. In 1950, Laposky used a cathode ray oscilloscope with sine wave generators and various other electrical and electronic circuits to create abstract art.

"My work in computer art is a form of oscillography, the results of which I have called 'Oscillons' or 'Electronic Abstractions.' These are composed of combinations of basic electronic wave forms as displayed on a cathode ray oscilloscope and photographed. Colour compositions are achieved by means of special filter arrangements. The resulting art works are presented in photographic exhibitions, kinetic oscilloscope displays, light boxes, or movies.

The relationship of the oscillons to computer art is that the basic waveforms are analogue curves, of the type used in analogue computer systems. I got into oscillographic art through a long-time interest in art or design derived from mathematics and physics. I had worked with geometric design, analytic and other algebraic curves, 'magic line' patterns from magic number arrangements, harmonograph machine tracings, pendulum

patterns, and so on. The oscilloscope seemed to me to be a way of getting a wider variety of similar kinds of design and with controlled effects to produce even newer forms not feasible with previous techniques." [1]

His "Oscillons" was exhibited and published in America and abroad (over 216 exhibitions and 160 publications since 1952). The permanent collection of Laposky's artworks is to be found at the Sanford Museum, the United States.

Herbert W. Franke (1927) produced similar art work in Germany (Erlangen) by creating "Lightforms" (in cooperation with Andreas Hübner, 1953-1955) and "Oscillogramms" (1956). Unexpected coincidence of the time and a method used by Laposky and Franke should be mentioned.

Herbert W. Franke studied physics, mathematics, chemistry, psychology and philosophy in Vienna. He received his Ph.D. in Theoretical Physics in 1950 by writing a dissertation about electron optics. In 1956, he started his experiments with oscilloscope-type images, then – electronic graphics, and in 1969 – with computer art. He actively used the new form of art throughout many years.

- "DRACULA Series", 1970-1971, computer graphics based on "Dragon Curves", a fairly new discovered form of mathematical fractals.
- "Rotations, Projections", 1970-1971, the images are based on interactively controlled motions in a perspective view. This kind of animation sequences was used in the "Laser" ballet which was performed on the experimental stage of the Bavarian State Opera.
- "Colorraster 75", 1975, an edition of the pictures was printed with one of the first inkjet printers available in Europe.
- "Cascade", 1978, live transformation of music into graphics: the underlying program for the Apple II GS converts sounds in relation to the frequencies into pictures.
- "Fourier-Transformations", in cooperation with Horst Helbig, ca. 1979, the first attempts with planar Fourier transformations already showed that it produced a variety of shapes, which were not inferior to fractals.
- "Virtual Sculptures", done with two software programs called "Mathematica" and "Bryce", after 1996. [2]

Franke became a cybernetic aesthetics and computer art lecturer of the University of Munich (1973-1998), and in the Academy of Fine Arts in Munich (1984-1998). In 2008, he was promoted to Senior Fellow by 'Konrad-Zuse-Zentrum für Informationstechnik Berlin' (ZIB). His list of publications includes more than 40 books, especially about art-science connections, and also science fiction novels. His second book, "Computer Graphics – Computer Art" (first edition – Bruckmann, München, 1971, second edition – Springer Verlag, Heidelberg, Berlin, New York, 1985), was the earliest comprehensive text on the subject. The works are in the following collections: Abteiberg Museum collection, ZKM, Kunsthalle Bremen, Germany, Victoria and Albert Museum, the United Kingdom. **A. Michael Noll** (1939), **Frieder Nake** (1938) and **Georg Nees** (1926) came as the 'second generation' of scientists – computer artists. Nake, Nees and Noll are sometimes called the three capital 'N's, generally in the context of history of computer art. Their artworks were often based on originally created algorithms.

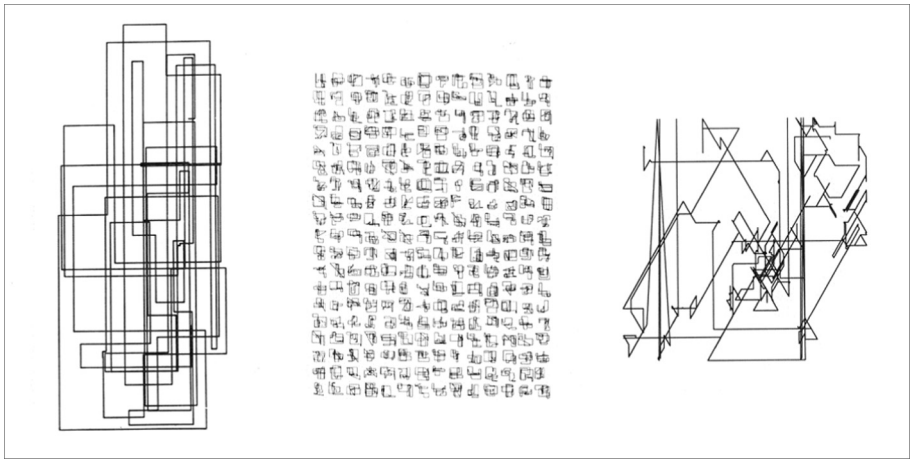


Fig. 1 Random polygons. G. Nees "23 Vertices", 1965; A.M. Noll "Vertical-Horizontal no. 3", 1964; F. Nake "Random polygon", 1965 (© G. Nees, © A. M. Noll, © F. Nake)

Besides being a pioneer in computer art and animation, **A. Michael Noll** has had a varied career as a researcher at Bell Labs in such areas as the effects of media on interpersonal communication, three-dimensional computer graphics and animation, human-machine tactile communication, speech signal processing, cepstrum pitch determination, and aesthetics. He was also a staff member of the White House Science Advisor, AT&T manager and planner, academic professor and administrator, author, columnist, classical music critic, and archivist and biographer.

In the late 1960s and early 1970s, Noll constructed interactive three-dimensional input devices and displays and a three-dimensional, tactile, force-feedback ("feelie") device (US patent 3,919,691 "Tactile Man-Machine Communications System" filed May 26, 1971, issued on November 1, 1975). This device was the forerunner of today's virtual reality systems. He was also one of the first researchers to demonstrate the potential of raster scan displays for computer graphics. He was an early pioneer in the creation of stereoscopic computer-animated movies of four-dimensional hyper-objects, of a computer-generated ballet, and of computer-animated title sequences for TV and film. [3]

He was very enthusiastic about creating computer simulations after Dutch painter Piet Mondrian ("Composition with Lines"), English painter Bridget Riley ("Ninety Parallel Sinusoids With Linearly Increasing Period"), as well as exploring mathematics for artistic purposes ("Gaussian Quadratic", "Hypercube Computer Animation"). Noll was one of the first researchers to use a digital computer to create artistic patterns and to formalize the use of random processes in the creation of visual arts. Noll as artist was active till 1980.

Noll's artworks are in permanent collections of the Museum of Modern Art, the Los Angeles County Museum of Art, the USC Fisher Gallery, the Performing Arts Library at Lincoln Centre, and the Academy of Motion Picture Arts and Sciences, the United States and Victoria and Albert Museum, the United Kingdom.

Frieder Nake is a professor of Computer Graphics at the Department of Computer Science at the University of Bremen. In the early 1960s, his combined interests in probability theory, information aesthetics, and software merged in generating computer art. At first, he used the Graphomat Zuse Z 64 drawing machine to produce four-colour plotter drawings, and later, he worked with computer Standard Electric ER 56. He was active during the 1960s and 1970s working with compositions of abstract geometric lines and shapes. His works are in the collection of the Victoria and Albert Museum, the United Kingdom. **Georg Nees** is a professor at the University of Erlangen, Germany. Nees studied mathematics, physics and philosophy in Erlangen and Stuttgart. In 1969, he received his Ph.D. in "Generative Computer Graphics". His academic advisor was Max Bense (1910–1990), philosopher, mathematician and founder of Information Aesthetics theory. Since 1964, Nees has been working with computer graphics, sculptures and film – both producing and theorizing about it. His first artworks were based on the programming language ALGOL and the Siemens computer system 4004 in conjunction with a tape-controlled plotter 'Zuse-Graphomat'. He is still active as an artist; nevertheless, his artistic style has not changed a lot since the 1960s. The artworks of Georg Nees are held by the Victoria and Albert Museum, the United Kingdom.

In 1965, the computer art by Michael Noll, Frieder Nake and Georg Nees was exhibited at the Howard Wise Gallery in New York, United States, along with random-dot patterns by Bela Julesz.

Béla Julesz (1928–2003) was a visual neuroscientist and experimental psychologist. He was the first creator of random dot stereograms in 1959. Julesz worked at Bell Laboratories on recognizing camouflaged objects from aerial pictures taken by spy planes. At that time, many vision scientists still thought that depth perception occurred in the eye itself, whereas now it is known to be a complex neurological process. Béla Julesz used a computer to create a stereo pair of random-dot images which, when viewed under a stereoscope, caused the brain to see 3D shapes. This proved that depth perception is a neurological process. Later, Christopher Tyler, a former student of Julesz, used the principles of random-dot stereograms to invent autostereograms, which create the same effect using a single image instead of two. [4]

Kenneth C. Knowlton (1931) is to be considered a pioneer of the so-called ASCII (American Standard Code for Information Interchange) art. He received his Ph.D. at M.I.T. in 1962 (thesis: Sentence Parsing with a Self-Organizing Heuristic Program). In 1963, working at Bell Laboratories, Knowlton developed the BEFLIX (Bell Flicks) – the first specialized computer animation programming language for bitmap computer-produced movies, created using an IBM 7094 computer and a Stromberg-Carlson 4020 microfilm recorder. He used it to make experimental film series in cooperation with artists including eight computer-generated animations "Poem Field" together with Stan VanDerBeek (1927–1984).

Knowlton also created new programming languages for graphics – EXPLOR, ATOMS and SPHERES. In 1966, together with Leon Harmon (1922–1982) he developed electronic scanning technologies explored in the "Studies in Perception" series. In "Studies in Perception I", they created an image of a reclining nude (the dancer Deborah Hay) by scanning a photograph with a camera and converting the analogue voltages to binary numbers which were assigned to typographic symbols based on halftone densities. It was printed in The New York Times (1967) and exhibited at one of the

earliest computer art exhibitions – "The Machine as Seen at the End of the Mechanical Age" – held at the Museum of Modern Art, New York (1968). [5] It became popular as an example of ASCII art, although Knowlton called this technique a **'**photomosaic'. He is still active as an artist and employee. His current artworks are sometimes mosaics made of seashells.

Table 1

Computer Art Pioneers (using data from [6])

Name, Surname	***Country***	***Enter***	***Category***	***Comments***
Ben Laposky	United States	1950	Mathematician / artist	Oscilogramms
Herbert Franke	Germany	1956	Mathematician / artist	Oscilogramms, algorithmic art
John Whitney (Senior)	United States	1958	Filmmaker	Made films as an artist in residence with IBM
Charles Csuri	United States	1960	Artist – algorist, computer animated filmmaker	Csuri's film „Hummingbird" purchased by the Museum of Modern Art (1968)
Michael Noll	United States	1963	Computer scientist / artist	Algorithmic art, Riley and Mondrian simulations
Frieder Nake	Germany	1963	Mathematician	Algorithmic art
Kenneth Knowlton	United States	1963	Computer scientist	Cooperation with 1963 (+ Lillian Schwartz), 1964 (+ Stan Vanderbeek), 1966 (+ Leon Harmon), Lillian Schwartz
George Nees	Germany	1969	Mathematician – computer sculpture	Algorithmic art
Manfred Mohr	Germany	1969	Artist – algorist	World's first museum based solo exhibition of computer-generated art at Musee d'Art Modern, Paris (France)
Harold Cohen	United Kingdom, moved to the United States	1972	Artist (abstract painter)	Creator of AI program robot AARON
Yoichiro Kawaguchi	Japan	1975	Artist / animator	Japan Pavilion EXPO ,1986, EXPO ,2010
Roman Verotsko	United States	1982	Artist - algorist	Algorithmic plotter / brush- algorist
David Em	United States	1983	Artist	First significant use of 3D

3 Computer Art Founding Fathers' Vision about the Computer Art Theory

The theoretical papers by computer art pioneers such as Frieder Nake, Ben Laposky, Michael Noll show their broad spectrum of interests – starting from mathematics, computer science, physics, engineer sciences and including art, art theory, esthetics, semiotics, psychology. They came into the field of art theory and aesthetics with fresh, innovative and sometimes naive ideas providing a basis for new discussions, conclusions and making the first attempts to define the aesthetics of the new digital age.

The immediate theoretical approach to their newly-created artworks seems to be extraordinary and different from the professional artists, as well as from art historians. Despite the growing role of curators and art marketing in the 20th century, artists were occasionally not keen on self positioning in the context of concrete art movement and style right after the work had been created. We can find some exceptions (Dada, Futurism came to the art stage with written manifests) but they never tried to publish papers within an academic environment.

In 1975, Ben F. Laposky wrote: "My interest in other kinds of art was to some extent in abstract geometric painting, cubism, synchronism and futurism. The oscillons are related to the newer developments of op art, Lumia (light) art, computer art, abstract motion pictures into an academic environment, video synthesizer (TV) art, and laser displays, such as Laserium... The oscillons are intended to be a form of creative fine art." [7]

Herbert Franke suggests on two aspects regarding to computer graphics – computer art. He states that: "Art can be regarded as a special form of communication. It is the task of the artist to provide a message, which in this particular case is also subject to certain aesthetic considerations, however they may be defined. Formerly, colours, sounds and tones were regarded as the raw material of art; today they would be considered information carriers. The elements of art are data, i.e. immaterial components. Even though this statement may sound rather sober, it does imply that art is not a material but rather an intellectual process." [8] He also draws attention to the fact that having an inventory of all the mathematical branches and an interest in visualization of all forms that come to light, one can obtain plenty of forms, shapes and structures never seen before – an expansion of our treasury of forms. Many of these forms have considerable aesthetic charm. According to the usual criteria we cannot call them original works of art. But they can be considered as elements available for new creations and can be used to develop artworks. [9]

Michael Noll has changed his mind several times regarding the significance of computer graphics and computer art at the stage of the 20th century art history. He has moved from the adoration of a newly-created art form to critics and pessimistic view on its sense and again to positive opinion on the future of this medium. During years 1962–1994, he had had more than ten scientific publications on computer art, digital aesthetics and even an essay on the human perception of computer-generated graphics.

"Composition With Lines" consists of a scattering of vertical and horizontal bars which, at first glance, seem to be randomly scattered throughout the painting. With further study, however, one realizes that Mondrian used considerable planning in placing each bar in proper relationship to all the others. Conceivably, Mondrian followed some

scheme or program in producing the painting although the exact algorithm is unknown. A digital computer and microfilm plotter were used to produce a semi-random picture "Computer Composition With Lines" similar in composition to Piet Mondrian's painting "Composition With Lines" (1917). Reproductions of both pictures were then presented to 100 subjects whose task was to identify the computer picture and to indicate which picture they preferred. Only 28% of the subjects were able to correctly identify the computer-generated picture, while 59% of the subjects preferred the computer-generated picture. Both percentages were statistically different (0.05 level) from selections based upon chance according to a binomial test." [10]

Fig. 2 P. Mondrian "Composition With Lines", 1917; A. Michael Noll "Computer Composition With Lines", 1965 (© Rijkmuseum, © A. Michael Noll)

Frieder Nake insists on: "Since Shannon and Weaver, it was believed that any message (a sequence or field of perceivable signals) contained information. The information content of a message could be measured. A painting could clearly be considered the carrier of signs. It could, indeed, be viewed as a complex sign composed of subsigns which were in turn composed of subsigns and so on. On each level of such a hierarchy the statistical information content (according to Shannon's axiomatic definition) could be determined." [11]

It has to be mentioned that there still exists a gap between contemporary art theory and history, and the computer art theory. Nowadays, when practically everyone, including the artists and art historians, is an everyday user of computers and the term "new media art" is frequently used instead of "computer art", this art form of the 20th century should be introduced to a wider audience.

4 First Examples of Computer Art in Latvia

The emergence of the computer art was tightly connected with the computer industry development. Latvia, a small state of the Baltic region, was incorporated into the Soviet Union after the Second World War. In former Soviet Latvia, the first generation computer

was constructed by a team lead by Jānis Daube in 1962. Some graphical features became available at the end of the 1960s. In 1969, a second generation computer GE-415 was bought from France to set it at the premises of the Computing Centre in Riga. The Museum of Computer Technique, IMCS UL in Latvia possesses the examples of the first computer drawings from that time.

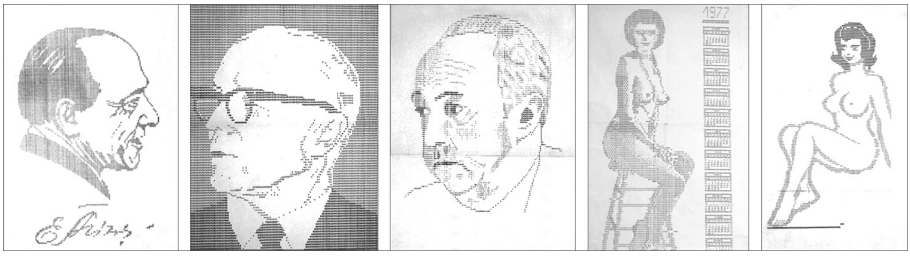


Fig. 3 "ASCII Art Samples", ca. 1970 (©Museum of Computer Technique, IMCS UL)

Due to the political situation of that time, no connections with the Western modern art were officially allowed. The art forms, such as Dadaism, abstract art (and its variations), etc., were practically unknown to a wider audience. The first authors of computer drawings were computer scientists, engineers. They had hardly any information about the contemporary art forms of the 20th century. The most common were ASCII art forms. These drawings always tended to be realistic. Sometimes they even had features characteristic to caricatures or cartoons. Even some 'underground' things like drawings of nudes (no wonder, considering the political status of Latvia at that time) had appeared since 1969. These first examples are not signed at all due to their completely different status in perception of the society, as well as the authors' own position. The computer scientists and engineers were not tended to position their experiments with data visualization as any form of visual art.

Later (during the 1970s), we can find some examples of kinetic art, conceptual and environmental art made by professional artists. Officially, these pieces were stated in the category of design objects. Most of them were never realized. They appeared only a few times in the expositions dedicated to design. Some kinetic objects were built, as well as communication design samples appeared. In 1980, the Riga Central Railway Station's renovation project was implemented. A multi-program light system at the clock tower and visual communication system inside the station building were developed. In nowadays terms, we can call it a synthesis of communication design and conceptual art.

5 Cooperation of Artists and Scientists

According to Andrea Grover [12], the first publicized cooperation of contemporary artists and scientists working on new technologies started in early 1960s. The NASA Art Program was established in 1962 by the United States to commission artists, including Norman Rockwell and Robert Rauschenberg, for the purpose of recording history of

space exploration through the eyes of artists. The collection now includes 2,500 works by more than 350 artists.

In 1964, Bell Laboratories started an informal artist-in-residence program that later evolved into the greatest art and technology programs in the country. It started with the beginnings of computer graphics, such as the above-mentioned Ken Knowlton's and Leon Harmon's computer-generated "Studies in Perception" series and BEFLIX animation system, which was used to produce dozens of animated films together with artists like Stan VanDerBeek and Lillian Schwartz.

In 1966, a group of New York artists worked with around 30 engineers and scientists from the Bell Telephone Laboratories to create performances that incorporated new technology. On October 13–23, 1966, they organized a major artistic event "9 Evenings: Theatre and Engineering". It has always been mentioned as landmark in the 20th century art history, as well as a starting point of using such technologies as video projection, wireless sound transmission, and Doppler sonar. In 1967, engineers Billy Klüver and Fred Waldhauer, and artists Robert Rauschenberg and Robert Whitman launched a non-profit organization *Experiments in Art and Technology (E.A.T.)* with a main goal to develop collaboration between artists and engineers. Composer and sound artist John Cage, dancer Merce Cunningham, and pop artist Andy Warhol have to be mentioned among the E.A.T. most active members. The best-known E.A.T. activity is the Pepsi Pavilion at Expo '70 (1970, Osaka, Japan) where E.A.T. artists and engineers cooperated to design and implement an immersive dome that included a fog sculpture by Fujiko Nakaya. Twenty-eight regional E.A.T. divisions were established throughout the U.S. in the late 1960s to promote cooperation between artists and engineers. They resulted in expanding the role of artists in social developments related to new technologies. E.A.T. activities lasted till 1989.

The Art and Technology artist-in-residence program (A&T) was launched during 1967–1971 by Maurice Tuchman, curator of modern art at the Los Angeles County Museum of Art (LACMA) in Los Angeles. Tuchman selected industry partners from southern California companies from one side, and American, as well as European artists, who were keen to use the provided new technologies, from the other. More than 70 artists had participated, among them such 20th century and contemporary art icons as Andy Warhol (1928–1987), Robert Rauschenberg (1925–2008), Roy Lichtenstein (1923–1997) and John Baldessari (1931).

This program was followed by many other programs for artists in residence. John Whitney, computer animation pioneer, became IBM's first artist in residence (1967). Since 1985 to this day, Australian Network for Art and Technology (ANAT) supports Australian and foreign artists and creative practitioners engaged within science and technology, including residencies. Similar activities took place in Britain. John Latham and Barbara Steveni established *Artist Placement Group (APG)* in 1966. They tried to engage artists in non-art environments.

Computer Technique Group (CTG) emerged in Japan from 1966 till 1969. Their statement was that a computer is "a medium, neither a tool nor device, and computers could become a medium for art and had a potential to become a medium that unites media through its art works and global activities."

Computer Arts Society (C.A.S.) was born almost at the same time in London. It started by an inaugural exhibition "Event One" in March, 1969. George Mallen, Alan

Sutcliffe and John Lansdown set up C.A.S. as a branch of the British Computer Society to facilitate the use of computers by artists. Its bulletin PAGE almost to this day has featured British and international computer artists, and hosted some fundamental discussions such as about the aims, nature and aesthetics of computer art.

Another periodical – LEONARDO – should be mentioned as a longstanding and still flourishing science and art project. This journal has achieved international recognition and high scientific level. Its history is a very personal story at the same time. Frank Malina (1912–1981) was a pre-Space Age rocket engineer and second director of the Jet Propulsion Laboratory (1944–1946). However, he became involved in kinetic art. While being an engineer, he had access to an abundance of scholarly Periodicals. There was no equivalent publication place for artists, so he decided to start one. The concept was simple – a publication by serious artists with subject integrity secured by the same kind of peer review of articles that is common in scientific journals. The journal Leonardo was founded in 1968 in Paris. *Leonardo* was and still is an international peer-reviewed research journal that features articles written by artists on their own work, and focuses on the interactions between the contemporary arts and the sciences and new technologies.

After the death of Frank Malina in 1981, and under the leadership of his son, Roger F. Malina (astrophysicist, Executive Director of the Centre for EUV Astrophysics at UC Berkeley at that time), *Leonardo* moved to San Francisco, California, as the flagship journal of the newly-founded non-profit organization Leonardo/The International Society for the Arts, Sciences and Technology (Leonardo/ISAST). Leonardo/ISAST has grown along with its community and today it is the leading organization for artists, scientists and others interested in the application of contemporary science and technology to the arts and music. [13] Leonardo is covered by Arts & Humanities Citation Index and Current Contents/Arts & Humanities of Thomson Reuters.

In Latvia, the first net art, then new media art, examples spread out in the mid of 1990s. Professional artists were initiators of collaboration between scientists and the IT industry. Annual new media festival in Riga – "Art and Communication" – first organized by RIXC (Centre for New Media Culture in Riga) in 1996 has acquired international recognition amidst worlds digital community. Artists prompted to cover such issues as Transbiotics (2010), Energy (2009), Spectropia (2008), Spectral Ecology (2007), Waves (2006), Media Architecture (2003). In 2007, new media artist Gints Gabrāns (1970) created project "Paramirrors" and presented it in the Latvian pavilion on the 52nd International Art Exhibition of La Biennale di Venezia, Venice, Italy. It was done in collaboration with Elmārs Blūms, Institute of Physics, University of Latvia, Ilze Aulika, Vismants Zauls, Mārtiņš Rutkis, Institute of Solid State Physics, University of Latvia, and Jānis Spīgulis, Institute of Atomic Physics and Spectroscopy, University of Latvia.

References

1. Online: http://www.atariarchives.org/artist/sec6.php (accessed December 2010)
2. Online: http://www.biologie.uni-muenchen.de/~franke/ (accessed December 2010)
3. Online: http://noll.uscannenberg.org/ (accessed December 2010)
4. Julesz, B. Foundations of Cyclopean Perception. Chicago: The University of Chicago Press, 1971.
5. Online: http://www.kenknowlton.com/ (accessed December 2010)

6. King, M. Computers and Modern Art: Digital Art Museum. "Proceedings of the 4th Creativity and Cognition Conference", Loughborough University, New York: ACM Press, pp. 88–94, 2002.
7. Laposky, B. Oscillons: Electronic Abstractions. Available online: http://www.atariarchives.org/artist/sec6.php (accessed December 2010)
8. Franke, H.W. The Expanding Medium: The Future of Computer Art. "Leonardo", vol.20, No.4, pp. 335–338, 1987.
9. Franke, H.W. and Helbig, H.S. Generative Mathematics: Mathematically Described and Calculated Visual Art, "Leonardo", 25, Nos. 3/4, pp. 291–294, 1992.
10. Noll, A.M. Human or Machine: A Subjective Comparison of Piet Mondrian's *'Composition with Lines*' and a Computer-Generated Picture, "The Psychological Record", Vol. 16. No. 1, pp. 1–10, January 1966.
11. Nake, F. Computer Art. A Personal Recollection. "Proceedings of the 5th Conference on Creativity &Amp; Cognition", ACM, New York, NY, USA, pp 54–62, 2005
12. Andrea Grover. Artists in labs, tech industries, or science agencies http://andreagrover.com/category/science/ (accessed December 2010)
13. Online: http://www.leonardo.info/ (accessed December 2010)

LATVIJAS UNIVERSITĀTES RAKSTI
770. sējums. DATORZINĀTNE UN INFORMĀCIJAS TEHNOLOĢIJAS

Latvijas Universitātes Akadēmiskais apgāds
Baznīcas ielā 5, Rīgā, LV-1010
Tālrunis 67034535

Iespiests SIA «Latgales druka»