Charles University in Prague, MFF, Department of Software Engineering
Czech Technical University in Prague, FIT, Dept. of Software Engineering
VŠB–TU Ostrava, FEECS, Department of Computer Science
Czech Society for Cybernetics and Informatics

Proceedings of the Dateso 2015 Workshop

DATESO 2015

Databases, Texts
Specifications, and Objects

`http://www.cs.vsb.cz/dateso/2015/`
`http://www.ceur-ws.org/Vol-1343/`

ČSKI

April 14 – 16, 2015
Nepřívěc u Sobotky, Jičín, Czech Republic

Technical editor:
Pavel Moravec, `pavel.moravec@vsb.cz`
VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Page count:	141
Publication:	479th
Edition:	1st
First published:	2015
ISBN:	978-80-7378-285-6

This proceedings was typeset by PDFLaTeX.
Cover design by Pavel Moravec (`pavel.moravec@vsb.cz`) and Tomáš Skopal.

Published by MATFYZPRESS publishing house of Faculty of Mathematics and Physics Charles University, Sokolovská 83, 186 75 Praha 8 – Karlín, Czech Republic as its 479th publication.

Preface

DATESO 2015, the international workshop on current trends on Databases, Information Retrieval, Algebraic Specification and Object Oriented Programming, was held on April 14 – 16, 2015 in Nepřívěc u Sobotky, Jičín, Czech Republic.

The 15th year was organized by Department of Software Engineering MFF UK Praha, Department of Software Engineering, FIT ČVUT Praha, Department of Computer Science VŠB-Technical University Ostrava, and Working group on Computer Science and Society of Czech Society for Cybernetics and Informatics. The DATESO workshops aim for strengthening connections between these various areas of informatics.

The proceedings of DATESO 2015 are also available at DATESO Web site: `http://www.cs.vsb.cz/dateso/2015/` and CEUR Workshop Proceeding site: `http://www.ceur-ws.org/Vol-1343/` (ISSN 1613-0073). The Program Committee selected 12 papers (8 full and 4 short papers) from 19 submissions, based on two independent reviews.

We wish to express our sincere thanks to all the authors who submitted papers, the members of the Program Committee, who reviewed them on the basis of originality, technical quality, and presentation. We are also thankful to the Organizing Committee. Special thanks belong to Czech Society for Cybernetics and Informatics.

Our thanks go also to Pavel Moravec who, as copy editor of DATESO Proceedings, helped to prepare this volume and provided technical support for the conference preparation portal.

April, 2015 M. Nečaský, J. Pokorný, P. Moravec (Eds.)

Table of Contents

Dynamic Local Scheduling of Multiple DAGs in Distributed Heterogeneous Systems

Ondřej Votava, Peter Macejko, and Jan Janeček

Czech Technical University in Prague
{votavon1, macejp1, janecek}@fel.cvut.cz
http://cs.fel.cvut.cz

Abstract. Heterogeneous computational platform offers a great ratio between the computational power and the price of the system. Static and dynamic scheduling methods offer a good way of how to use these systems efficiently and therefore many algorithms were proposed in the literature in past years. The aim of this article is to present the dynamic (on-line) algorithm which schedules multiple DAG applications without any central node and the schedule is created only with the knowledge of node's network neigbourhood. The algorithm achieves great level of fairness for more DAGs and total computation time is close to the standard and well known competitors.

1 Introduction

Homogeneous (heterogeneous) computational platform consists of a set of identical (different) computers connected by a high speed communication network [1, 2]. Research has been done last few years on how to use these platforms efficiently [3–5]. It is believed that scheduling is a good way on how to use the computation capacity these systems offer [1, 6]. Traditional attitude is to prepare the schedule before the computation begins [7, 8]. This requires information about the network topology and parameters and also node's computational abilities. Knowing all of this information we can use the static (offline) scheduling algorithm. Finding the optimal value of *makespan* – i.e. the time of the computation in total – is claimed to be NP complete [9, 10]. Therefore research has been done and many heuristics have been found [11, 12].

Compared to static scheduling, dynamic (online) scheduling allows us to create the schedule as part of the computation process. This allows dynamic algorithms to use the feedback of the system and modify the schedule in accordance with current state of the system. Dynamic algorithms are often used for scheduling multiple applications [13–16] at the same time and therefore fairness of generated schedules is important.

The algorithm presented in this paper does not require the global knowledge of the network, it uses the information gathered from node's neighbors only. The phase of creating the schedule is fixed part of the computation cycle. The algorithm is intended to be used for scheduling more DAGs simultaneously and tries to achieve fair division of tasks for all computing nodes.

The structure of this article, which is the enhanced version of [17], is as follows, in the section two we describe the problem of scheduling and make a brief summary of related work. In the section three we describe the algorithm itself and in the following section we describe the testing environment and results we obtained by running several simulations. In the fifth section we conclude the results from section four, show the pros and cons of the presented algorithm and discuss the future improvements.

2 Problem definition

The application model can be described as a directed acyclic graph $AM = (\mathbf{V}, \mathbf{E}, \mathbf{B}, \mathbf{C}$ [12, 18], where:

V $= \{v_1, v_2, \dots, v_v\}, |\mathbf{V}| = v$ is the set of tasks, task $v_i \in \mathbf{V}$ represents the piece of code that has to be executed sequentially on the same machine,

E $= \{e_1, e_2, \dots, e_e\}, |\mathbf{E}| = e$ is the set of edges, the edge $e_j = (v_k, v_l)$ represents data dependencies, i.e. the task v_l cannot start the computation until the data from task v_k has been received, task v_k is called the parent of v_l, v_l is called the child of v_k,

B $= \{b_1, b_2, \dots, b_v\}, |\mathbf{B}| = v$ is the set of computation costs (e.g. number of instructions), where $b_i \in \mathbf{B}$ is the computation cost for the task v_i,

C $= \{c_1, c_2, \dots, c_e\}, |\mathbf{C}| = e$ is the set of data dependency costs, where $c_j = c_{k,l}$ is the data dependency cost (e.g. amount of data) corresponding to the edge $e_j = (v_k, v_l)$.

The task which has no parents or children is called entry or exit task respectively. If there are more than one entry/exit tasks in the graph a new virtual entry/exit task can be added to the graph. Such a task would have zero weight and would be connected by zero weight edges to the real entry/exit tasks.

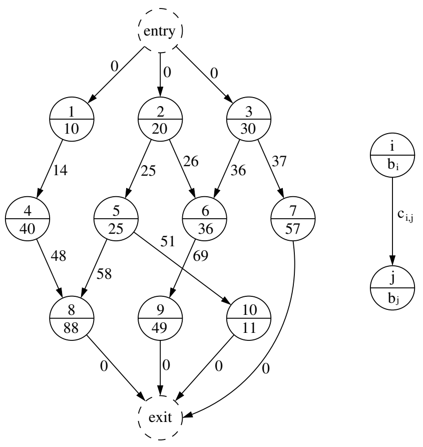


Fig. 1. Application can be described using DAG.

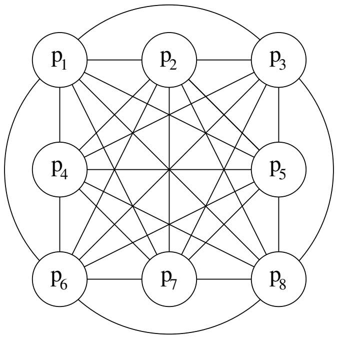


Fig. 2. The computation system can be represented as a general graph.

Since the application can be described as DAG we use terms *application*, *DAG application* or simply *DAG* as synonyms in this paper.

The computation system consist of a set of computing units all of which are connected by a high speed communication network. It can be described using a general graph $CS = (\mathbf{P}, \mathbf{Q}, \mathbb{R}, \mathbb{S})$, where:

$\mathbf{P}$ $= \{p_1, p_2, \ldots, p_p\}, |\mathbf{P}| = p$ is a set of the computers,
$\mathbf{Q}$ $= \{q_1, q_2, \ldots, q_p\}, |\mathbf{Q}| = p$ is the set of speeds of computers, where q_i is the speed of computer p_i,
$\mathbb{R}$ is a matrix describing the communication costs, the size of $\mathbb{R}$ is $p \times p$,
$\mathbb{S}$ is a matrix used to describe the communication startup costs, it is usually one dimensional, i.e. it's size is $p \times 1$.

2.1 Merging application and computation model

Since the structure of the application and the characteristics of the computation systems are known it is no problem to get the information about computation time of each application's node at any computation node. This information is stored in a $\mathbb{W}$ matrix, whose dimensions are $v \times p$, where the item at the position $[i, j]$ contains the information about the length of the computation of the task i on a computation unit j. The value of $\mathbb{W}[i, j]$ is computed by the following equation

$$\mathbb{W}[i, j] = \frac{b_i}{q_j}, \tag{1}$$

where b_i is the computation cost of task v_i and q_j is the speed of computer p_j (e.g. instructions per second).

The total communication time for a message m (corresponding data dependency for the edge (v_k, v_l)) that is send from the computer i to the computer j can be computed by this equation

$$c_m = \mathbb{S}[i] + \mathbb{R}[i][j] \cdot c_{k,l}. \tag{2}$$

2.2 Our model description

The content of the matrix $\mathbb{W}$ is dependent on the properties of computation nodes. However, the computers differ only in a certain ways. The "fast" computers are k times faster than "slow" computers. On that account the columns in the $\mathbb{W}$ are usually only multiples of one column. This information can be reduced to the constant k_p for each processor p. When the computation system is allowed to change, the matrix $\mathbb{W}$ is not useable either since it does not reflect any dynamic behaviour.

The structure, we decided to use, can be described as follows. We selected one processor to serve as a reference – p_{ref}. The computation time of one instruction on this processor lasts one time unit. The speedup of the processor p_i is then defined as

$$SU_p(i) = \frac{q_i}{q_{ref}}, \tag{3}$$

where q_i is the speed of processor p_i and q_{ref} is the speed of the reference processor.

The time duration of computation of a task v_j on the processor p_i is then computed "on the fly" by the equation

$$time_{j,i} = \frac{b_j}{SU_p(i)} \; . \tag{4}$$

Finally, the computation platform is described as a set of speedups and the communication matrices and the merging of application and computation model is being done as a part of the computation. Even the communication matrices may be reduced in our model. They contain only information about computation node's neighbours and differ for all nodes. Still, this is a problem for implementation part and does not affect the description model, as the "neighbour's" matrices are only a part of "global" communication matrices.

2.3 Related work

Task scheduling or task mapping has been in active research for a long time. Several static algorithms were introduced and dynamic algorithms were published too. Most of static scheduling algorithms are designed to work with one DAG. List scheduling algorithms are very popular and they are often used. HEFT [19] is a simple and effective algorithm used as a reference in our article. HEFT creates a list of tasks sorted by an upward rank[1] and then it assigns tasks to the processor so that the execution time of the task is minimized. Another algorithm presented in [19] is Critical Path On a Processor (CPOP). This algorithm is more complex and optimizes tasks on a critical path. By modifying list algorithms and allowing the execution of tasks more than once tasks duplication algorithms were introduces. HCPFD [2] compared to the HEFT obtains better makespan in most cases. Task duplication achieves surprisingly good results when applied to the computation model containing multi core computers [18].

The way of computing multiple DAGs is usually presented in dynamic algorithms. In [20] there was introduced a static method how to schedule multiple DAGs and the aim was not only to optimize makespan but also to achieve the fair sharing of resources for the competing DAGs. The idea of generating a new graph by appending whole DAGs to the current one is used in [21]. Compared to [20] this algorithm is dynamic, i.e. the graph is build when a new DAG arrives to the system.

Truly dynamic algorithm is described in [14]. This algorithm divides the nodes into two groups. The first one contains nodes used for computation, the second one contains scheduling nodes. Scheduling nodes are independent and the knowledge about the activity of other scheduling nodes is received through the statistics of usage of the computing nodes. The quality of such scheduling is then dependent on the quality of statistics created by computation nodes.

[1] See [19] for details

Unlike the previous one the algorithm presented in [16] is based on one central scheduling unit. The algorithm takes into account the time for scheduling and dispatching and focuses on reliability costs. Another model of completely distributed algorithm is presented in [15]. This algorithm divides nodes into groups and uses two levels of scheduling. The high level decides which group to use and low level decides which node in the group to use.

The algorithm described in [22] works a bit different way. The node works with it's neighborhood and the distribution of task of parallel application is done according to the load of the neighbours. If the load of a node is too high, the algorithm allows the task to be migrated among the network. Genetic programming technique is used to decide where to migrate the task.

The problem of task scheduling is loosely coupled with the network throughput. The description of network used in this paper is not very close to the reality and the problems connected to bottle necks or varying delay may cause problems. The behaviour of task scheduling applications running in the network, which has different parameters, is very well described in [23]. According to this article we expect there are no bottle necks in the networks.

3 Proposed algorithm

In this section we present the algorithm Dynamic Local Multiple DAG (DLMDAG). The algorithm itself, described in Algorithm 1, is a dynamic task scheduling algorithm that supports both homogeneous and heterogeneous computation platforms.

The main idea of the algorithm is based on the assumption that the communication lasts only very short time compared to the computation (at least in one order of magnitude). The computation node, which is the creator of a schedule for a certain DAG application, sends a message to all of its neighbours where it asks how long would the computation of these tasks last if they were computed by the neighbour. Then it continues computing the task and during this computation replies for the question arrive. According to the data (timestamps) received, the node makes a schedule for the set of tasks (asked in previous step), then it sends a message to it's neighbours with the information about who should compute which task and generates another question about the computation time for the next set of tasks.

The algorithm description (Algorithm 1) uses these terms. The task is called "ready" when all of its data dependencies are fulfilled. Ready tasks are stored in a *tasksReady* priority queue. The criterion for ordering is the time computed by *computePriority*. The task that is ready and is also scheduled should be stored in a *computableTasks* queue. Each task's representation contains one priority queue for storing the pair information about finish time and neighbour at which the finish time would be achieved. The queue is ordered by the time.

computePriority method is used to make the timestamps for the tasks. It is computed when a DAG application comes to the computation node (p_k) and it

Algorithm 1 The core

```
neighbours[], readyTasks {priority queue of tasks ready to compute}
computableTasks {queue of scheduled tasks for computing}
if not initialized then
    neighbours = findNeighbours()
    initialized = true
end if
if received DAG then
    computePriority(DAG) {Priority of tasks by traversing DAG}
    push(findReadyTasks(DAG), readyTasks)
end if
if received DATA then
    correctDependecies(DATA)
    push(findReadyTasks(DATA.DAG), readyTasks)
end if
if received TASK then
    push(TASK, computableTasks)
end if
if received REQUEST then
    for all task ∈ REQUEST do
        task.time = howLong(task) {time for task + time for tasks in computableTasks}
    end for
    send(REQUEST-REPLY, owner)
end if
if received REQUEST-REPLY then
    for all task ∈ REQUEST-REPLY do
        push((task.time, sender), task.orderingQueue)
    end for
end if
loop {The main loop of algorithm}
    schedule = createSchedule(tasksReady) {Creates schedule and removes tasks from queue}
    for all (task, proc) ∈ schedule do
        send(task, proc)
    end for
    for all n ∈ neighbours do
        send(REQUEST, n) {tasks from tasksReady}
    end for
    TASK = pop(computableTasks)
    compute(TASK)
    send(DATA, TASK.owner) {Nothing is send if local task}
end loop
```

is generated according to this equation

$$priority(v_j) = time_{j,k} + \max_{\forall i \in par_{v_j}} priority(i), \tag{5}$$

where par_{v_j} is the set of parents of node v_j and $priority(v_0) = time_{0,k}$.

The final scheduling is based on the priority queue *task.orderingQueue*. The scheduling step described in Algorithm 2 is close to HEFT [19]. One big difference is that our algorithm uses the reduced list of tasks[2] and is forced to use all neighbours[3] even if it would be slower than computing at local site.

The algorithm is called local. It is because it uses only information about the node's local neighborhood. Each node creates a set of neighbours in the initialization stage of the algorithm. Therefore there are no matrices $\mathbb{R}$ and $\mathbb{S}$ or there are these matrices but they are different for each computational node.

[2] Only ready tasks are scheduled

[3] If there are not enough ready tasks then not all neighbours are used.

Algorithm 2 Scheduling phase

```
schedule {empty set for pairs}
num = min(|tasksReady|, |neighbours|)
for i = 0; i < num; i + + do
    task = pop(tasksReady)
    proc = pop(task.orderingQueue)
    push((task, proc),schedule)
    removeFrom(proc, tasksReady) {Once neighbour used it cannot be scheduled again}
end for
return schedule
```

The size of matrix $\mathbb{R}$ for the computational node p_i is $\mathbb{R}_{p_i} = (s_i \times s_i)$ where $s = |neighbours_i|$ is the amount of neighbours of the node p_i.

3.1 Time complexity

The time complexity of the algorithm can be divided into two parts. The computation part is connected to the sorting and scheduling phase of the algorithm and the communication part is connected to the necessity of exchanging messages for the scheduling phase. The DAG consists of v tasks and the computation node has s neighbours. One question message is sent about every task to all of the neighbours. Question contains information from one to s tasks and the precise number is dependent on the structure of the DAG. The node which receives the question message always sends a reply to it. As the node finishes the scheduling phase of the algorithm another message with a schedule is sent to every neighbour who is involved in the schedule. The last message (schedule information) can be put together with the question's one and there is from $\mathbf{3\,v/s}$ to $\mathbf{3\,v}$ messages sent in total.

Computation part is based on the sorting operations of the algorithm. There are two types of priority queues being used all of which are based on the heap. The first one is the *tasksReady*. Every task from a DAG is put once in this queue and the time complexity is $O(v \log v)$. The second priority queue (*task.orderingQueue*) stores one piece of information for every neighbour. The queue is used for every task and for every neighbour and the time complexity obtained by this queue is $O(v\,s \log s)$ and therefore the time complexity of the computational part of the algorithm is $\mathbf{O(v \log v + v\,s \log s)}$.

4 Performance and comparison

The algorithm was implemented in a simulation environment [24] and it was executed several times for different scenarios. Makespan, unfairness and average utilization of computing nodes were measured.

Makespan is the total computation time of the application, it is defined as

$$makespan(DAG) = finishTime(v_l) - startTime(v_s), \qquad (6)$$

where $finishTime(v_l)$ is the time when the last task of DAG was computed and $startTime(v_s)$ is the time when the first task of DAG began the computation.

Since several DAG applications compete for the shared resources the execution time for each DAG is longer compared to the execution time when there was the only one application in the system. The slowdown of the DAG represents ratio of the execution time when only one DAG was in system and when there were more in the system. It is described as

$$Slowdown(DAG) = T_{shared}(DAG)/T_{single}(DAG), \tag{7}$$

where T_{shared} is the execution time when more than one DAG was scheduled and T_{single} is the execution time when there was only this DAG scheduled. The schedule is fair when all of the DAGs achieve almost the same slowdown[20] and the schedule is unfair when there are big differences in the slowdown of DAGs. The unfairness for the schedule S for a set of n DAGs $A = \{DAG_1, DAG_2, ..., DAG_n\}$ is defined

$$Unfairness(S) = \sum_{\forall d \in A} |Slowdown(d) - AvgSlowdown|, \tag{8}$$

where average slowdown is defined as

$$AvgSlowdown = \frac{1}{n} \sum_{\forall d \in A} Slowdown(d) \tag{9}$$

The utilization of a computation unit p_j for the schedule S is computed by this equation:

$$Util_S(p_j) = \sum_{\forall t \in tasks_S} makespan(t)/totalTime_j, \tag{10}$$

where $tasks_S$ is a set of tasks which were computed on a p_j in the schedule S and $totalTime_j$ is the total time of the simulation, which is the time when the last task of all DAGs has finished.

Average utilization for the whole set of processors $\mathbf{P}$ and for the schedule S is then defined as

$$AvgUtil_S(\mathbf{P}) = \frac{1}{p} \sum_{i=1}^{p} Util_S(p_i). \tag{11}$$

4.1 Testing environment

Three computation platforms containing 5, 10 and 20 computers were created. A full mesh with different communication speed for several lines was chosen as a connection network – this created a network without bottle necks and allowed the algorithm obtain minimal makespan time [23].

Nodes were divided into groups of 5 and the group used a gigabit connection with a delay of 2 ms. In the network with ten nodes the groups were connected by 100 MBit lines and in the network with 20 nodes the groups were connected as follows:

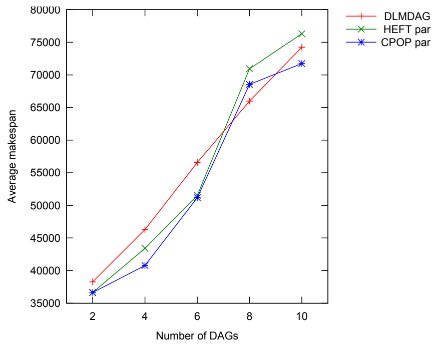


Fig. 3. Makespan for different number of DAGs running concurrently (all platforms)

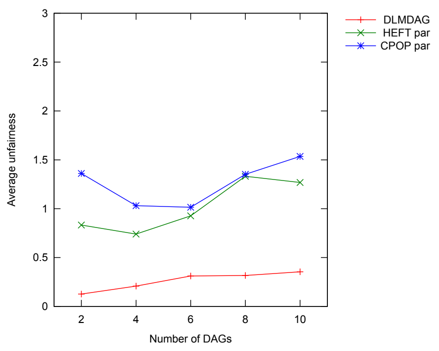


Fig. 4. Unfairness for different concurrently running DAGs (all platforms)

- 4 groups of 5 nodes intraconnected by gigabit,
- 2 groups connected by 100 MBit,
- 3^{rd} and 4^{th} group connected by 10 MBit with others.

Sets of 2, 4, 6, 8 and 10 applications were generated using the method described in [19]. The application contained 25 tasks with different computation and data dependency costs. The schedule for each of the set was generated by simulation[4] of DLMDAG algorithm and by static algorithms HEFT and CPOP.

We used two methods of connecting several DAGs into one for the static algorithms, the first one is sequence execution of DAGs in a row, the second one is to generate virtual start and end nodes and connect DAGs to these nodes with a zero weighted edges. DAGs were ordered in the sequential execution test by the rule the shorter the makespan of DAG is the sooner it is executed. In total there were 100 sets of 2 DAGs, 100 sets of 4 DAGs etc. and the results we obtained we averaged. For the DLMDAG all DAGs arrived to the system at time 0 and on the one node.

4.2 Results

Results of sequential execution of DAGs for HEFT and CPOP achieved much longer makespans and therefore were not included into graphs. HEFT par and CPOP par mean that connection of DAGs was created using virtual start and end tasks.

The makespan achieved by DLMDAG is very close to the HEFT and CPOP (fig. 3). The differences after averaging were just units of percents. The special case was the architecture of five computers (fig. 8), in this case DLMDAG outperforms the others. When there were 10 or 20 computers in the system (fig. 6), DLMDAG achieved slightly worse results. Since HEFT and CPOP use the whole

[4] Simulation tool OMNeT++[24] was used

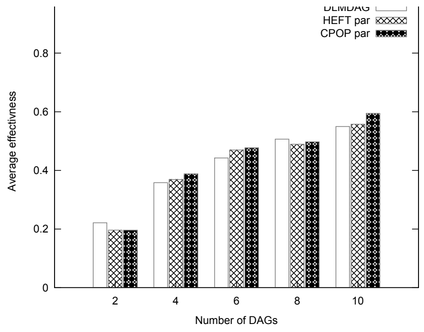


Fig. 5. Utilization for different number of DAGs running concurrently (all platforms)

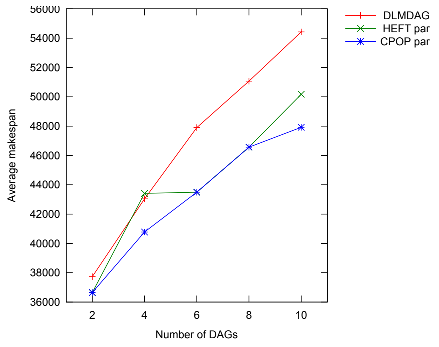


Fig. 6. Makespan for different numbers of applications (20 PC platform)

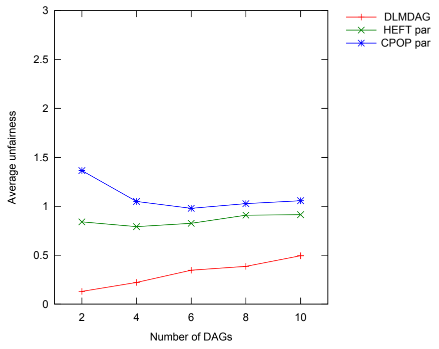


Fig. 7. Unfairness for different numbers of applications (20 PC platform)

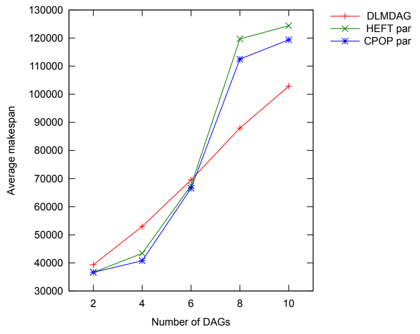


Fig. 8. Makespan for diferent numbers of applications (5 PC platform)

structure of applications DLMDAG works only with ready tasks and therefore it may be unable to use the platform with more devices so efficiently. These results are then dependent on the structure of the applications that were scheduled. The more parallel application is, the better results DLMDAG obtains.

The unfairness (figures 4, 7) for DLMDAG is at the very low level and the growth of it is slow. The unfairness level obtained by HEFT and CPOP in comparison with DLMDAG is worse.

The utilization of nodes (figure 5) corresponds to the makespan achieved by the algorithms. Growing the amount of DAGs in the system the utilization of nodes increases for all algorithms. As mentioned earlier, DLMDAG achieves high level of parallelization and therefore the average utilization of all nodes is also increasing.

5 Conclusion

The algorithm presented in this article is dynamic, it does not use any central point for scheduling neither it requires the information about the whole network.

DLMDAG is based on the local knowledge of the network – only neighbours create the schedule – and the schedule is created using several messages by which the computation times are gathered on the scheduling node. The simulations of the algorithm were executed and results obtained were compared to the traditional offline scheduling algorithms.

DLMDAG is able to use the computation resources in a better way than compared algorithms when there are more tasks in the system than computation units. As the number of computation nodes increases the result DLMDAG achieves become worse than competitor's.

Future work There are several possibilities to improve the proposed algorithm. Initially the computation systems do change. The algorithm should be able to modify the schedules to reflect the network changes. Subsequently the current algorithm is fixed to the scheduling node and it's neighbours and this may cause performance problems, the algorithm could be able to move the application to some other node with different neighbours.

References

1. D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Job Scheduling Strategies for Parallel Processing* (D. Feitelson and L. Rudolph, eds.), vol. 1291 of *Lecture Notes in Computer Science*, pp. 1–34, Springer Berlin / Heidelberg, 1997. 10.1007/3-540-63574-2_14.
2. T. Hagras and J. Janecek, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," *Parallel Computing*, vol. 31, no. 7, pp. 653 – 670, 2005. Heterogeneous Computing.
3. H. Kikuchi, R. Kalia, A. Nakano, P. Vashishta, H. Iyetomi, S. Ogata, T. Kouno, F. Shimojo, K. Tsuruta, and S. Saini, "Collaborative simulation grid: Multiscale quantum-mechanical/classical atomistic simulations on distributed pc clusters in the us and japan," in *Supercomputing, ACM/IEEE 2002 Conference*, p. 63
4. D. Kehagias, M. Grivas, G. Pantziou, and M. Apostoli, "A wildly dynamic grid-like cluster utilizing idle time of common pc," in *Telecommunications in Modern Satellite, Cable and Broadcasting Services, 2007. TELSIKS 2007. 8th International Conference on*, pp. 36 –39, sept. 2007.
5. A. Wakatani, "Parallel vq compression using pnn algorithm for pc grid system," *Telecommunication Systems*, vol. 37, pp. 127–135, 2008.
6. M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," in *In Encyclopedia of Electrical and Electronics Engineering*, pp. 679–690, John Wiley, 1999.
7. Y. kwong Kwok and I. Ahmad, "Benchmarking the task graph scheduling algorithms," in *In Proc. IPPS/SPDP*, pp. 531–537, 1998.
8. J. Liou and M. Palis, "A comparison of general approaches to multiprocessor scheduling," *Parallel Processing Symposium, International*, vol. 0, p. 152, 1997.
9. J. Ullman, "Np-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, no. 3, pp. 384 – 393, 1975.
10. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1979.

11. T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810 – 837, 2001.
12. H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 260–274, 2002.
13. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," *Heterogeneous Computing Workshop*, vol. 0, p. 30, 1999.
14. M. Iverson and F. Ozguner, "Dynamic, competitive scheduling of multiple dags in a distributed heterogeneous environment," *Heterogeneous Computing Workshop*, vol. 0, p. 70, 1998.
15. M. A. Iverson and F. Özgüner, "Hierarchical, competitive scheduling of multiple dags in a dynamic heterogeneous environment," *Distributed Systems Engineering*, vol. 6, no. 3, p. 112, 1999.
16. X. Qin and H. Jiang, "Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems," *Parallel Processing, International Conference on*, vol. 0, p. 0113, 2001.
17. O. Votava, P. Macejko, J. Kubr, and J. Janeček, "Dynamic Local Scheduling of Multiple DAGs in a Distributed Heterogeneous Systems," in *Proceedings of the 2011 International Conference on Telecommunication Systems Management*, (Dallas, TX), pp. 171–178, American Telecommunications Systems Management Association Inc., 2011.
18. J. Janeček, P. Macejko, and T. M. G. Hagras, "Task scheduling for clustered heterogeneous systems," in *IASTED International Conference - Parallel and Distributed Computing and Networks (PDCN 2009)* (M. Hamza, ed.), pp. 115–120, February 2009. ISBN: 978-0-88986-783-3, ISBN (CD): 978-0-88986-784-0.
19. H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task scheduling algorithms for heterogeneous processors," in *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*, pp. 3 –14, 1999.
20. H. Zhao and R. Sakellariou, "Scheduling multiple dags onto heterogeneous systems," *Parallel and Distributed Processing Symposium, International*, 2006.
21. J. Barbosa and B. Moreira, "Dynamic job scheduling on heterogeneous clusters," in *Parallel and Distributed Computing, 2009. ISPDC '09. Eighth International Symposium on*, pp. 3 –10, 302009-july4 2009.
22. R. de Mello, J. Andrade Filho, L. Senger, and L. Yang, "Grid job scheduling using route with genetic algorithm support," *Telecommunication Systems*, vol. 38, pp. 147–160, 2008. 10.1007/s11235-008-9101-5.
23. Y. Kitatsuji, K. Yamazaki, H. Koide, M. Tsuru, and Y. Oie, "Influence of network characteristics on application performance in a grid environment," *Telecommunication Systems*, vol. 30, pp. 99–121, 2005. 10.1007/s11235-005-4320-5.
24. A. Varga *et al.*, "The omnet++ discrete event simulation system," in *Proceedings of the European simulation multiconference (ESM'2001)*, vol. 9, p. 65, sn, 2001.

Data Structures for Indexing Triple Table*

Roman Meca, Michal Krátký, Peter Chovanec, and Filip Křižka

Department of Computer Science, VŠB – Technical University of Ostrava
Czech Republic
{roman.meca.st, michal.kratky, peter.chovanec, filip.krizka}@vsb.cz

Abstract. Semantic-based approaches are relatively new technologies. Some of these technologies are supported by specifications of W3 Consortium, i.e. RDF, SPARQL and so on. There are many areas where semantic data can be utilized, e.g. social networks, annotation of protein sequences etc. From the physical database design point of view, several index data structures are utilized to handle this data. In many cases, the well-known B-tree is used as a basic index supporting some operations. Since the semantic data are multidimensional, a common way is to use a number of B-trees to index the data. In this article, we review other index data structures; we show that we can create only one index when we utilize a multidimensional data structure like the R-tree. We compare a performance of the B-tree indices with the R-tree and some its variants. Our experiments are performed over a huge semantic database, we show advantages and disadvantages of these data structures.

1 Introduction

Semantic-based approaches are new technologies trying to allow computers to handle semantic information. These approaches have many specific applications. The main advantage of a semantic system is that the computer can reveal unexpected facts, e.g. a new effective drug combination in medicine [6], unexpected relationships in social networks [19] or artificial intelligence [28] can be discovered. This is the reason why the semantic systems are a current research topic.

The W3 Consortium have released some specifications related to semantic technologies[1], e.g. RDF [32] as a model of the semantic data or SPARQL [25] as a query language for the RDF data. In addition, more general specifications are also usable for a semantic DBMS supporting SPARQL or another query language, e.g. XML [9] or WSDL [12] for a communication with the DBMS.

In this article, we also list a lot of semantic DBMS with the query languages they support and the indices they utilize. Since the semantic DBMS often utilize a relational DBMS as a storage for the RDF triple table (representing the RDF data), the B-tree [13] is often used as the main index. The main issue of this

* This work is partially supported by SGS, VŠB – Technical University of Ostrava, No. SP2015/192 and No. SP2015/170, Czech Republic.

** This work is an extented version of an article accepted at the DATESO workshop.

[1] `http://www.w3.org/standards/techs/rdf#w3c_all`

physical implementation is that a number of B-trees have to be built to support queries over the triple table. However, there are other index data structures capable to handle semantic data. In this article, we show that it is possible to create only one index if we utilize a multidimensional data structure like the R-tree [22] or some of its variants (namely the Signature R-tree [30] or the Ordered R-tree [31]).

In Section 2, we present some basic terms related to semantic technologies. In Section 3, we describe the basic physical design for the RDF data. Section 4 describes some negative issues of the B-tree as an index data structure for the triple table. In addition, the R-tree, R*-tree [8], and two their variants are described. In Section 5, we summarize the advantages and disadvantages of these data structures for various queries over the LUBM data collection [21]. Finally, we conclude the article and outline the possibilities of our future work.

2 Semantic Technologies

In this section, we briefly introduce a theoretical basis of the RDF model [32] and the SPARQL query language [25] standardized by W3C. We recommend the book [20] for a more detailed review.

2.1 RDF Model

RDF (*Resource Description Framework*) is a general model representing information on the Web; data are modeled as a directed labeled graph [32]. Each edge represents a relationship between an *object* and a *subject*: two nodes of the graph. The label of the edge is called *property.* An example of the graph is given in Figure 1. This tuple (subject, property, object) is called an *RDF triple* (s,p,o).

The values of each triple usually include IRI (Internationalized Resource Identifiers) [15] identifying an abstract or a physical resource. In [20], the author introduces the following definition:

Definition 1 (RDF triple). *Let us assume there are pairwise disjoint infinite sets I, B, and L, where I represents the set of IRIs, B the set of blank nodes, and L the set of literals. We call a triple* $(s, p, o) \in (I \cup B)I(I \cup B \cup L)$ *an RDF triple, where s represents the subject, p the predicate, and o the object of the RDF triple.*

A *triple table* is a set of RDF triples; it is a representation of the RDF graph. In Table 1, we see a fragment of the triple table to the RDF graph in Figure 1. A triple store or an RDF database is an engine enabling to store an RDF graph and efficient processing of queries. However, we usually require other operations like update, insert or delete.

Some RDF stores add a fourth element to the triple; this fourth element contains the context of the triple [14]. There are RDF engines enabling to manage these quads [16].

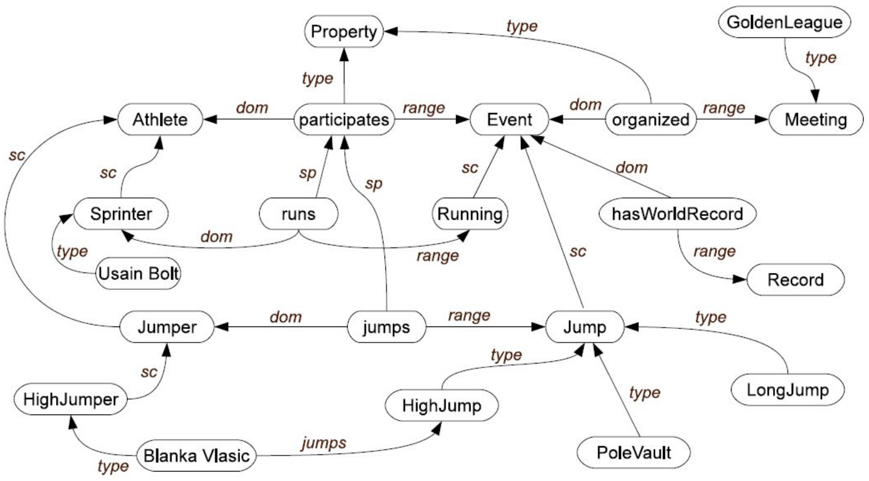


Fig. 1. An example of an RDF graph [18]

Subject	**Property**	**Object**
LongJump	type	Jump
Blanka Vlasic	jumps	HighJump
GoldenLeague	type	Meeting

Table 1. A fragment of an RDF triple table [18]

The RDF specification [32] does not define any way how to store and index the triple table; therefore, there are many variants of the physical design of the triple table and we describe them in Section 3.

2.2 SPARQL Query Language

Although there are many query languages for RDF data[2], e.g. SPARQL/Update (or SPALUR) [38], SPARQL 1.1 [25] is a de-facto standard query language for RDF data. It is similar to SQL in many features. SPARQL 1.1 also includes insert, update, and delete operations.

The basic query construct of the SELECT statement includes `SELECT <projection> WHERE <sequence of triple patterns>`. A variable in SPARQL defined by the symbol ? and a name represents the main difference compared to SQL; they define unknown values of o, s or p in a pattern as well as a relationship among triple patterns. We distinguish four types of the SPARQL query (for more details see [25]):

– *SELECT* – returns the result relation defined by the projection and patterns.

[2] `http://www.w3.org/2001/11/13-RDF-Query-Rules/`

– *ASK* – similar to the SELECT query; however, it returns the boolean value; *true* if the result is not empty, otherwise *false.*
– *CONSTRUCT* – allows to format own result graph over the triples returned by the patterns.
– *DESCRIBE* – returns the node (and its neighbours) defined by the patterns.

A form of `<pattern>` determines the selectivity of a query over the triple table. We can distinguish a point query (s, p, o) returning 0 or 1 triple, or a range query where the query $(s, *, *)$ can returns more triples than the query $(s, p, *)$.

Example 1 (SPARQL Queries).

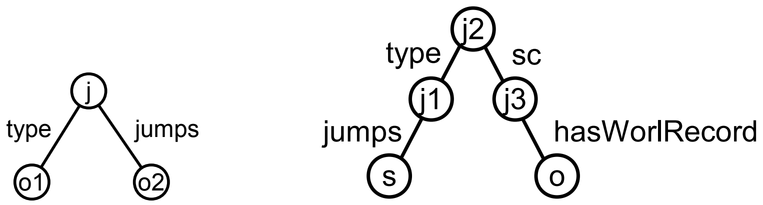
1. `SELECT ?s ?p ?o WHERE { ?s ?p ?o }`
   This query selects the whole triple table, it represents the range query $(*, *, *)$.
2. `SELECT * WHERE { <Blanka Vlasic> <jumps> <HighJump> }`
   `ASK { <Blanka Vlasic> <jumps> <HighJump> }`
   These two queries are similar; the SELECT query returns 0 or 1 triple, on the other hand, the ASK query returns true in the case the triple exists in the graph. These queries represent the point (s, p, o) query over the triple table.
3. `SELECT ?s WHERE { ?s <type> <Jump> }`
   `ASK { ?s <type> <Jump> }`
   `CONSTRUCT ?s <type> <Discipline> WHERE { ?s <type> <Jump> }`
   These three queries include the same selection: the range query ($*$, `<type>`, `<Jump>`). The SELECT query returns all subjects matched by the range query, the ASK query returns true if any triple exists in the graph, and the CONSTRUCT query returns triples ($*$, `<type>`, `<Discipline>`) for all triples retrieved by the selection.
4. `SELECT ?p ?o WHERE { <organized> ?p ?o }`
   This query selects all triples matched by the range query (`<organized>`, $*$, $*$). The selectivity of this query is probably lower than the selectivity of the queries 2 and 3; however, it is higher compared to the query 1.

Moreover, the selection includes zero or more join operations. In Figure 2, we show two queries including more join operations. A query with one join is shown in Figure 2(a). In this SELECT, we can see two output variables *o1* and *o2*. In Lines 2 and 3, the range queries (*, `<type>`, *) and (*, `<jumps>`, *) are defined. Results of these range queries are then joined using the subject represented by the j variable and objects for variables *o1* and *o2* are returned.

A more complex SPARQL query with join is shown in Figure 2(b). This SELECT also contains the output variables *o1* and *o2*. However, this query is evaluated by a sequence of three joins: the first join involves sets defined by queries in Lines 2 and 3, the second join involves the result of the previous join and the result of the query in Line 4, and the last join involves the result of the previous join and the result of the query in Line 5. The result of the complete query includes subjects and objects for the variables s and o.

```
SELECT ?o1 ?o2 WHERE {
 ?j <type> ?o1 .
 ?j <jumps> ?o2
}
```

```
SELECT ?s ?o WHERE {
 ?s <jumps> ?j1 .
 ?j1 <type> ?j2 .
 ?j2 <sc> ?j3 .
 ?j3 <hasWorlRecord> ?o
}
```



(a) (b)

Fig. 2. Two SPARQL query with join and their graph representations

3 Existing Triple Stores

In Table 2, we show triple stores introduced from 2002 to 2014. These triple stores include academic prototypes, commercial solutions as well as open source projects. Although some details of their implementation are not known, we can distinguish three basic types of the physical design for the triple table [18]:

1. *Triple Table* (*TT*) – in this case, triples are stored in a sequence array.
2. *Property Table* (*PT*) – in this case, we define a tuple $(s, o_1, o_2, \ldots, o_n)$ for properties $p_1, p_2, \ldots p_n$. Tuples of this schema are stored in a sequence array. We can define more property tables in that cases the number of properties is higher than n.
3. *Vertical Partitioning* (*VP*) – the property table where $n = 1$.

Except these main approaches there are also some other variants and improvements, for example Hierarchical Property Partitioning utilized in roStore [17]. In some works, we distinguish the *Multiple indices* approach, which means that some combinations of various indices together with a modification of the above described types are depicted. In Table 2, we can see the B-tree and its variants are the most commonly used data structure indexing the triple table.

[3] `http://www.guha.com/rdfdb/`
[4] `http://rdfstore.sourceforge.net/`
[5] `http://www.bigdata.com/`

Store	**Published**	**Last update**	**Physical design type**	**Index data structure**	**Supported query language**
JENA [34]	2002	2014	PT	Hash-table, B-tree	SPARQL
RDFSuite [3]	2001	2003	PT	B-tree	SPARUL
Sesame [10]	2002	2014	PT	B-tree	SPALUR
3store [23]	2003	2013	TT	Hash-table	RDQL/SPARQL
rdfDB[3]	2004	2010	TT	B-tree	SPARQL
RDFStore[4]	2004	2006	TT	BerkeleyDB	SPARQL
Redland [7]	2002	2014	TT	Hash-table	SPARQL
AllegroGraph[1]	2006	2014		B-tree	SPARQL
sw-Store[2]	2009	2014	VP	B-tree, Bitmap	SPARQL
4store [24]	2009	2013	PT	Hash-table	SPALUR
YARS [26]	2005	2006	MI	B-tree	N3 extension
YARS2 [27]	2007		MI	Sparse index, B-tree	SPARQL
Kowari [42]	2005	2005	MI	AVL tree, B-tree	iTQL/RDQL
Hexastore [41]	2008		MI	B-tree	SPARQL
RDFJoin [35]	2008		VP	B-tree	SPARQL
RDFKB [36]	2009		MI	B-tree	-
BitMat [4]	2009	2013	MI	3D Bitmap	SPARQL-like
RDF-3X [37]	2008	2013	MI	B-tree	SPARQL
Parliament [29]	2009	2014	MI	B-tree, Heap table	-
Virtuoso[16]	2009	2014	MI	B-tree, Bitmap	SPALUR
RDFCube[33]	2007		MI	3D Hash-table	-
GRIN [40]	2007		MI	B-tree	SPARQL
BigData[5]	2008	2014	MI	B-tree	SPARQL 1.1
Oracle [11]	2005	2014	MI	B-tree, R-tree	SPARQL
Marmotta [5]	2013	2014	MI	B-tree	SPARQL

Table 2. Triple Stores. TT - triple table PT - property table VP - vertical partitioning MI - multiple indices

4 Index Data Structures

4.1 B-tree

The B-tree is an one-dimensional paged data structure supporting point and one-dimensional range queries as well as update operations [13]. As result, in the case we want to support a general range query without a sequential scan of all leaf nodes, we have to create more indices.

For example, in the case of a B-tree with the compound key (s, p, o), we can effectively utilize range queries $(s, p, *)$ and $(s, *, *)$. On the other hand, fast processing of the range query $(*, p, o)$ demands a sequential scan over all leaf nodes of the B-tree. To cover all combination of searched dimensions with efficient range query execution, three B-trees have to be created (see Table 3). Consequently, this solution means that the size of indices is probably higher than the table size. This issue is even more evident in the case of the Quad table; in Table 4, we see that we need 6 indices to cover all range queries over quads. There are two problematic issues related to this technique: the higher space overhead and the additional overhead of the update operations since more indices have to be updated.

	Compound key of the B-tree		
	(s, p, o)	(o, s, p)	(p, o, s)
Supporting range queries	(s, p, o)	(o, s, p)	(p, o, s)
	$(s, p, *)$	$(o, s, *)$	$(p, o, *)$
	$(s, *, *)$	$(o, *, *)$	$(p, *, *)$
	$(*, *, *)$	$(*, *, *)$	$(*, *, *)$

Table 3. B-tree indices for the triple table

	Compound key of the B-tree					
	(s, p, o, c)	(p, o, c)	(o, c, s)	(c, s, p)	(c, p)	(o, s)
Supporting range queries	(s, p, o, c)	$(p, *, *)$	$(o, *, *)$	$(c, *, *)$	(c, p)	(o, s)
	$(s, p, o, *)$	$(p, o, *)$	$(o, c, *)$	$(c, s, *)$		
	$(s, p, *, *)$	(p, o, c)	(o, c, s)	(c, s, p)		
	$(s, *, *, *)$					
	$(*, *, *, *)$					

Table 4. B-tree indices for the quad table

4.2 R-tree

Since the multidimensional R-tree [22] supports a general multidimensional range query, we can use it as a solution of the above mentioned problems instead of a sequence scan in the B-tree. The R-tree can be thought of as an extension of the B-tree in a multidimensional space. It corresponds to a hierarchy of nested n-dimensional *minimum bounding rectangles* (MBR). If $\mathcal{N}$ is an interior node, it contains couples of the form (R_i, P_i), where P_i is a pointer to a child of the node $\mathcal{N}$. If R is its MBR, then the rectangles R_i corresponding to the children $\mathcal{N}_i$ of $\mathcal{N}$ are contained in R. Rectangles at the same tree level can overlap. If $\mathcal{N}$ is a leaf node, it contains couples of the form (R_i, O_i), so called *index records*, where R_i contains a spatial object O_i.

The split algorithm has the significant affect on the index performance. Three split techniques (*Linear*, *Quadratic*, and *Exponential*) proposed in [22] are based on a heuristic optimization. The Quadratic algorithm has turned out to be the most effective and other improved versions of R-trees are based on this method. An MBR can overlap another MBR in the same level of the tree; the probability increases linearly with increasing data dimension. This effect is known as *curse of dimensionality* [43].

There are many variants of the R-tree, e.g. R*-trees [8], R$^+$-tree [39]. The R*-tree [8] differs from the R-trees mainly in the insertion algorithm. Although original R-tree algorithms tried only to minimize the area covered by MBRs, the R*-tree algorithms try to minimize overlapping between MBRs at the same levels and maximize the storage utilization. The R$^+$-tree [39] is a variant of the R-tree which allows no overlap between regions corresponding to nodes at the same tree level; however, an item can be stored in more than one leaf node.

Since some intervals of a range query include only one value in the case of the triple table, we call the query as the narrow range query [30]. Therefore, we utilize the Signature R-tree [30] allowing to handle the range query more efficiently than the R-tree and its variants. Moreover, we use the Ordered R-tree [31] since we can define an ordering of attributes. These data structures are described in the following sections.

4.3 Signature R-tree

The Signature R-tree [30] contains MBRs in inner nodes (we suppose point data in leaf nodes) and one signature related to each MBR. The signature is created for tuples inserted in the subtree related to each MBR. As result, we can use two types of filtering when a range query scans the tree: the first filtering method tests whether an MBR is intersected by a query rectangle and the second filtering method tests whether a signature can include tuples of the query. As result, the Signature R-tree reads a lower number of nodes during the range query processing. This R-tree variant is however proposed only for point data and narrow range queries.

4.4 Ordered R-tree

The Ordered R-tree [31] is a simple combination of the R-tree and the B-tree. It means, we can use a general multidimensional range query, however we can define an ordering for tuples inserted in the tree. Evidently, we can define only one ordering in one tree. There are two consequences:

1. For some range queries (corresponding to ordering defined for the tree), all leaf nodes intersected by the query rectangle include only result tuples. It is not generally true for the R-tree and its variants, but the range query of the B-tree provides the same behaviour.
2. We get tuples of the result sorted and it is not necessary to sort them after the range query is processed.

In this article, we utilize mainly the first property.

5 Experiments

In our experiments[6], we compare the B-tree, as the main index data structure utilized in semantic DBMS, with the R-tree[7], Signature R-tree, and Ordered R-tree. All index data structures are implemented in C++[8]. We utilize a generated synthetic data collection called LUBM including 133,573,856 triples [21], the size of the text file is 22.2 GB.

Query Group	**Type**	**Result set size**	**#Queries**	**#Iterations**
1	Range query	$< 1; 1 >$	6	10,000
2	Range query	$< 2; 1,000 >$	6	50
3	Range query	< 1,001; 1,000,000 >	6	1
4	Range query	$< 1,000,001; \infty)$	6	1
5	Point query	$< 1; 1 >$	33,234,949	1

Table 5. Specification of query groups

We test the performance of point and range queries processed over the index data structures when a SPARQL query is evaluated. We use 5 groups of queries determined by the selectivity (see Table 5)[9]. QG5 represents a sequence of point queries processed during a join operation. In the case of QG1 and QG2, it is necessary to repeat a sequence of queries since the processing time of one query is unmeasurable. The number of iterations is written in the column #Iteration of the table. The column #Queries contains a number of various queries in one query group.

[6] We run our experiments on 2 x Intel Xeon E5 2690 2.9GHz and 300GB RAM memory, OS Windows Server 2008.

[7] More precisely, the R*-tree has been tested.

[8] A part of the RadegastDB framework developed by DBRG – `http://db.cs.vsb.cz/`

[9] A complete list of queries can be found in `http://db.cs.vsb.cz/TechnicalReports/indices_for_rdf_data-query.pdf`

We built the B-trees, the R-tree, the Signature R-tree, and the Ordered R-trees for the test data collection[10]. In Table 6 and Figure 3, we see basic characteristics of these indices. Since these data structures include string ids instead of strings, a term index is built. In the case of the Ordered R-tree, we do not need more trees like in the case of the B-tree, however, in this article, we want to test whether it is possible to find an optimal ordering for the Ordered R-tree, therefore we build the tree for more orderings of the attributes. We can see that the B-tree size is up-to 3× higher than the size of the R-tree-based indices. The R-tree is build in 58% of the B-tree build time. On the other hand, the build time for other R-tree-based indices is up-to 2× less efficient compared to the B-tree.

Index Data Structure		**#Nodes**	**Size [GB]**	**Build Time [s]**
Term index		4,543,671	8.67	3,794.7
B-tree	(s, p, o)	4,465,853	8.51	3,857.9
	(p, o, s)			
	(o, s, p)			
R-tree		1,495,289	2.85	**2,228.1**
Signature R-tree		1,641,905	3.13	6,143.5
Ordered R-tree	(s, p, o)	1,541,677	2.94	6,404.1
	(p, o, s)	1,499,602	2.86	7,193.5
	(o, s, p)	1,433,703	2.73	6,791.1
	(s, o, p)	1,541,677	2.94	7,232.2
	(p, s, o)	1,579,935	3.01	7,535.6
	(o, p, s)	**1,429,151**	**2.73**	6,933.0

Table 6. Statistics of index data structures

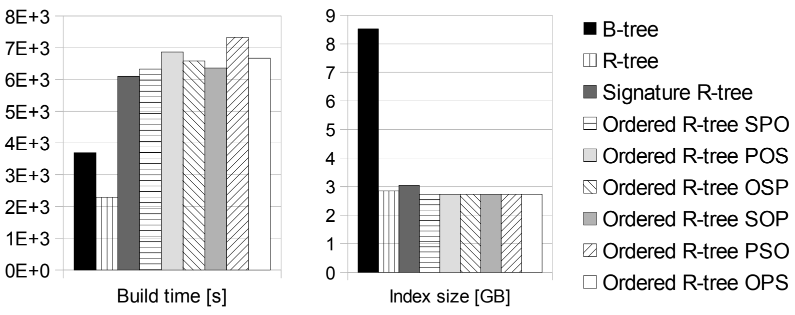


Fig. 3. Index build time and index size

[10] The page size is 2,048 B for all data structures.

In Figure 4, we can see the query processing time for all query groups; the processing time is the average time of all queries in one group. Similarly, Figure 5 includes DAC for all query groups. Evidently, the B-tree provides the most efficient performance especially in the case of the higher selectivity. The reason of this result is the minimal DAC of the B-tree since only leaf nodes including result tuples are scanned. In the case of the lower selectivity (see GP4 in Figure 4), results of all index data structures are similar.

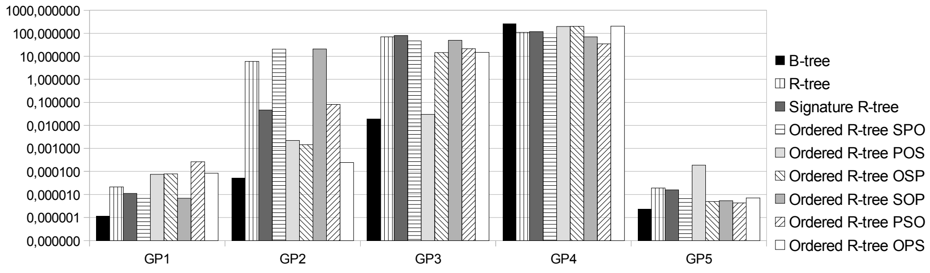


Fig. 4. Processing time for all query groups [s]

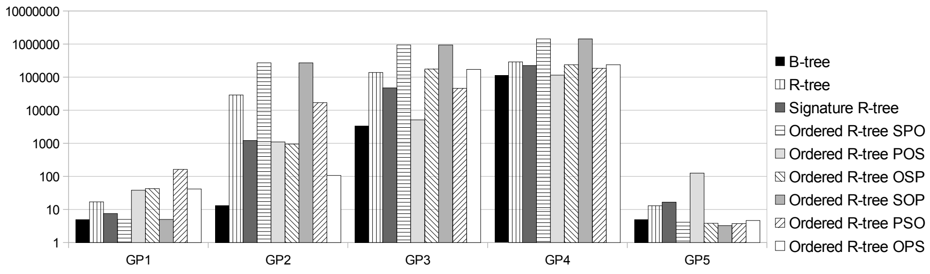


Fig. 5. DAC for all query groups

We see that the Signature R-tree and the Ordered R-tree outperform the R-tree in most cases. Although the average processing time of the Signature R-tree is lower compared to the Ordered R-tree, we can find a query in each query group where it exists an ordering of the Ordered R-tree such that the Ordered R-tree outperforms the Signature R-tree. Let us consider query processing times in Figure 6. In the case of Q1 `(S='AssociateProfessor', P='type', O=*)`, the Ordered R-trees SPO and SOP outperform the Signature R-tree and other Ordered R-trees, however in the case of Q7 `(S=*, P='PublicationAuthor', O='AssistentProfessor')` the performance of these Ordered R-trees is the lowest. Similarly, in the case of Q11 `(S=*, P=*, O='Course2')`, the Ordered R-tree OPS outperforms other R-tree variants and its performance is the same as the performance of the B-tree. Similarly, in the case of Q14 `(S=*, P='worksFor', O=*)`, the Ordered R-tree SOP outperforms other R-tree variants. However, we must keep in mind that this effect depends on a query and a concrete ordering of the Ordered R-tree.

Although, it is clear that the B-tree provides the most efficient processing time, there are some improvements of multidimensional data structures. The first one, the index size of a multidimensional data structure is up to 3× lower the B-tree index size. The second one, in the case of the B-tree it is necessary to change ordering of values in a triple when a query processor want to use an index with different ordering than another index returns, it means an additional time overhead in this case.

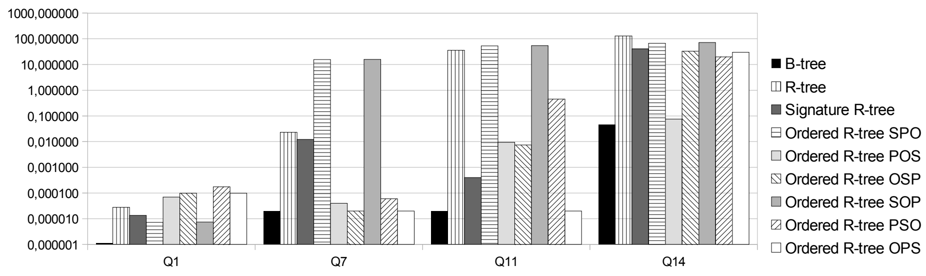


Fig. 6. Processing time for some queries [s]

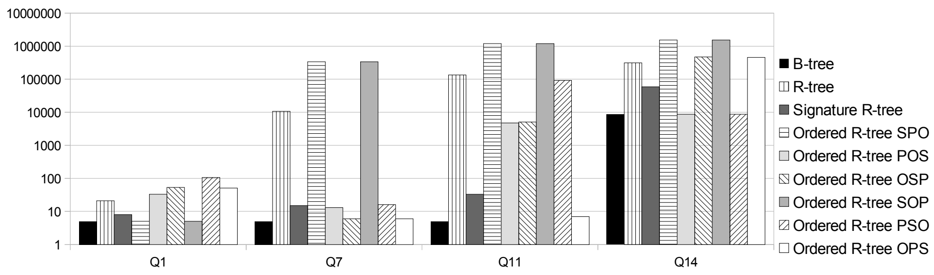


Fig. 7. DAC for some queries

As result, let us consider a workload including queries accessing the most tree nodes. If the cache size is lower than the number of B-tree nodes, a multidimensional data structure would provide the higher performance than the B-tree in the case the cache includes all nodes of the multidimensional data structure.

6 Conclusion

In this article, we compared the performance of the B-tree with the R-tree, the Signature R-tree, and the Ordered R-tree for the triple table and point and range queries processed during the evaluation of a SPARQL query. The Signature R-tree and the Ordered R-tree outperform the R-tree for most queries. Although the average processing time of the Signature R-tree is lower compared to the Ordered R-tree, in each query group, we can find a query where there is such an ordering of the Ordered R-tree outperforming the Signature R-tree.

The B-tree provides the most efficient processing time; the average processing time of the B-tree is 74% of the Signature R-tree's processing time. However, there are some specific improvements of multidimensional data structures. The first one, index size of a multidimensional data structures is up to $3\times$ lower than the B-tree index size. The second one, in the case of the B-tree it is necessary to change ordering of values in each triple when a query processor want to use an index with different ordering than another index returns. Consequently, it means an additional time overhead of the query processing.

References

[1] J. Aasman. *Allegro Graph: RDF Triple Database.* Tech. rep. Technical Report 1, Franz Incorporated, 2006. URL: `http://www.franz.com/agraph/allegrograph/`.

[2] D.J. Abadi et al. "SW-Store: a vertically partitioned DBMS for semantic web data management". In: *The VLDB Journal* 18.2 (2009), pp. 385–406.

[3] S. Alexaki et al. "The ICS-FORTH RDFSuite: Managing voluminous RDF description bases". In: *Proceedings of 2nd Internacional Workshop on the Semantic Web (SemWeb'01).* 2001.

[4] M. Atre, J. Srinivasan, and J.A. Hendler. *BitMat: A Main Memory RDF Triple Store.* Tech. rep. 2009. URL: `http://www.cs.rpi.edu/~atrem/bitmat_techrep.pdf`.

[5] Reto Bachmann-Gmur. *Instant Apache Stanbol.* Packt Publishing Ltd, 2013. ISBN: 978-1-78328-123-7.

[6] Amos Bairoch et al. "The universal protein resource (UniProt)". In: *Nucleic acids research* 33 (2005), pp. D154–D159.

[7] D. Beckett. "The design and implementation of the Redland RDF application framework". In: *Computer Networks* 39.5 (2002), pp. 577–588.

[8] Norbert Beckmann et al. "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles". In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD 1990).* Vol. 19. AMC, 1990, pp. 322–331.

[9] Tim Bray et al. "Extensible markup language (XML)". In: *World Wide Web Journal* 2.4 (1997), pp. 27–66.

[10] J. Broekstra, A. Kampman, and F. Van Harmelen. "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema". In: *Proceedings of the Semantic Web-ISWC.* Vol. 2342. Springer, 2002.

[11] E.I. Chong et al. "An efficient SQL-based RDF querying scheme". In: *Proceedings of 31th International Conference on Very Large Data Bases (VLDB 2005).* VLDB Endowment. 2005, pp. 1216–1227.

[12] Erik Christensen et al. *Web services description language (WSDL) 1.1.* Recommendation. W3C, 2001. URL: `http://www.w3.org/TR/wsdl`.

[13] Douglas Comer. "Ubiquitous B-tree". In: *ACM Computing Surveys (CSUR)* 11.2 (1979), pp. 121–137.

[14] R. Cyganiak, A. Harth, and A. Hogan. *N-quads: Extending n-triples with context.* Tech. rep. 2008. URL: `http://sw.deri.org/2008/07/n-quads/`.

[15] Martin Dürst and Michel Suignard. *Internationalized resource identifiers (IRIs).* Tech. rep. RFC 3987, January, 2005. URL: `http://www.ietf.org/rfc/rfc3987.txt`.

[16] Orri Erling and Ivan Mikhailov. "Virtuoso: RDF support in a native RDBMS". In: (2010), pp. 501–519.

[17] David Faye et al. "RDF triples management in roStore". In: *Actes de IC2011* (2012), pp. 755–770.

[18] David Célestin Faye, Olivier Curé, and Guillaume Blin. "A survey of RDF storage approaches". In: *ARIMA Journal* 15 (2012). URL: `http://arima.inria.fr/015/015002.html`.

[19] Tim Finin et al. "Social networking on the semantic web". In: *Learning Organization journal* 12.5 (2005), pp. 418–435.

[20] S. Groppe. *Data management and query processing in semantic web databases.* Springer, 2011. ISBN: 978-3-642-19356-9.

[21] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. "LUBM: A benchmark for OWL knowledge base systems". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 3.2 (2005), pp. 158–182.

[22] Antonin Guttman. "R-trees: a dynamic index structure for spatial searching". In: *Proceedings of the ACM International Conference on Management of Data, (SIGMOD '84).* Vol. 14. 2. 1984, pp. 47–57.

[23] S. Harris and D.N. Gibbins. "3store: Efficient bulk RDF storage". In: *volume 89 of CEUR Workshop Proceedings* (2003).

[24] S. Harris, N. Lamb, and N. Shadbolt. "4store: The design and implementation of a clustered RDF store". In: *Proceedings of 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009).* 2009, pp. 94–109.

[25] S. Harris and A. Seaborne. "SPARQL 1.1 query language". In: *W3C Recommendation* (2013). URL: `http://www.w3.org/TR/sparql11-query/`.

[26] A. Harth and S. Decker. "Optimized index structures for querying rdf from the web". In: *Proceedings of 3th Latin American Web Congress, (LA-WEB 2005).* IEEE. 2005.

[27] A. Harth et al. "Yars2: A federated repository for querying graph structured data from the web". In: *The Semantic Web* 4825 (2007), pp. 211–224.

[28] M Tim Jones. *Artificial Intelligence A System Approach.* Laxmi Publications, Ltd., 2008. ISBN: 978-0763773373.

[29] D. Kolas, I. Emmons, and M. Dean. “Efficient linked-list rdf indexing in parliament”. In: *Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems.* Vol. 9. 2009, pp. 17–32.

[30] Michal Krátký et al. “Efficient processing of narrow range queries in multidimensional data structures”. In: *Proceedings of 10th International Database Engineering and Applications Symposium, (IDEAS'06).* IEEE. 2006.

[31] Filip Křižka, Michal Krátký, and Radim Bača. “On support of ordering in multidimensional data structures”. In: *Proceedings of Advances in Databases and Information Systems (ADBIS 2010).* Vol. 6295. LNCS. Springer. 2010, pp. 575–578.

[32] Frank Manola, Eric Miller, Brian McBride, et al. “RDF primer”. In: *W3C recommendation* 10 (2004). URL: `http://www.w3.org/TR/rdf-primer/`.

[33] Akiyoshi Matono, SaidMirza Pahlevi, and Isao Kojima. “RDFCube: A P2P-Based Three-Dimensional Index for Structural Joins on Distributed Triple Stores”. In: *Databases, Information Systems, and Peer-to-Peer Computing.* Vol. 4125. LNCS. Springer, 2007. ISBN: 978-3-540-71660-0.

[34] B. McBride. “Jena: A semantic web toolkit”. In: *Internet Computing, IEEE* 6.6 (2002), pp. 55–59.

[35] J.P. McGlothlin and L.R. Khan. *RDFJoin: A scalable data model for persistence and efficient querying of RDF datasets.* Tech. rep. 2009.

[36] J.P. McGlothlin and L.R. Khan. “RDFKB: efficient support for RDF inference queries and knowledge management”. In: *Proceedings of the 2009 International Database Engineering & Applications Symposium.* ACM. 2009, pp. 259–266.

[37] T. Neumann and G. Weikum. “RDF-3X: a RISC-style engine for RDF”. In: *Proceedings of the VLDB Endowment.* Vol. 1. 1. VLDB Endowment, 2008, pp. 647–659.

[38] A. Seaborne et al. “SPARQL/Update: A language for updating RDF graphs”. In: *W3C Member Submission* 15 (2008).

[39] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. “The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects”. In: *Proceedings of 13th International Conference on Very Large Data Bases (VLDB 1997).* Morgan Kaufmann, 1987.

[40] Octavian Udrea, Andrea Pugliese, and VS Subrahmanian. “GRIN: A graph based RDF index”. In: *Proceedings of the 22nd national conference on Artificial intelligence, (AAAI'07).* Vol. 1. 2007, pp. 1465–1470.

[41] C. Weiss, P. Karras, and A. Bernstein. “Hexastore: sextuple indexing for semantic web data management”. In: *Proceedings of the VLDB Endowment* 1.1 (2008), pp. 1008–1019.

[42] D. Wood, P. Gearon, and T. Adams. “Kowari: A platform for semantic web storage and analysis”. In: *Proceedings of XTech 2005 Conference.* 2005.

[43] Cui Yu. *High-Dimensional Indexing.* Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 2002. ISBN: 3-540-44199-9.

Vocabulary for Linked Data Visualization Model[⋆]

Jakub Klímek[1] and Jiří Helmich[2]

[1] Czech Technical University in Prague, Faculty of Information Technology
Thákurova 9, 160 00 Praha 6, Czech Republic
klimek@fit.cvut.cz

[2] Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
helmich@ksi.mff.cuni.cz

Abstract. There is already a vast amount of Linked Data on the web. What is missing is a convenient way of analyzing and visualizing the data that would benefit from the Linked Data principles. In our previous work we introduced the Linked Data Visualization Model (LDVM). It is a formal base that exploits the principles to ensure interoperability and compatibility of compliant components. In this paper we introduce a vocabulary for description of the components and an analytic and visualization pipeline composed of them. We demonstrate its viability on an example from the Czech Linked Open Data cloud.

Keywords: Linked Data, RDF, visualization, vocabulary

1 Introduction

Vast amount of data represented in a form of Linked Open Data (LOD) is now available on the Web. Unfortunately, not so many users are capable of using the data in a useful way yet. The data is represented in RDF and often uses commonly known vocabularies, which brings opportunities for data analysis and visualization that were not there before. However, the appropriate tools that would exploit these new benefits are still lacking.

Figure 1 shows datasets transformed to Linked Data by our research group over the past few years. The circles are the individual datasets and the edges mean there is a decent amount of links among entities of the two datasets. This gives the users some very rough ideas of what they can find in those datasets. Each dataset should also be described by its metadata, which gives more information about what is inside. However, the Linked Data principles offer more. For each of our datasets a SPARQL endpoint – an open endpoint to a database where everyone can place a structured query - is available. This in combination with commonly used Linked Data vocabularies means that anyone can simply ask whether a particular dataset contains interesting data. The obvious issue here is that non expert users do not know SPARQL so they do not know how to ask the right question. For example, if a user is interested in opening hours of a particular institution of public power, he could query the appropriate dataset that

[⋆] This work was partially supported by a grant from the European Union's 7th Framework Programme number 611358 provided for the project COMSODE

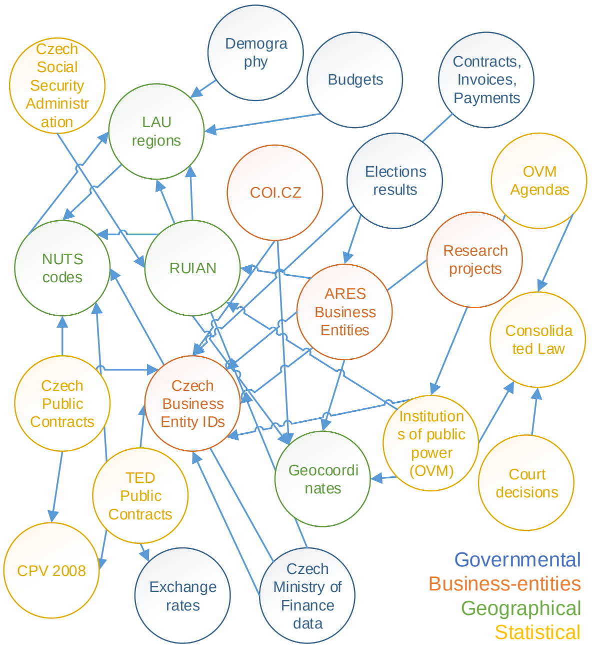


Fig. 1. Czech Linked Open Data Cloud

he sees in our Czech LOD cloud, if he knew how. This situation is somehow similar to programming and common algorithms. For every programming language there are libraries of algorithms implemented by experts who know how to do that, packaged for use by people who do not. Because on the Web of Data there are vocabularies that are de facto standards for representation of certain types of data such as opening hours of locations in general[3], if the data is in the dataset, it would be found by a general query suited for this, perhaps written by an expert. This means that a regular user could find our dataset of institutions of public power and assume that it uses the standard vocabulary. Then he could use the query from a library of queries suited for common tasks using the common vocabularies and execute this query on a dataset of his choosing. He would get the result, possibly even displayed in a user friendly way, again thanks to the standard vocabularies and all this without understanding SPARQL, RDF, Linked Data, etc.

Previously, we introduced the *Linked Data Visualization Model* (LDVM) [4], which allows users to create and reuse analytic and visualization components that leverage the Linked Data principles. We also showed a tool *Payola* [6] implementing the model and in [7] we demonstrated that expert users can prepare analyses and visualizations and allow non-experts to use them to get data from the LOD cloud[4].

[3] `http://www.heppnetz.de/ontologies/goodrelations/v1.html#OpeningHoursSpecification`
[4] `http://lod-cloud.net/`

In this paper we introduce the LDVM vocabulary, which allows LDVM implementations to store and exchange configuration of individual LDVM components as well as whole analytic and visualization pipelines in RDF and in compliance with the Linked Data principles. The vocabulary contains support for pipeline nesting so that complete pipelines created by experts can be wrapped as another component to be used in pipelines by non-experts. By publishing the vocabulary we also open our approach to Linked Data analysis and visualization so that anyone who is interested can easily create a reusable component or pipeline and share it with others. The technical benefits are easy sharing, open format easily adoptable by other implementations, easier management of configurations – all the configurations can be maintained by SPARQL queries and the possibility to configure the components and pipelines programmatically. In addition, there are all the generally known Linked Data benefits such as ability to better provide context through linking to other sources, better provenance tracking, etc.

This paper is structured as follows. In section 2 we briefly describe the principles of LDVM. In section 3 we introduce the LDVM vocabulary, which is the main contribution of this paper. In section 4 we show the usage of the vocabulary on examples. In section 5 we survey related work and in section 6 we conclude.

2 Linked Data Visualization Model

In our previous work we defined the Linked Data Visualization Model (LDVM), an abstract visualization process customized for the specifics of Linked Data. In short, LDVM allows users to create data visualization pipelines that consist of four stages: Source Data, Analytical Abstraction, Visualization Abstraction and View. The aim of LDVM is to provide means of creating reusable components at each stage that can be put together to create a pipeline even by non-expert users who do not know RDF. The idea is to let expert users to create the components by configuring generic ones with proper SPARQL queries and vocabulary transformations. In addition, the components are configured in a way that allows the LDVM implementation to automatically check whether two components are compatible or not. If two components are compatible, then the output of one can be connected to the input of the other in a meaningful way. With these components and the compatibility checking mechanism in place, the visualization pipelines can then be created by non-expert users.

2.1 Model Components

There are four stages of the visualization model populated by LDVM components. *Source Data* stage allows a user to define a custom transformation to prepare an arbitrary dataset for further stages, which require their input to be RDF. In this paper we only consider RDF data sources such as RDF files or SPARQL endpoints, e.g. DBPedia. The LDVM components at this stage are called *data sources*. The *Analytical Abstraction* stage enables the user to specify analytical operators that extract data to be processed from one or more data sources and then transform it to create the desired analysis. The transformation can also compute additional characteristics like aggregations. For example, we can query for resources of type `dbpedia-owl:City` and then compute the number of cities in individual countries. The LDVM components at this stage are called *analyzers*. In the *Visualization Abstraction* stage of LDVM we need to prepare the data to be compatible with the desired visualization technique. We could have prepared the analytical abstraction in a way that is directly compatible with a visualizer. In that case, this step can be skipped. However, the typical use case for Visualization Abstraction is to facilitate reuse of existing analyzers and existing visualizers that work with similar data, only in different formats. For that we need to use a LDVM *transformer*. In *View* stage, data is passed to a *visualizer*, which creates a user-friendly visualization. The components, when connected together, create a analytic and visualization pipeline which, when executed, takes data from a source and transforms it to produce a visualization at the end. However, not every component can produce meaningful results from any input. Typically, each component is designed for a specific purpose, e.g. visualizing map data, and it does not make sense with other data. This means that only components that are somehow compatible can create a meaningful pipeline.

2.2 Component Compatibility

Now that we described the four basic types of LDVM components, let us take a look at the notion of their *compatibility*, which is a key feature of LDVM. We first introduced the idea of compatibility checking in [4] and then further refined it in [7]. However, as the implementation progressed we developed this feature even further.

The idea is based on the ability to check whether a component can work with the data it has on its input. We can check this using e.g. a SPARQL query, but we can do that only when the pipeline is already running, when we actually have the data to check. However, we want to use the component compatibility in design time to rule out component combinations that do not make any sense and to help the users to use the right components before they actually run the pipeline. Therefore, we need a way to check the compatibility without the actual data. For this, we use two constructs - an *input descriptor* and an *output data sample*. The input descriptor describes what is expected in the input data. For simplicity, let us use a set of SPARQL queries for the descriptor. A descriptor is bound to an input of its component.

In order to evaluate the descriptors in design time, we require that each LDVM component that produces data (data source, analyzer, transformer) also provides a static sample of the resulting data. For the data sample to be useful, it should be as small as possible, so that the input descriptors of other components execute as fast as possible. Also, it should contain the maximum amount of classes and properties whose instances can be produced by the component, making it as descriptive as possible. For example, when an analyzer transforms data about cities and their population, its output data sample will contain a representation of one city with all the properties that the component can possibly produce given it has all the inputs it needs. Note that, e.g. for data sources, it is also possible to implement the evaluation of descriptors over the output data sample as evaluation directly on the represented SPARQL endpoint.

Each LDVM component has a set of *features*, where each feature represents a part of the expected component functionality. A component feature can be either mandatory or optional. For example, a visualizer that displays points and their descriptions on a map can have 2 features. One feature represents the ability to display the points on a map.

This one will be mandatory, because without the points, the whole visualisation lacks purpose. The second feature will represent the ability to display a description for each point on the map. It will be optional, because when there is no data for the description, the visualization still makes sense - there are still points on a map. Whether a component feature can be used or not depends on whether there is the data needed for it on the input, therefore, each feature is described by a set of input descriptors.

We say that a feature of a component in a pipeline is *usable* when all queries in all descriptors are evaluated `true` on their respective inputs. A component is *compatible* with the mapping of outputs of other components to its inputs when all its mandatory features are usable. The usability of optional features can be further used to evaluate the expected quality of the output of the component. For simplicity, we do not elaborate on the output quality in this paper. The described mechanism of component compatibility can be used in design time for checking of validity of the visualization pipeline. It can also be used for suggestions of components that can be connected to a given component output. In addition, it can be used in run time for verification of the compatibility using the actual data that is passed through the pipeline. Finally, this concept can be also used for periodic checking of data source content, e.g. whether the data has changed its structure and therefore became unusable or requires pipeline change.

3 LDVM Vocabulary

In our current take on implementation of LDVM we aim to have individual components running as independent services that exchange only information needed to access the input and output data. Also we aim for easy configuration of individual components as well as easy configuration of the whole pipeline. In accordance with the Linked Data principles, we now use RDF as the format for storage and exchange of configuration so that any component that works with RDF can use LDVM components both individually and in a pipeline. For this purpose we have devised a vocabulary for LDVM, which is the main contribution of this paper. In Figure 2 there is a UML class diagram depicting the structure of the vocabulary. Boxes represent classes, edges represent object properties (links) and properties listed inside of the class boxes represent data properties. We chose the `ldvm`[5] prefix for the vocabulary, which is developed on GitHub[6].

3.1 Templates and Instances

There are blue and green classes. The blue classes belong to template level of the vocabulary and green classes belong to the instance level. The easiest way to imagine the division is to imagine a pipeline editor with a toolbox. In the toolbox, there are LDVM component templates with their default configuration. When a designer wants to use a LDVM component in a pipeline, he drags it onto the editor canvas, creating an instance. There can be multiple instances of the same LDVM component template in a single pipeline, each with configuration that overrides the default one. The template holds input descriptors and output data samples, which are used for the compatibility checking. The

[5] `http://linked.opendata.cz/ontology/ldvm/`

[6] `https://github.com/payola/ldvm`

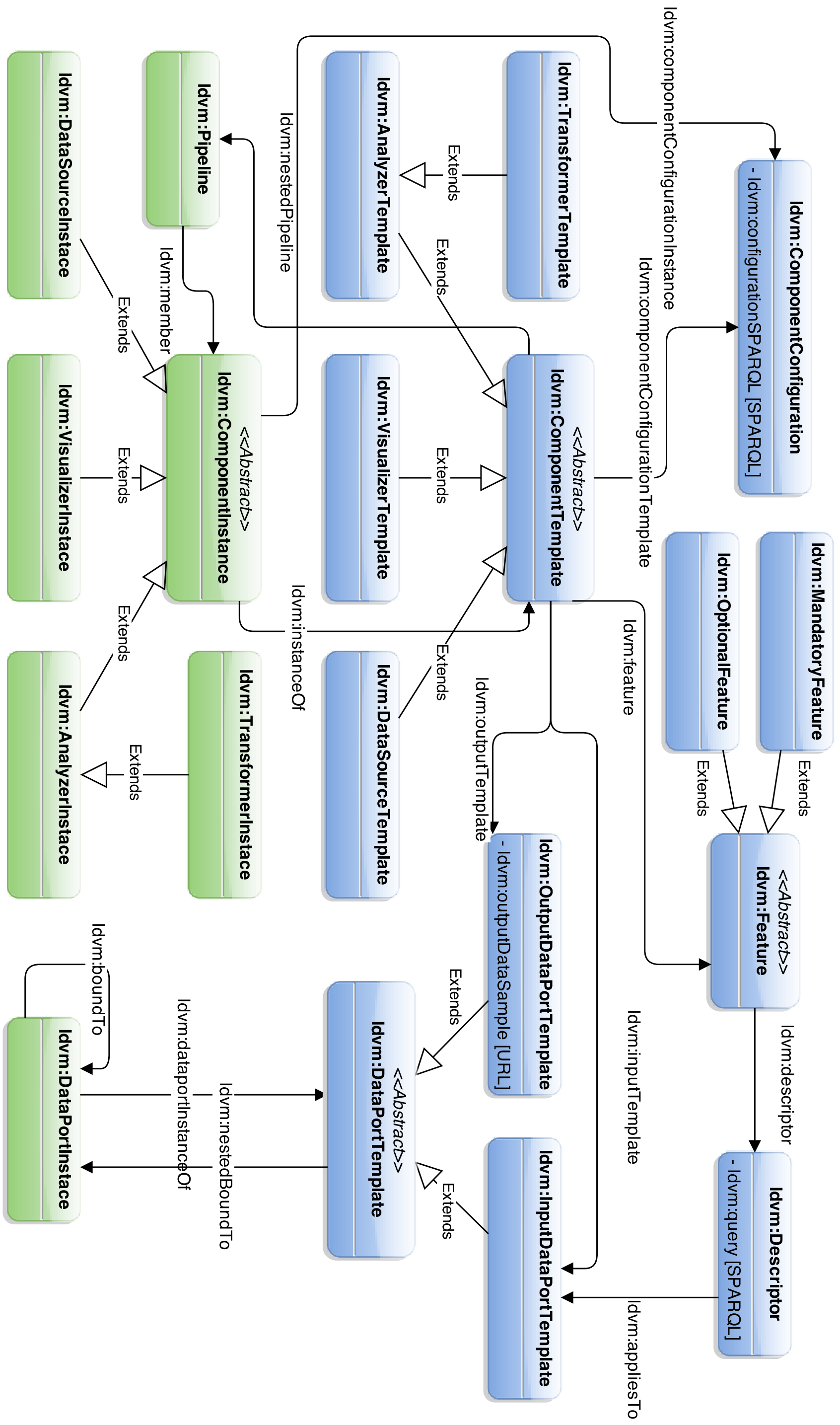


Fig. 2. LDVM Vocabulary

instance configuration and input and output mappings are then used for compatibility checking of a finished pipeline, which also depends on the content of the data sources. Also, they are used during pipeline execution to verify compatibility on the actual data. Each instance is connected to its template using the `ldvm:instanceOf` property.

3.2 Component Types

There are four basic component types as described in subsection 2.1 - data sources, analyzers, transformers and visualizers. They have their representation on both the template level - descendants of the `ldvm:ComponentTemplate` class - and instance levels - descendants of the `ldvm:ComponentInstance` class. From the implementation point of view, transformers are just analyzers with one input and one output, so the difference is purely semantic. This is why transformers are subclass of analyzers.

3.3 Data Ports

Components have input and output data ports. On the template level we distinguish the inputs and outputs of a component. To `ldvm:InputDataPortTemplate` the input descriptors of features can be applied. `ldvm:OutputDataPortTemplate` has the `ldvm:outputDataSample` links to the output data samples. Both are subclasses of `ldvm:DataPortTemplate`. The data ports are mapped to each other - output of one component to input of another - as instances of `ldvm:DataPortInstance` using the `ldvm:boundTo` property. This data port instance mapping forms the actual visualization pipeline, which can be then executed. Because data ports are not LDVM components, their instances are connected to their templates using a separate property `ldvm:dataPortInstanceOf`.

3.4 Features and Descriptors

On the template level, features and descriptors (see subsection 2.2) of a component are represented. Each component template can have multiple features connected using the `ldvm:feature` property. The features themselves - instances of either the `ldvm:MandatoryFeature` class or the `ldvm:OptionalFeature` class - can be described using standard Linked Data techniques and vocabularies such as `dcterms` and `skos`. Each feature can have descriptors, instances of `ldvm:Descriptor` connected using the `ldvm:descriptor` property. The descriptors have their actual SPARQL queries as literals connected using the `ldvm:query` property. In addition, the input data port templates to which the particular descriptor is applied are denoted using the `ldvm:appliesTo` property.

3.5 Configuration

Now that we have the LDVM components, we need to represent their configuration. On the template level, components have their default configuration connected using the `ldvm:componentConfigurationTemplate` property. On the instance level, components point to their configuration, when it is different from the default one, using the `ldvm:componentConfigurationInstance` property. The configuration itself is the same whether it is on the template level or the instance level and therefore we do not distinguish the levels here and we only have one class `ldvm:ComponentConfiguration`.

The structure of the configuration of a LDVM component is completely dependent on what the component needs to function. It is also RDF data and it can use various vocabularies. It can be even linked to other datasets according to the Linked Data principles. Therefore it is not a trivial task to determine the boundaries of the configuration data in the RDF data graph in general. On the other hand, each component knows precisely what is expected in its configuration and in what format. This is why we need each component to provide a SPARQL query that can be used to obtain its configuration data so that the LDVM implementation can extract it. That SPARQL query is connected to every configuration using the mandatory `ldvm:configurationSPARQL` property.

3.6 Pipeline

Finally, the pipeline itself is represented by the `ldvm:Pipeline` class instance. It links to all the instances of LDVM components used in the pipeline.

3.7 Nested Pipelines

A key feature for collaboration of expert and non-expert users is pipeline nesting. An expert can create a pipeline that is potentially complex in number of components, their configuration and binding, but could be reused in other pipelines as a black box data source, analyzer or transformer. The intuitive way of achieving this goal is to let the expert to create the pipeline without a visualizer and potentially even without a data source. This pipeline would then create the black box with its own inputs represented by the missing input mappings of the inner pipeline and outputs represented by the outputs of the inner components to which no input is bound. However, there is one conceptual problem. This inner pipeline is made of component instances and we want to create a component template (reusable black box) out of it. For this, we need a property `ldvm:nestedPipeline` that indicates, that a pipeline is nested in the component template. In addition, we need to map the input data port templates of the new component template to be bound to the input data port instances of the components of the inner pipeline. Also, we need the output instances of the components of the inner pipeline to be bound to the output data port templates of the new component template. This is indicated by the `ldvm:nestedBoundTo` property.

4 Examples

In this section we will introduce examples of how actual templates and instances use LDVM. We use the Turtle RDF syntax[7] and due to space limitations we shorten full URLs and omit human readable labels in the data, which we otherwise recommend according to the "label everything" principle.

[7] http://www.w3.org/TR/turtle/

4.1 SPARQL Analyzer Template

In this section we show how a SPARQL analyzer component template uses LDVM vocabulary. See Listing 1.1.

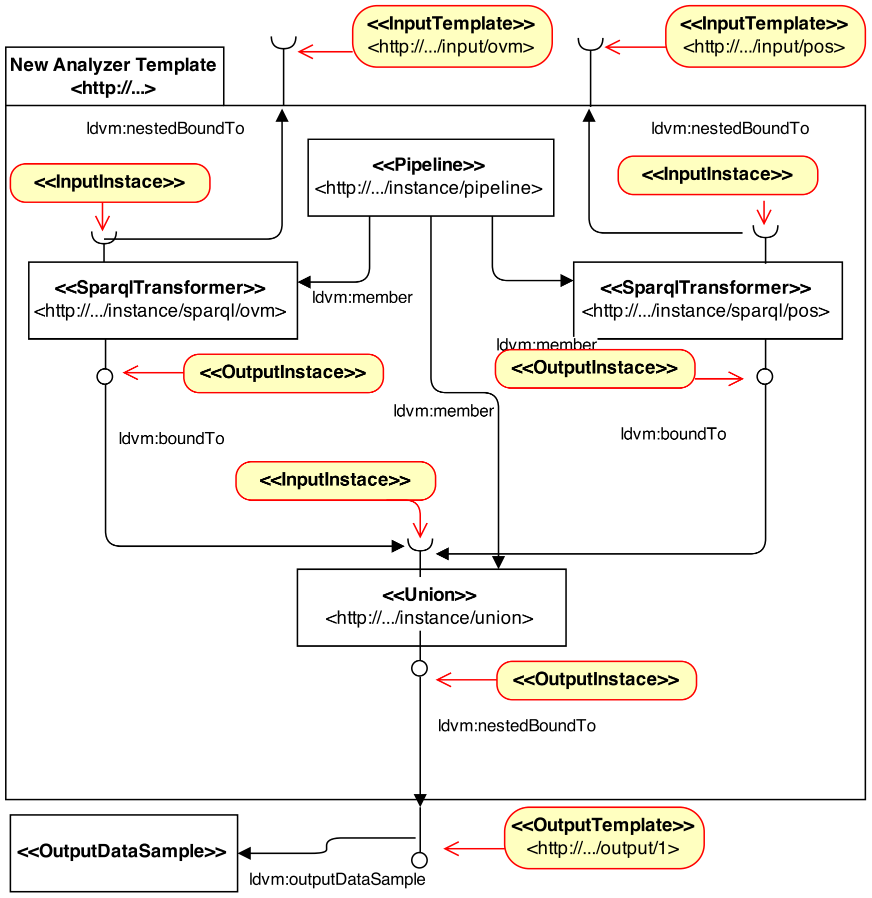
```
 1 a-sparql:SparqlAnalyzerConfiguration a rdfs:Class ;
 2   rdfs:subClassOf ldvm:ComponentConfiguration .
 3 a-sparql:query a rdf:Property ;
 4   rdfs:domain a-sparql:SparqlAnalyzerConfiguration ;
 5   rdfs:range xsd:string .
 6 a-sparql-r:Configuration a a-sparql:SparqlAnalyzerConfiguration ;
 7   a-sparql:query "CONSTRUCT {GRAPH ?g {?s ?p ?o}} WHERE {GRAPH ?g {?s ?p ?o}}" ;
 8   ldvm:configurationSPARQL """
 9     PREFIX a-sparql: <http://linked.opendata.cz/ontology/ldvm/analyzer/sparql/>
10
11     CONSTRUCT {
12       ?config a-sparql:query ?query;
13           dcterms:title ?title .
14     }
15     WHERE {
16       ?config a a-sparql:SparqlAnalyzerConfiguration;
17       OPTIONAL {?config a-sparql:query ?query . }
18       OPTIONAL {?config dcterms:title ?title . }
19     }
20     """ .
21 a-sparql-r:Input a ldvm:InputDataPortTemplate .
22 a-sparql-r:Output a ldvm:OutputDataPortTemplate .
23 a-sparql-r:Descriptor a ldvm:Descriptor ;
24   ldvm:query """ASK {?s ?p ?o}""" ;
25   ldvm:appliesTo a-sparql-r:Input .
26 a-sparql-r:Feature a ldvm:MandatoryFeature ;
27   ldvm:descriptor a-sparql-r:Descriptor .
28 a-sparql-r:SparqlAnalyzerTemplate a ldvm:AnalyzerTemplate ;
29   ldvm:componentConfigurationTemplate a-sparql-r:Configuration ;
30   ldvm:inputTemplate a-sparql-r:Input ;
31   ldvm:outputTemplate a-sparql-r:Output ;
32   ldvm:feature a-sparql-r:Feature .
```

Listing 1.1. SPARQL Analyzer example

First, note that each LDVM component should define its own mini-vocabulary needed for its configuration. In the case of an analyzer that executes a SPARQL query, we need to configure the query. Therefore, we create a class representing the configuration of the SPARQL analyzer - see line 1 - a subclass of `ldvm:ComponentConfiguration`. Then we define the property to be used for the SPARQL query - see line 3 and we instantiate the the configuration as a default configuration - see line 6. Note the query that actually gets the whole configuration. This one would actually get every configuration of every SPARQL analyzer in the data. The LDVM implementation adds a special BIND clause that fixates the `?config` variable on the URI of the specific configuration. In addition, the component template has an input (line 21), output (line 22) and a mandatory feature (line 26) with its descriptor (line 23) that returns `true` whenever there is some RDF data on the input. Finally, we create the new component template itself on line 28.

4.2 Nested Pipeline Example

In this section we show how a nested pipeline instance can be wrapped into a new analyzer template. It is a simple pipeline of 3 analyzers where the first two take the input data from individual inputs, transform it and pass it to the third one. The third one merges the data and passes it to the output.



New Analyzer Template <http://...>
<<InputTemplate>> <http://.../input/ovm>
<<InputTemplate>> <http://.../input/pos>
ldvm:nestedBoundTo
ldvm:nestedBoundTo
<<InputInstace>>
<<Pipeline>> <http://.../instance/pipeline>
<<InputInstace>>
<<SparqlTransformer>> <http://.../instance/sparql/ovm>
ldvm:member
<<SparqlTransformer>> <http://.../instance/sparql/pos>
ldvm:member
<<OutputInstace>>
<<OutputInstace>>
ldvm:member
ldvm:boundTo
ldvm:boundTo
<<InputInstace>>
<<Union>> <http://.../instance/union>
<<OutputInstace>>
ldvm:nestedBoundTo
<<OutputDataSample>>
<<OutputTemplate>> <http://.../output/1>
ldvm:outputDataSample

Fig. 3. Analyzer template containing nested pipeline

See Figure 3 where we chose a graphical representation rather than a textual one where boxes are entities, the class of the entities is written in bold and the actual URI of the entity is shortened. The new analyzer template has two inputs and one output and contains the nested pipeline. There is a link to a new output data sample from the output. There are two instances of the SPARQL analyzer template (see subsection 4.1), which transform the data from the individual inputs, their inputs are bound to them using the `ldvm:nestedBoundTo` property. The third member of the pipeline has one input bound to the output of the SPARQL analyzers and one output bound to the output of the template itself.

At the same time, Figure 3 is an example of a very simple pipeline instance, which is the one nested in the new component template. What is missing due to lack of space is the instance configuration of a component, which can overwrite the one specified at template level. The configuration itself, however, looks the same at both levels and depends completely on the component being configured.

5 Related Work

The problem of Linked Data not being accessible to non-experts is well-known. With the LDVM Vocabulary we aim at an open web-services like environment that is independent of the specific implementation of the LDVM components. This of course requires proper definition of interfaces and the LDVM vocabulary is the base for that. However, the approaches so far usually aim at a closed browser environment that is able to analyze and visualize the Linked Data Cloud similarly to our first version of Payola [6]. They do not provide configuration and description using a reusable vocabulary. The approaches include *Hide the stack* [5], where the authors describe a browser meant for end-users, which is based on templates based on SPARQL queries. Also recent is *LDVizWiz* [1], which is a very LDVM-like approach to detecting categories of data in SPARQL endpoints and extracting basic information about entities in those categories. An lightweight application of LDVM in enterprise is described in LinDa [9]. Yet another similar approach that analyzes SPARQL endpoints to generate faceted browsers is *rdf:SynopsViz* [3]. In [2] the authors use their *LODeX* tool to summarize LOD datasets according to the vocabularies used. For more tools for Linked Data visualization see [7]. The most relevant related work to the specific topic of a vocabulary supporting Linked Data visualization is Fresnel - Display Vocabulary for RDF [8]. Fresnel specifies how a resource should be visually represented by Fresnel-compliant tools like LENA [8] and Longwell [9]. Therefore, Fresnel vocabulary could be perceived as a vocabulary for describing LDVM visualization abstraction. This is partly because the vocabulary was created before the Linked Data era and therefore focuses on visualizing RDF data without considering vocabularies and multiple sources.

6 Conclusions

In this paper we briefly described our Linked Data Visualization Model (LDVM) and proposed a Linked Data vocabulary for description of its components and their configuration. The vocabulary supports description of inputs and outputs of individual components, which allows LDVM implementations to check whether components are compatible with each other. In addition, the vocabulary supports creation of new LDVM compatible

[8] `https://code.google.com/p/lena/`

[9] `http://simile.mit.edu/issues/browse/LONGWELL`

component templates and representation of analytic and visualization pipelines based on those components. This support includes creation of component templates from pipeline instances, which facilitates cooperation between expert and non-expert users of LDVM implementations. Expert users can create complex pipelines and provide them as black box components to the non-experts who can then use them in their pipelines. We showed the vocabulary usage on an example of a component template and example of a nested pipeline. There are multiple advantages of representing the templates, their configuration and whole pipelines in RDF according to the LDVM vocabulary. For example, the data processed by the pipelines can be linked to the actual pipelines, the templates and pipelines can be easily manipulated by SPARQL queries and shared among users.

References

1. G. A. Atemezing and R. Troncy. Towards a linked-data based visualization wizard. In *ISWC 2014, 5th International Workshop on Consuming Linked Data (COLD 2014), 20 October 2014, Riva del Garda, Italy*, Riva Del Garda, ITALY, 10 2014.
2. F. Benedetti, S. Bergamaschi, and L. Po. Online Index Extraction from Linked Open Data Sources. In A. L. Gentile, Z. Zhang, C. d'Amato, and H. Paulheim, editors, *Proceedings of the 2nd International Workshop on Linked Data for Information Extraction (LD4IE)*, number 1267 in CEUR Workshop Proceedings, pages 9–20, Aachen, 2014.
3. N. Bikakis, M. Skourla, and G. Papastefanatos. rdf:SynopsViz – A Framework for Hierarchical Linked Data Visual Exploration and Analysis. In V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai, editors, *The Semantic Web: ESWC 2014 Satellite Events*, Lecture Notes in Computer Science, pages 292–297. Springer International Publishing, 2014.
4. J. M. Brunetti, S. Auer, R. García, J. Klímek, and M. Nečaský. Formal Linked Data Visualization Model. In *Proceedings of the 15th International Conference on Information Integration and Web-based Applications & Services (IIWAS'13)*, pages 309–318, 2013.
5. A.-S. Dadzie, M. Rowe, and D. Petrelli. Hide the Stack: Toward Usable Linked Data. In G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. De Leenheer, and J. Pan, editors, *The Semantic Web: Research and Applications*, volume 6643 of *Lecture Notes in Computer Science*, pages 93–107. Springer Berlin Heidelberg, 2011.
6. J. Klímek, J. Helmich, and M. Nečaský. Payola: Collaborative Linked Data Analysis and Visualization Framework. In *10th Extended Semantic Web Conference (ESWC 2013)*, pages 147–151. Springer, 2013.
7. J. Klímek, J. Helmich, and M. Nečaský. Application of the Linked Data Visualization Model on Real World Data from the Czech LOD Cloud. In C. Bizer, T. Heath, S. Auer, and T. Berners-Lee, editors, *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014.*, volume 1184 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.
8. E. Pietriga, C. Bizer, D. R. Karger, and R. Lee. Fresnel: A Browser-Independent Presentation Vocabulary for RDF. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 158–171. Springer, 2006.
9. K. Thellmann, F. Orlandi, and S. Auer. LinDA - Visualising and Exploring Linked Data. In *Proceedings of the Posters and Demos Track of 10th International Conference on Semantic Systems - SEMANTiCS2014*, Leipzig, Germany, 9 2014.

Parallel Approach to Context Transformations

Michal Vašinek, Jan Platoš

Department of Computer Science, FEECS, VŠB-Technical University of Ostrava,
17.listopadu, 708 33, Ostrava - Poruba,
`michal.vasinek@vsb.cz,jan.platos@vsb.cz`,

Abstract. Context transformation is a process that turns input data into one with lower zero order entropy. The main weakness of algorithms presented sofar is the complexity of replacing procedure. In this paper we describe properties related to the parallelization of replacing procedure and we propose a modified version of a basic context transformation algorithm, that uses a parallel approach.

Keywords: compression, context, transformation, entropy, parallel computing

1 Introduction

There are two main classes of algorithms employed in the data compression. The first class of algorithms deals directly with the compression and their purpose is to decrease the size of the input message. Examples [3] of such algorithms are Huffman coding, Arithmetic coding, Lempel Ziv algorithms family, PPM and many others. The second class of algorithms behaves more like preprocessors for the first class, these algorithms are usually called transformations, examples are Burrows-Wheeler transformation [1] or MoveToFront [2] transformation that are used in bzip2 file compressor.

The purpose of transformations is not to decrease message size but to change the internal message structure that could be more easily handled by some of the first class algorithms. In [5] and [6] we propose a reversible transformation method called a 'Context transformation', that we use to reduce zero-order entropy of input messages. Transformed data are then compressed using entropy coding algorithms, like Huffman coding. In this paper we describe properties related to the parallelization of context transformation algorithms.

We use several notation conventions: we use Σ to denote set of message symbols, characters in equations are greek symbols from Σ and unless stated otherwise, they can represent any member of Σ, i.e. $\alpha = \beta$ as well as $\alpha \neq \beta$ it should be clear from the context. When we present examples of transformations we use symbols from english alphabet to denote particular transformations like $ab \rightarrow ac$, then each character $a, b, c, \ldots$ are distinct characters $a \neq b \neq c \neq \ldots$.

The rest of the paper is organized as follows. Section 2 contains description of the proposed Context transformations and their properties. Section 3 analyses complexity of the transformation with respect to the number of symbols in the alphabet for both proposed transformation types. Section 4 describes the ability of the transformations to work in parallel. Section 5 contains design of the parallel algorithm and presents results achieved on the selected dataset. Last Section 6 concludes the paper and discusses achieved results.

2 Context Transformations

The context transformation is a method that takes two digrams, the digram $\alpha\beta$ that is present in the input message and the digram $\alpha\gamma$ that is not present in the input message. Transformation replaces all occurences of $\alpha\beta$ for $\alpha\gamma$. The symbol α is called a context symbol and it is present in both digrams.

Definition 1 *Context transformation(CT) is a mapping* $CT(\alpha\beta \rightarrow \alpha\gamma, w) : \Sigma^n \rightarrow \Sigma^n$*,* Σ *is the alphabet of the input message* w *and* n *is the length of the input message, that replaces all digrams* $\alpha\beta$ *for digram* $\alpha\gamma$*, where* $p(\alpha, \gamma) = 0$ *and* $\beta \neq \gamma$*.*

We may consider also the transformation of the form $\beta\alpha \rightarrow \gamma\alpha$, such transformation would correspond to the same transformation like in Definition 1 if we perform replacement on the mirror message. There is one rule that must be followed and it is that such transformation must be applied on the input message from right to left, respectively from left to right in the case of $\beta\alpha \rightarrow \gamma\alpha$. This rule ensures that for each context transformation an inverse transformation that transforms the new message back to the input message exists. The proof of this rule can be found in [5] and [7]. Context transformations are a subset of more general set of generalized context transformations:

Definition 2 *Generalized context transformation(GCT) is a mapping* $GCT(\alpha\beta \leftrightarrow \alpha\gamma, w) : \Sigma^n \rightarrow \Sigma^n$*,* Σ *is the alphabet of the input message* w *and* n *is the length of the input message, that exchanges all digrams* $\alpha\beta$ *for digram* $\alpha\gamma$ *and vice-versa.*

GCT transformation can be applied in any direction, but $CT \subset GCT$ only when they are applied in the same direction. In this paper we describe GCT transformations applied from right so it is consistent with the former notion of context transformations.

The main difference between a context and a generalized context transformation is that we can swap any two digrams beginning with *alpha*, hence we are no more restricted on cases when one of digrams is not present in a message. Example of each transformation is presented in Fig. 1.

The following paragraphs contains brief discussion of the context transformation process and its implication to zero order Shanonn entropy [4]. The reader can find the detailed description in [5] and [7]. When we will speak about entropy we mean the Shannon's entropy over alphabet Σ of individual symbols defined by:

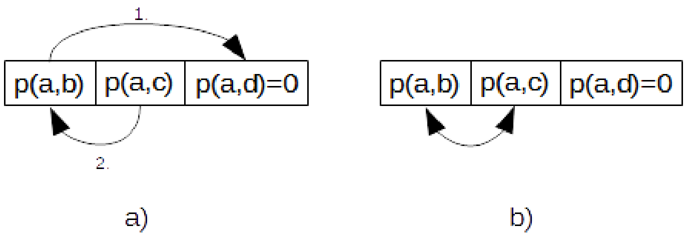


Fig. 1. Examples of two different types of transformations, $p(b) > p(c) > p(d)$, $p(a,c) > p(a,b)$, $p(b,c) > p(a,b)$: a) Sequence of context transformations $ab \rightarrow ad, ac \rightarrow ab$, b) Generalized context transformation $ab \leftrightarrow ac$

$$H = -\sum_{x \in \Sigma} p(x) log(p(x)) \tag{1}$$

where $p(x)$ is probability of symbol x. Context transformations are based on the fact that the structured data, like a human-written texts or programming codes has its inner structure and the most of the information is not hidden in the entropy of input data alphabet but it depends more on occurences of digrams, trigrams, words, ...

Suppose example string 'kakaoo', frequencies of its digrams can be represented by a matrix, we call such matrix a 'Context matrix'. Entry of a context matrix is non-negative integer that represents frequency of digram represented by row symbol followed by column symbol. For our example string, the context matrix is shown in Table 1.

	k	a	o
k	0	2	0
a	1	0	1
o	0	0	1

Table 1. The context matrix that represents the string 'kakaoo'

Since the probability distribution of symbols is uniform, the entropy of our example string is for given alphabet maximal. From context matrix we see that there are several accessible transformations, first we select a context transformation that replaces all digrams 'ka' for a digram 'kk', so the condition of zero and non-zero entry is fulfilled.

The resulted string is 'kkkkoo' and its context matrix is shown in Table 2. We can see two different consequences of this transformation:

– the alphabet size decreased,

– the entropy decreased.

	k	a	o
k	3	0	1
a	0	0	0
o	0	0	1

Table 2. The context matrix that represents the transformed string 'kkkkoo'

When we use a generalized context transformation instead of a simple context transformation we arrive in two different strings based on which transformation direction was applied. When GCT is applied from left then $GCT_{\rightarrow}(w) = kkaaoo$ and from right like in a context transformation case $GCT_{\leftarrow}(w) = kkkkoo$.

We may select another transformation, for example 'oo' for 'ok', leaving w in the state 'kkkkok', that has even lower entropy. What we examine in this paper is, if we can run these two transformations simultaneously, respectively under what conditions these can be perfomerd in parallel.

2.1 Transformations and their consequences on Huffman coding

Compression schema based on CT or GCT consist of two phases, in the first phase the input message is transformed using (G)CT, and finaly in the second phase, the message is compressed using some zero-order entropy coder.

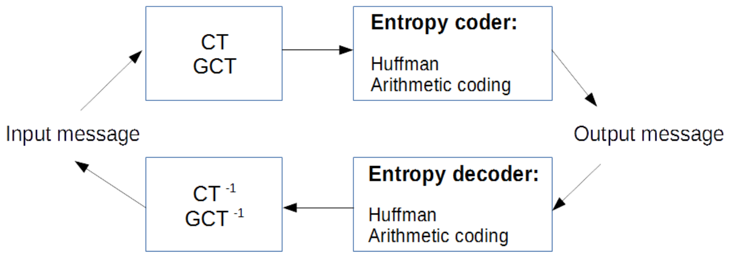


Fig. 2. Compession schema based on context transformations.

Suppose static Huffman coding as a zero-order entropy coder, if $p(\beta) > p(\gamma)$ then for lengths of their corresponding Huffman codes holds that $|c(\beta)| \leq |c(\gamma)|$. If $p(\alpha, \gamma) \neq 0$ and $p(\alpha, \beta) = 0$ (resp. $p(\alpha, \beta) < p(\alpha, \gamma)$) and transformation $CT(\alpha\gamma \rightarrow \alpha\beta)$ (resp. $GCT(\alpha\beta \leftrightarrow \alpha\gamma)$) is applied, then all symbols γ that are part of digrams $\alpha\gamma$ will be coded by the code of length $|c(\beta)|$ instead of the code of length $|c(\gamma)|$.

3 Complexity analysis

Presented algorithms are all greedy approaches, algorithms iteratively search for the best entropy reducing transformations and then apply transformation on the source. Formally we divide each iteration in two steps:

1. Search procedure - find the best transformation according the actual context matrix,
2. Replacing procedure - apply selected transformation on the input string w.

```
n ← input size
w ← input string
compute_matrix()
repeat
    transformation = search_procedure()
    H0 = entropy()
    H1 = entropy(transformation)
    if $n * (H0 - H1) < LIMIT$ then
        return
    end if
    replacing_procedure(transformation,w)
until true
```

Fig. 3. Basic structure of context transformation algorithms

The infinite loop in Fig. 3 terminates when there are no more transformations that are able to reduce entropy more, than by a limit given by $LIMIT$ variable. The $LIMIT$ variable is set up to the size of transformation description. In our algorithms $LIMIT = 24$ because we store description of each transformation as a triplet of byte values $\alpha\beta\gamma$.

Both parts of the algorithm can be parallelized, but the operation that is the most time consuming is the replacing procedure as is depicted in Table 3. In the Section 5 we present a modified version of the basic algorithm, that uses the fact, that some transformations can be performed simultaneously and we try to parallelize the replacing procedure.

Search(ms)	Replacement(ms)
1.144	28.813

Table 3. Average time per file needed for search and replacement procedures. Dataset assasin(6097 files).

In the next sections we analyse complexities of both parts and we show that for source string w of length $|w| \to \infty$ the only significant part remains to be replacing procedure.

3.1 Search procedure

Algorithm in Fig. 3 is a basic greedy algorithm that searches for and eventually performs context transformations in such a way that the resulted entropy of alphabet Σ is lower than in the untransformed case Search procedure operates on the context matrix, where rows and columns are sorted according their frequencies. The algorithm iterates through rows of the matrix from the most probable one to the least probable one.

The simplified version of the search procedure is presented in the following pseudocode:

```
$cm \leftarrow context\ matrix$
$\Delta H = 0 \leftarrow the\ change\ of\ zero\ order\ entropy$
$T \leftarrow searched\ transformation$
for $row = 1$ to $dim(cm)$ do
  for $col\_1 = 1$ to $dim(cm)$ do
    for $col\_2 = col\_1 + 1$ to $dim(cm)$ do
      $\Delta H_{temp} = compute\_dH(row, col\_1, col\_2)$
      if $\Delta H_{temp} - \Delta H < 0$ then
        $\Delta H = \Delta H_{temp};$
        $T = [row, col\_1, col\_2]$
      end if
    end for
  end for
end for
```

Fig. 4. Outline of the search procedure

The function *compute_dH* has constant complexity because it computes only change of entropy, probabilities will be modified only for symbols β and γ, so we don't need to recompute entropy of all symbols, but is is sufficient to recompute entropies of β and γ. The complexity of the search procedure is dependent only on the size of the alphabet and its worst case complexity is $O(t|\Sigma|^3)$, because we have to perform $1/2|\Sigma|^3$ searches. In our algorithm design the maximum allowable size of the alphabet is $|\Sigma| = 256$, since we interpret as a letter only a one byte values. That concludes that the upper limit of operations needed for one search is fixed and in the worst case it is given by $|\Sigma|^3$.

There are several techniques how the search procedure can be designed with lower complexity, i.e. precomputation and storage of n best transformations at the beginning and then only updating the modified ones, leads to the number of entropy recomputation given by $|\Sigma|^3 + 2t|\Sigma|^2$.

3.2 Replacing procedure

The second step, the replacing procedure is very simple, it passes data t times and exchanges both digrams. Let t be a number of performed GCTs and n is the length of the input, then the complexity of replacing procedure is $O(tn)$.

The inverse transformation consists only from replacing procedure and so it also has the complexity $O(tn)$. If the generalized context transformation is of the form $\alpha\beta \leftrightarrow \alpha\gamma$ then its inverse transformation is of the same form, but is taken in the opposite direction. The experimentally observed t was in range 100-1000 of transformations.

3.3 Comparison

When we let the complexitities functions to be equal we arrive at the limit when one computation of a replacement procedure becomes more significant than the one computation of a search procedure. We have to arrive at the limit because the search procedure is independent of the input size:

$$c_1 tn = c_2|\Sigma|^3 + c_3 t|\Sigma|^2 \tag{2}$$

Constants c_i represents implementation specific coefficients. The number of transformations t on both sides of the equation can be rearranged leaving us with:

$$n = \frac{c_2|\Sigma|^3 + c_3 t|\Sigma|^2}{c_1 t} = \frac{c_2|\Sigma|^3}{c_1 t} + \frac{c_3|\Sigma|^2}{c_1} \tag{3}$$

as a number of transformations $t \to \infty$ the first term on the right side becomes zero:

$$\lim_{t\to\infty} n = \frac{c_3|\Sigma|^2}{c_1} \tag{4}$$

when $|\Sigma|$ is finite, then for all source strings of length m, where $m > n$, the replacing procedure will be more computationally intensive than the search procedure.

$$c_1 tm > c_2|\Sigma|^3 + c_3 t|\Sigma|^2 \tag{5}$$

4 Parallel transformations

In this section we describe conditions needed for parallelization of the context and generalized context transformations.

4.1 Commuting transformations

Let's consider two different transformations T_1 and T_2, we say that these two transformations commute, if they satisfy following definition:

Definition 1. *Two different transformations T_1 and T_2 commute if:*

$$T_1(T_2(w)) = T_2(T_1(w)) \tag{6}$$

In our model example of the string $w = kakaoo$ the two presented transformations commute, since if T_1 is 'ko' to 'kk' transformation and T_2 is 'oo' to 'ok' transformation then $T_2(T_1('kakaoo')) = T_1(T_2('kakaoo')) = kkkkok$.

As an example of non-commuting transformations let's consider transformation T_1 again and transformation T_3 that replaces digrams 'ao' to 'aa'. Applying these two transformations in both ways will lead to different results. $T_3(T_1(w)) =' kkkkao'$ but $T_1(T_3(w)) =' kkkkoo'$ so we see that $T_1(T_3(w)) \neq T_3(T_1(w))$

4.2 Parallel transformations

Commutativity is not sufficient property to decide if two transformations could be run in parallel. As may be seen along with property that they have an inverse transformation to them.

Let's consider again our example word $w = kakaoo$ and two transformations $T_{4,}$ representing $ak \rightarrow aa$ and T_5 representing $ao \rightarrow aa$. These two transformations commute and transform together the word 'kakaoo' into the word 'kaaaao', but when we perform inverse transformation, we can get different results, since we don't know which of two inverse transformations will replace digram 'aa' first.

Before we show how to handle inverse transformations, we introduce two sets, an abstract set D_i and set K_i, that will be later used to prove a theorem about parallel transformations:

Definition 2. *Let D_i be a set of all unique digrams that are modified(created or destroyed) by transformation $T_i(\alpha\beta \leftrightarrow \alpha\gamma)$ and let the set K_i be a set of all transformation digrams $K_i = \{\alpha\beta, \alpha\gamma\}$.*

The set D_i contains digrams $\alpha\beta$, $\alpha\gamma$ and all digrams of type βX and γX, where $X \in \Sigma$. Suppose two sets D_1 and D_2 and let $D_1 \cap D_2 \neq \emptyset$, these two sets share at least one common digram d, suppose that $d \in K_1 \cup K_2$, it means that transformations T_1 and T_2 will interfere on this particular digram i.e. when transformation T_1 modifies digram $\alpha\beta$ on $\alpha\gamma$ then the second transformation won't be able to modify digram $\alpha\beta$ as it would do in the case when there is no other transformation T_1. From the above reasoning we form a lemma about parallel transformations:

Lemma 1. *Two transformations T_1 and T_2 can be run in parallel if:*

$$D_1 \cap D_2 = \emptyset \tag{7}$$

Proof. Since $D_1 \cap D_2 = \emptyset$ then no digrams that are modified by both transformations T_1 and T_2 exist and so these two transformations can be applied together. □

Lemma 1 gives us a simple condition that we use in our algorithms to construct a set of parallel transformations, but it is a weak condition, because there still exists parallel transformations, but $D_1 \cap D_2 \neq \emptyset$, i.e. suppose $T_1(ab \leftrightarrow ac)$ and $T_2(db \leftrightarrow dc)$, for these two transformations $D_1 \cap D_2 = \{bX, cX\}$.

Theorem 1. *Two transformations T_1 and T_2 can be run in parallel if:*

$$D_1 \cap K_2 = \emptyset \tag{8}$$

and

$$D_2 \cap K_1 = \emptyset \tag{9}$$

Proof. Suppose that transformation T_1 is of the form $T_1(\alpha\beta \leftrightarrow \alpha\gamma)$, it has correspoding sets $D_1 = \{\alpha\beta, \alpha\gamma, \beta X, \gamma X\}$ and $K_1 = \{\alpha\beta, \alpha\gamma\}$.

If T_2 contains in its K_2 one of the elements from D_1 then it means that the transformation T_1 can modify some of the elements that would be otherwise transformed(replaced) by T_2, so the two transformations can be run in parallel only if $D_1 \cap K_2 = \emptyset$.

Similar reasoning can be used to prove theorem for equation (9). If T_2 contains in its D_2 one of the elements from K_1, i.e. $\alpha\beta$ then if T_2 will change any occurence of $\alpha\beta$ first, the first transformation won't be able to modify it to $\alpha\gamma$, so such transformations cannot be parallelized and for parallel transformations must hold $D_2 \cap K_1 = \emptyset$.

Now suppose that $D_1 \cap K_2 = \emptyset$, but $D_2 \cap K_1 \neq \emptyset$, then some element i.e. $\alpha\beta$ is in the set D_2, we know that transformations are not parallel when $\alpha\beta \in K_1 \wedge \alpha\beta \in K_2$, now we prove that they cannot be parallelized also if $\alpha\beta \in D_2 \backslash K_2$, bacause then T_2 is of the form $X\alpha \leftrightarrow XY$, but when α is modified on Y then instead of $\alpha\beta$ will be $Y\beta$ and transformation T_1 cannot modify it. The same is valid in the case when $D_1 \cap K_2 \neq \emptyset$, but $D_2 \cap K_1 = \emptyset$. So both conditions in Theorem 1 must be valid together. $\square$

Because our parallel algorithm is based on Lemma 1, we show several other properties that parallel transformations based on Lemma 1 have.

Theorem 2. *Two different transformations T_1 and T_2 can be run in parallel if:*

$$T_1(T_2(w)) = T_2(T_1(w)) = w_T \tag{10}$$

and

$$T_1^{-1}(T_2^{-1}(w_T)) = T_2^{-1}(T_1^{-1}(w_T)) = w \tag{11}$$

Proof. Suppose the transformation $T_{\leftarrow}(\alpha\beta \leftrightarrow \alpha\gamma)$, where the arrow at the index is a label for transformation direction i.e. from right to left $\leftarrow$; it has its coresponding set of modified digrams $D_{\leftarrow} = \{\alpha\beta, \alpha\gamma, \beta X, \gamma X\}$, next suppose the transformation $T_{\rightarrow}(\alpha\beta \leftrightarrow \alpha\gamma)$ and its coresponding set of digrams $D_{\rightarrow} = \{\alpha\beta, \alpha\gamma, \beta X, \gamma X\}$, we see that $D_{\leftarrow} = D_{\rightarrow}$, but we know that $T_{\rightarrow}(T_{\leftarrow}(w)) = w$, so transformations T and T^{-1} share the same set D.

If $T_1(T_2(w)) = T_2(T_1(w)) = w_T$ then it means that there are no interfering digrams, because if there would be such digrams then the equality would not hold, so $D_{\leftarrow,1} \cap D_{\leftarrow,2} = \emptyset$, but we showed that $D_{\leftarrow} = D_{\rightarrow}$ so also intersect of inverse transformations sets is empty and T_1^{-1} and T_2^{-1} will also recover w correctly. □

Lemma 1 and Theorem 2 describes the same phenomenom and they can be generalized for the arbitrary set of parallel transformations:

Theorem 3. *The set of transformations T can be run in parallel if for all pairs of transformations T_i and T_j holds that:*

$$D_i \cap D_j = \emptyset \tag{12}$$

Proof. Suppose the set T can be run in parallel, and suppose two transformations $T_i, T_j \in T$ such that $D_i \cap D_j \neq \emptyset$, then it means that there exist some digram $\alpha\beta$ common for T_i and T_j and these transformations cannot be run in parallel, but this is in contradiction with hypothesis that T is parallel, so the set $D_i \cap D_j = \emptyset$. □

There is one important fact to emphasize that emerged as a consequence of Theorem 2, it is a statement about inverse transformations:

Corollary 1. *Let $T = \{T_1, T_2, \ldots, T_n\}$ is a set of parallel transformations, then the set $T^{-1} = \{T_1^{-1}, T_2^{-1}, \ldots, T_n^{-1}\}$ is parallel as well.*

Proof. In Theorem 2 we proved that $D_i = D_i^{-1}$ and because for the set T holds that for all sets D_i, D_j is $D_i \cap D_j = \emptyset$, then also $D_i^{-1} \cap D_j^{-1} = \emptyset$ and the transformation set T^{-1} can be run in parallel.

Corollary 1 is very important result because it tells us that we may parallelize not only the set T but also its inverse T^{-1} so an inverse transformation algorithm is parallelizable as well.

With the knowledge of Lemma 1 we know how to construct set T, now we explore how large the set possibly can be for particular alphabet Σ:

Theorem 4. *The maximal size M_T of the set of parallel transformations T for particular alphabet Σ is:*

$$M = \lfloor \frac{|\Sigma|}{2} \rfloor \tag{13}$$

Proof. There are two basic types of the set D, one type coresponds to the transformation of the form $\alpha\alpha \leftrightarrow \alpha\beta$ and the second type coresponds to the transformation $\alpha\beta \leftrightarrow \alpha\gamma$. The first set $D_1 = \{\alpha\beta, \alpha\alpha, \alpha X, \beta X\} = \{\alpha X, \beta X\}$ influences two rows of the context matrix meanwhile the second set $D_2 = \{\alpha\beta, \alpha\gamma, \beta X, \gamma X\}$ influences three rows. So when only transformations of the first type are selected into T then at most $\lfloor |\Sigma|/2 \rfloor$ transformations can be run in parallel. □

Theorem 5. *Relation to be parallel between transformations is not transitive.*

Proof. Suppose a sets of digrams $D_{i\in\{0,1,2\}}$. Let $D_0 \cap D_1 = \emptyset$ and $D_1 \cap D_2 = \emptyset$, if transitivity holds then $D_0 \cap D_2 = \emptyset$, but this is not generally true. Consider following example, let $T_0(ab \leftrightarrow ac)$,$T_1(ad \leftrightarrow ae)$, transformations are clearly parallelizable. Let $T_2(ac \leftrightarrow af)$, $D_1 \cap D_2 = \emptyset$, but $D_0 \cap D_2 \neq \emptyset$. □

5 Parallel version of the basic context transformation algorithm

As we saw in the section about parallel transformations, there is a distinct number of rows affected by each transformation. This knowledge allows us, based on Lemma 1, collect parallel transformations in the first step and afterwards perform them simultaneously. Individual transformations are perfomed simultaneously in shared memory. The process is outlined in Fig.5.

```
$cm \leftarrow$ $context\ matrix$
$t\_array = [] \leftarrow parallel\ transformations$
$t\_size \leftarrow array\_size$
for $row = 1$ to $dim(cm)$ do
  for $col = 1$ to $dim(cm)$ do
    $i = 0$
    while $parallel\_transformation(t\_array, row, col)$ and $i < t\_size$ do
      $append\_transformation(t\_array, get\_transformation())$
      $i = i + 1$
    end while
    for $transformation$ in $t\_array$ do
      $perform\_parallel(transformation)$
    end for{This section is run in parallel}
  end for
end for
```

Fig. 5. Parallel modification of basic transformation algorithm

We tested algorithm on the computer machine equipped by four proccessors. Parallel algorithm was more then three times faster than the serial one. The results are shown in Table 4.

Table 4. Average processing time per file. Dataset assasin.

Serial(ms)	Parallel(ms)	Speed-up
45.383	13.397	3.387

Serial and parallel algorithms can have different transformation paths(i.e. different transformations or the order in which transformations are performed).

In the serial version of the algorithm, there can be also performed transformations inaccessible to the parallel algorithm, so the resulted entropy is lower in the serial case as is presented in Table 5.

No transformation	Serial	Parallel
4.938	4.054	4.106

Table 5. Entropy comparison. Dataset assasin.

6 Conclusion

We showed a basic properties that have to be fulfilled to run replacing procedure of context transformation algorithms in parallel. The presented parallel version of the basic context transformation algorithm significantly increases the speed of processing of individual data files. On the other hand, usage of parallel algorithm can lead to minor increase of the resulted entropy.

There are two directions in which we would like to continue our research, the first direction is to prepare parallel algorithms according Theorem 1 instead of Lemma 1 and since algorithms presented sofar performs transformations using only digrams, in the future work we will also focus on development of algorithms operating on different context lengths.

Acknowledgment

This work was supported by the SGS in VSB - Technical University of Ostrava, Czech Republic, under the grant No. SP2015/146.

References

1. Burrows, M. and Wheeler D.J., *A block sorting lossless data compression algorithm*, 1994
2. Bentley, J, L. and Sleator, D. D. and Tarjan, R. E. and Wei, Victor K. *A locally adaptive data compression scheme* Commun. ACM, vol 4, pp. 320-330, 1986
3. Salomon, D., *Data Compression: The Complete Reference*, Springer, NewYork, 2007.
4. Shannon, C. E., *A mathematical theory of communication* Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, 1948.
5. Vašinek, M., *Kontextové mapy a jejich aplikace* Master Thesis, VŠB - Technical University of Ostrava, 2013
6. Vašinek, M. and Platoš, J., *Entropy reduction using context transformations* Data Compression Conference(DCC), pp. 431-431, 2014
7. Vašinek, M., *Context Transformations* Ph.D. Workshop of Faculty of Electrical Engineering and Computer Science, 2014

Methodologies and best practices for Open Data publication

Jan Kučera[1,2], Dušan Chlapek[1], Jakub Klímek[2], Martin Nečaský[2]

[1] University of Economics, Prague, Czech Republic
{jan.kucera, chlapek}@vse.cz
[2] Charles University in Prague, Czech Republic
{klimek, necasky}@ksi.mff.cuni.cz

Abstract. Publication and reuse of machine-readable data on the Web is one of the current trends in data management that is mainly manifested by the Open Data movement. This movement is especially strong in the government and public sector domain where many Open Government Data initiatives have been launched in a large number of countries across the globe. In the European Union the recent update of the PSI Directive aims at fostering the reuse of data and information held by the public sector bodies by promoting publication of data in open machine-readable formats together with the relevant metadata. Even though the support of governments and the EU to Open Data and PSI reuse seems to be strong, public sector bodies are facing many challenges when publishing Open Government Data and the desired reuse is not always evident. In order to overcome these challenges Open (Government) Data publication methodologies are being proposed and the best practices in this domain are being formulated. In this paper we discuss the current challenges related to the OGD publication and reuse, we provide an overview of the existing methodologies and the best practices for publication of Open Government Data, we present an OGP publication methodology developed in the COMSODE project.

1 Introduction

With many Open Government Data initiatives being executed in a large number of countries across the globe (see for example [36]) and the recent update of the PSI Directive [8] we can see a significant shift towards provision of data held by the public sector bodies in machine readable formats together with the relevant metadata.

According to [23] Open Data is "*data that can be freely used, reused and redistributed by anyone – subject only, at most, to the requirement to attribute and sharealike.*" In this paper we refer to Open Data published by public sector bodies (PSBs) on the web in machine-readable formats as Open Government Data (OGD).

Open Government Data promise significant benefits that can range from increased efficiency and effectiveness of the public sector bodies to greater trust and improved transparency [25]. Significant economic impacts are expected from the reuse of OGD as well. However current studies often provide only estimates and there is still lack of empirical evidence [30]. Even though the support of the top management is a necessary prerequisite of a successful OGD initiative it does not guarantee the reuse of the released data [25]. One of the reasons might be that the public sector bodies sometimes view the OGD from different perspective than the potential users [32].

It is evident that there are many issues related to publication and reuse of OGD. In order to help the involved stakeholders to deal with these issues methodologies and best practice guidelines for OGD publication have been developed or are currently under development. In this paper we discuss the current challenges related to the OGD publication and reuse and we provide an overview of some of the current methodologies proposing the best practices of the OGD publication. We also present the *Methodology for publishing datasets as open data* [18] developed in the COMSODE project which tries to address some of the known problems in this domain and we introduce two projects in that this methodology is utilized.

This paper is structured as follows. This introduction is followed by a section discussing the current problems and issues related to the OGD publication. Next examples of the existing OGD publication methodologies are presented. *Methodology for publishing datasets as open data* is introduced in the next section. Conclusions are presented at the end of this paper.

2 Challenges of the OGD publication

Public sector bodies are facing a number of challenges when publishing Open Government Data. Some of the challenges might further hinder reuse of the data. For example unclear licensing of datasets might prevent the re-users from developing sustainable business models on top of the published data. Both Ubaldi [30] and Janssen, Charalabidis and Zuiderwijk [12] provide a comprehensive discussion of the challenges in the OGD domain. Kučera and Chlapek [14] point out that there are not only benefits that could be reaped out of the OGD reuse but there are also risks that need to be mitigated.

Table 1 summarizes the current challenges related to the OGD publication discussed in the literature. We classify the challenges into the following groups:

- *Political and social challenges (SOC)* – challenges related to the political support, decision making and social problems;
- *Economic challenges (ECO)* – challenges and problems related to benefits and costs of OGD and to its measurement; potential problems related to the financing of the OGD initiatives belong to this group too;
- *Organizational challenges and challenges related to the internal processes (ORG)* – problems related to the organizational structures and the internal processes through which the OGD are delivered by the PSBs;
- *Legal challenges (LEG)* – problems related to the legal openness of OGD as well as the legislative issues;
- *Technical challenges (TCH)* – issues and challenges related to the technology, data formats or infrastructure needed to publish OGD.

Table 1. Challenges related to the OGD publication.

ID	Problem/issue	Group	Refs.
CL1	Too many OGD initiatives – users in Netherland sometimes feel frustrated by too many OGD initiatives.	SOC	[12]
CL2	Misinterpretation or contradictory conclusions – different users might draw different conclusions out of the data or the data might be misinterpreted.	SOC	[14], [12]
CL3	Provided feedback might not always have the necessary level of quality to be used for improvements.	SOC	[12]
CL4	Some of the published datasets have little value for the users or the possible use is not always obvious.	ECO	[12]
CL5	Some PSBs seek benefits for themselves rather than the benefits to the society.	ECO	[12]
CL6	Fees might represent a barrier to the re-use. However some PSBs are required to sell data to cover their costs.	ECO	[30], [12]
CL7	Not enough resources, especially in case of the small public sector bodies	ECO	[12]
CL8	No systematic OGD cost measurement	ECO	[30], [17]
CL9	No systematic OGD benefits assessment	ECO	[17]
CL10	No standard process or policy for the OGD publication. Responsible persons might not always know how to proceed with the OGD publication.	ORG	[30], [13], [12]
CL11	Lack of interaction between OGD users and publishers – PSBs not always respond to the provided feedback or questions of the users. There might be lack of the appropriate processes and tools [12].	ORG	[30], [12], [29]
CL12	There is not always a centralized OGD portal available to the PSBs.	ORG	[12]
CL13	Publication of OGD requires an appropriate structure of processes, roles and responsibilities. However these are not always in place and setting up the right organizational structure requires significant effort.	ORG	[30]
CL14	Published datasets are in many cases not regularly updated and thus the provided data might be obsolete or non-valid.	ORG	[29], [12]
CL15	There is a risk of violation of protection of the personal information or other protected information when publishing OGD. Concerns about the possible violation of legislation acts as a barrier to the OGD publication.	LEG	[12], [17]
CL16	Published datasets have missing, unclear or restrictive terms of use. This results in legal uncertainty of the potential users.	LEG	[30], [12]

ID	Problem/issue	Group	Refs.
CL17	Same or similar datasets do not always share the same format or schema.	TCH	[30], [12]
CL18	Sometimes users need to register to access data. Such practice is seen as discriminatory by [28].	TCH	[12]
CL19	Published data does not represent the primary data but only processed data.	TCH	[12]
CL20	Quality of the published data is often not good enough. Common data quality issues are related to the accuracy, completeness and timeliness of the data.	TCH	[12], [29]
CL21	It is difficult to find the required data.	TCH	[30], [12], [29]
CL22	Missing description of the data formats and schemas. Missing explanation of the data. Missing standards.	TCH	[12]
CL23	In some cases it might be difficult to publish OGD due to the underlying ICT infrastructure (e.g. in case of the "legacy" applications).	TCH	[30], [12]
CL24	Lack of suitable software tools for OGD publication.	TCH	[12]

List of the challenges related to the OGD publication presented in the table 1 is by no means comprehensive. Although some of the problems discussed above might be addressed by the PSBs themselves, e.g. by putting more emphasis on quality of the published data and metadata (CL14, CL20, CL21) and the user engagement (CL11), some of the challenges will probably require more systematic changes. Charging for data is one of such issues. In its recent notice [7] the European Commission recommends regular assessment of the potential costs and benefits of a zero-cost policy and a marginal cost policy. However according to [32] if the civil servants are responsible for the income of the relevant PSBs it might lead to maximization of the fees. In some cases PSBs even see the commercial re-users as competitors and believe in selling their data [32].

Some of the challenges presented above might not be unique to the OGD domain, e.g. insufficient data quality. However OGD utilize the web as a medium for the data provision and consumption and due to this it contributes to the data on the web phenomenon [16]. Current draft of the W3C Data on the Web Best Practices points to the fact that the openness and flexibility of the web can lead to new challenges [16]. The fact that the publishers and the users might be unknown to each other is one of them [16]. According to [24] the concept of quality is cross-disciplinary, however there is no single agreed up-on definition of quality. Data quality might be understood within the contexts of the fitness of the data for its intended use [5]. However if the OGD publishers are not aware of the potential users it might be difficult to specify the intended use of the published data which in turn might affect the assessment of the data quality. This illustrates that in case of the OGD some of the already known problems related to data management are put into the new context which might require specific solutions.

However the approach of the PSBs to OGD is not the only barrier to the OGD reuse. According to [26] there is a lack of knowledge of how the data can be utilized among the potential users and thus more success stories are required. More attention should be also paid to the OGD based business plans as a business plan is a precondition of any long-term OGD reuse [26].

3 Open Government Data publication methodologies

Challenges discussed in the previous section show that publishing data on the web for reuse is not just a matter of providing the data in machine-readable formats. There are various other problems that are not-technical in nature, like the challenges related to the licensing, user engagement or appropriate internal processes and organizational structures. In order to help the stakeholders to deal with the known challenges and problems methodologies and best practice guidelines for OGD publication have been developed or are currently under development. In this paper OGD publication methodology is defined as a set of methods, procedures or practices for publication of Open Government Data.

Existing OGD publication methodologies that are discussed in this paper are listed in table 2. For each of the methodologies its name is provided as well as its authors or publisher and the country of origin. Relevant references are also provided.

Table 2. OGD publication methodologies.

Name	Author/publisher	Country	Refs.
Best Practices for Publishing Linked Data	W3C	International	[11]
Czech Open Government Data Publication Methodology	D. Chlapek, J. Kučera, M. Nečaský	Czech Republic	[4]
Government Data Openness and Re-Use	M. Álvarez Espinar	Spain	[2]
Guide for disclosure of public data	Difi	Norway	[6]
Guidelines on Open Government Data for Citizen Engagement (2nd edition)	United Nations	International	[31]
Open Data Certificate	The Open Data Institute	International	[21]
Open Data Field Guide	Socrata	USA	[27]
Open Data Handbook	Open Knowledge Foundation	International	[23]
Open Data Handbook (Flanders)	Flemish government	Belgium/ Flanders	[9]
Open Data Institute Guides	The Open Data Institute	International	[22]

Name	Author/publisher	Country	Refs.
Open Data Ireland: Best Practice Handbook	D. Lee, R. Cyganiak, S. Decker	Ireland	[15]
Open Government Data Toolkit	World Bank	International	[35]
Methodology for publishing datasets as open data	COMSODE	International	[18]
Methodological Guidelines for Publishing Linked Data	B. Villazón-Terrazas, O. Corcho	Spain	[33]
National Guidelines for valorizing Public Sector Information	Agenzia per l'Italia Digitale	Italy	[1]
Project Open Data	The White House	USA	[34]

It is obvious that OGD publication methodologies are being developed both at the international level as well as at the national or local level. Space limitations do not allow us to discuss each of the methodologies in detail but they differ in scope, focus and structure. For example *Open Data Handbook* developed by the Open Knowledge Foundation [23] provides an introduction to the concept of Open Data and it provides basic recommendations for its publication. Compared to the *Open Data Handbook* the *Open Data Ireland: Best Practice Handbook* [15] provides more detailed recommendation and it also compares current international and Irish practices.

It is interesting that the United Nations and the World Bank, both well-known international organizations, developed their OGD methodologies. United Nations provides quite a comprehensive set of recommendations aimed at establishing and executing an OGD initiative [31]. The World Bank often refers to other methodologies or papers instead of developing its own recommendations. However it developed the *Open Data Readiness Assessment* tool which helps to assess the OGD readiness of a government [35].

In the USA the *Project Open Data* is supervised by the White House but it is open to anyone who wishes to participate (see [34]). On the other hand Open Data Field Guide [27] was developed by a private company Socrata which also provides solutions for OGD portals.

Alongside the USA there ale local/national OGD publication methodologies in the Czech Republic, Flanders (Belgium), Italy, Ireland, Norway and Spain. However it is necessary to say that the list of the methodologies in table 2 might not be comprehensive as a more detailed study aimed at the OGD initiatives across the globe would be necessary.

Some of the methodologies are aimed primarily at Linked Data or Linked Open Data (see [3]). Namely *Best Practices for Publishing Linked Data* [11] and *Methodological Guidelines for Publishing Linked Data* [33]. However Linked Open Data is mentioned or addressed by other methodologies as well, for example [9], [15], or [18].

Open Data Certificate is a tool for assessment of the quality of the open datasets [21]. There are four levels of the certificate [20]: Raw, Pilot, Standard and Expert. The certificate is awarded to a dataset according to what practices are being followed by its publisher. Because the required practices for the respective certificate levels are described, the Open Data Certificate can be considered as an OGD publication methodology. However there are currently no step-by-step guidelines to implementation of the required practices.

4 Methodology for publishing datasets as open data

4.1 Overview of the methodology

Methodology for publishing datasets as open data (COMSODE methodology) represents one of the outcomes of the project Components Supporting the Open Data Exploitation (COMSODE) [18]. It is a generic methodology that covers both technical and non-technical issues related to the publication of OGD. It is mainly aimed at PSBs that have already decided to publish some of their data as Open Data (although the question "*Why should I publish Open Data?*" is being discussed in the methodology there are no specific guidelines for gaining the top management support).

The COMSODE methodology consists of the five building blocks:

- Phases – a phase represents a stage of the Open Data publication process. Phases reflect the lifecycle of an open dataset and they are further divided into task.
- Cross-cutting activities – activities that should be performed in every phase of the open data publication process are called the cross-cutting activities. Cross-cutting activities are also divided into tasks.
- Artefacts – artefacts represent the inputs and outputs of the tasks.
- Roles – a role represents a responsibility assigned to one or more persons in an organization. In the context of the methodology roles are being assigned with responsibilities for the tasks of the phases of the cross-cutting activities.
- Practices – practices provide more detailed guidelines to execution of the tasks specified by the methodology.

The following phases of the open data publication process are proposed in the COMSODE methodology [18]:

1. (P01) Development of open data publication plan,
2. (P02) Preparation of publication,
3. (P03) Realization of publication,
4. (P04) Archiving.

Objectives of the first phase *(P01)* are to identify potential datasets for opening up, to analyze and prioritize the datasets taking into account risks, benefits and cost and to develop an open data publication plan for the selected datasets. In the second phase *(P02)* the selected datasets are prepared for publication, tasks in this phase involve for example transformation of the data into machine-readable formats, creation of metadata or selecting the appropriate license. Once the datasets are prepared for publication, tasks of the third phase *(P03)* can be executed. Maintenance of the datasets is also performed during this phase. The goal of the last phase *(P04)* is to manage end-of-life stage of the dataset lifecycle. Activities of this phase are triggered when it is no longer possible to maintain or even make available some of the previously published open datasets, e.g. due to the changes in legislation.

Four cross-cutting activities were identified that should be performed throughout the whole publication process [18]:

1. (CA01) Data quality management,
2. (CA02) Communication management,
3. (CA03) Risk management,
4. (CA04) Benefits management.

The cross-cutting activities are aimed at management of the data quality *(CA01)*, the communication and collaboration between the publisher and the (potential) users of its data *(CA02)* and at management of the benefits *(CA03)* and potential risks *(CA04)* related to publication of OGD.

Feedback from the re-users is an important part of the OGD ecosystem [13], [26]. Therefore, it is not viewed just as a single step in the publication process but rather as a cross-cutting activity that should be performer throughout the whole publication process. This approach is depicted in figure 1. More details about how the feedback should be processed and how the users should be engaged is described in [18].

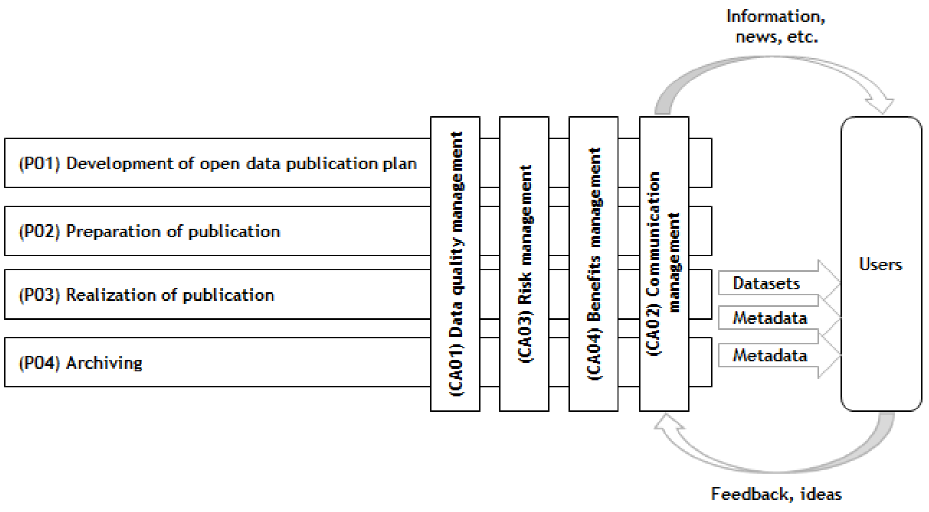

Information, news, etc.
(P01) Development of open data publication plan
(P02) Preparation of publication
(P03) Realization of publication
(P04) Archiving
(CA01) Data quality management
(CA03) Risk management
(CA04) Benefits management
(CA02) Communication management
Datasets
Metadata
Metadata
Users
Feedback, ideas

Figure 1. Feedback loop in the COMSODE methodology, source: [18]

4.2 Challenges addressed by the methodology

COMSODE methodology addresses some of the challenges of the OGD publication discussed in the section no. 2:

- Identification of datasets for opening up – the methodology provides recommendation for identification of datasets for opening up and it also promotes publication of datasets that are perceived as "*high-value datasets*" (addresses the challenge *CL4*).
- It discusses the potential OGD benefits including the benefits for the whole economy which might help to avoid situations when PSBs too much focus on their internal benefits *(CL5)*. However, benefits for the PSBs are discussed as well.
- The methodology provides recommendations for the effort estimation which might help to manage costs of the OGD initiative *(CL8)*. There is a separate cross-cutting activity aimed at the management of benefits *(CL9)*.
- The methodology proposes an OGD publication process and it sets responsibilities for the proposed tasks. This might help to establish standard process and organization structure supporting the OGD publication *(CL10, CL13)*.
- In order to prevent the lack of interaction between OGD users and publishers *(CL11)* the cross-cutting activity *(CA03) Communication management* should be performed.
- The methodology is independent on the availability of the central data portal *(CL12)*. However if the central data portal is available, PSBs are free to utilize it.
- A separate cross-cutting activity is introduced in order to ensure that the published data has the desired level of quality *(CL20)*. Maintenance of the data and metadata is addressed by specific tasks and the related practices *(CL14)*.
- A separate cross-cutting activity is introduced in order to manage the OGD related risks including the risk of the personal data protection violation *(CL15)*.
- The methodology proposes recommended practices for dataset licensing *(CL16)*.
- There are several practices in the methodology aimed at ensuring high level of technical openness of the datasets. These practices include but are not limited to the reuse of the existing schemas and ontologies *(CL17)* and documenting the schemas in a machine-readable way *(CL22)*.
- The methodology promotes data cataloging which should help to improve discoverability of the datasets *(CL21)*.

However, there are some remaining issues that are not addressed by the current version of the COMSODE methodology, namely:

- Social issues – the methodology is aimed mainly at the individual public sector bodies. Solving the social issues would probably require actions on the government level (consolidation/coordination of the OGD initiatives, *CL1*) or actions aimed at the re-users (building knowledge and skills how the data can be use and how to provide feedback that can be effectively used for improvements, *CL2-3*).
- Fees (*CL6*) – the methodology provides no guidelines reading fees.
- Limited resources, especially in case of the small PSBs *(CL7)* – the methodology does not provide any recommendations specifically tailored for particular types of PSBs. This challenge is therefore not addressed.
- The actual design of the data portals is outside the scope of the methodology and thus it does not provide any recommendation regarding registration of the users on

the portals *(CL18)*. Dealing with the legacy applications is also beyond the scope of the methodology *(CL23)*.

The COMSODE methodology promotes a risk based approach to the OGD publication. This means that if some data cannot be published in its primary form due the possible breach of the personal data protection, it proposes to anonymize the data. It does not prevent publication of the primary data *(CL19)*, but it respects that the publication of some primary data is not always possible.

The COMOSDE methodology is software tool independent. It only proposes a Reference architecture of software tools for open data publication [19]. However, a platform called the Open Data Node is developed in the COMSODE project [10].

4.3 Methodology in use

The COMSODE methodology has been utilized in a project aimed at opening up data of the Supreme Audit Office of the Czech Republic. As a first step a project plan following the phases P01-03 of the methodology was prepared. Identification of suitable datasets for opening up and development of the open data publication plan was performed in January and February 2015. The publication of the selected datasets is expected to a happen in June 2015. A selected subset of the datasets will also be published as Linked Open Data.

In January 2015 the Ministry of Interior of the Czech Republic launched a project aimed at supporting the Czech PSBs in their OGD initiatives. The COMSODE methodology serves as one of the most significant inputs upon which the Czech Open Government Data standards will be developed.

5 Conclusions

The Open Government Data promise significant benefits to citizens, business as well as to the public sector. However, PSBs often face challenges when publishing OGD. Based on the literature review, 24 challenges related to the OGD publication and reuse were identified. These challenges include the political and social challenges, economic, legal and technical challenges as well as the organizational challenges and challenges related to the internal processes. Even though some of the identified challenges might not be completely unique to the OGD domain, e.g. the insufficient data quality, OGD might represent a unique context for these challenges which might require specific solutions.

In order to help the stakeholders to deal with the known challenges and problems, methodologies and best practice guidelines for OGD publication have been developed or they are currently under development. We were able to identify 16 OGD publication methodologies at both international and national or local level. Further analysis of these methodologies might provide a better understanding of the current best practices for publication and reuse of the Open Government Data.

Methodology for publishing datasets as open data is one of the existing OGD publication methodologies. This methodology is one of the generic OGD publication methodologies. It proposes an OGD publication process organized into four phases: (P01) Development of open data publication plan, (P02) Preparation of publication, (P03) Realization of publication, (P04) Archiving. The phases are accompanied by four cross-cutting activities: (CA01) Data quality management, (CA02) Communication management, (CA03) Risk management, (CA04) Benefits management. This methodology is currently used to in a project aimed at opening up data of the Supreme Audit Office of the Czech Republic and is expected to be used as one of the key resources upon which the Czech Open Government Data standards will be developed by the Ministry of Interior of the Czech Republic.

Acknowledgements. The research is supported by the EU ICT FP7 project COMSODE under GA No. 611358.

6 References

1. Agenzia per l'Italia Digitale. Linee guida nazionali per la valorizzazione del patrimonio informativo pubblico [National Guidelines for valorizing Public Sector Information] (2014), `http://www.agid.gov.it/sites/default/files/linee_guida/patrimoniopubblicolg2014_v0.7finale.pdf`
2. Álvarez Espinar, M.: Government Data Openness and Re-Use (2014), `http://transparencia.gencat.cat/web/sites/transparencia/.content/pdfs/governobert/governobert_2_en.pdf`
3. Berners-Lee, T.: Linked Data - Design Issues (2006), `http://www.w3.org/DesignIssues/LinkedData.html`
4. Chlapek, D., Kučera, J., Nečaský, M.: Metodika publikace otevřených dat veřejné správy ČR [Czech Open Government Data Publication Methodology] (2012), `http://www.mvcr.cz/soubor/metodika-publ-opendata-verze-1-0-pdf.aspx`
5. Data Management Association, The: Guide to the Data Management Body of Knowledge. Technical Publications (2010)
6. Difi: Del og skap verdier - Veileder i tilgjengeliggjøring av offentlige data [Difi Guide for disclosure of public data] (2013), `http://data.norge.no/document/del-og-skap-verdier-veileder-i-tilgjengeliggj%C3%B8ring-av-offentlige-data`
7. European Union: Comission Notice. Guidelines on recommended standard licences, datasets and charging for the reuse of documents (2014/C 240/01) (2014), `http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52014XC0724%2801%29`
8. European Union: Directive 2013/37/EU of the European parliament and the Council of 26 June 2013 amending Directive 2003/98/EC on the re-use of public sector information (2013), `http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32013L0037`
9. Flemish government: Open Data Handleiding [Open Data Handbook] (2014), `http://www.opendataforum.info/images/Open_Data_Handboek_20141119.pdf`

10. Hanečák, P.: Open Data Node – what it is, what it does, what is next (2014), http://www.comsode.eu/index.php/2014/06/open-data-node-what-it-is-what-it-does-what-is-next/
11. Hyland, B., Atemezing, G., Villazón-Terrazas, B.: Best Practices for Publishing Linked Data (2014), http://www.w3.org/TR/ld-bp/
12. Janssen, M., Charalabidis, Y. and Zuiderwijk, A.: Benefits, Adoption Barriers and Myths of Open Data and Open Government. Information Systems Management, vol. 29, no. 4, pp. 258-268 (2012)
13. Janssen, M. and Zuiderwijk, A.: Open data and transformational government. In: Proceedings of the Transforming Government Workshop 2012 (tGov2012), pp. 1-7. Brunel University, London, United Kingdom (2012)
14. Kučera, J. and Chlapek, D.: Benefits and Risks of Open Government Data. Journal of Systems Integration, vol. 5, no. 1, pp. 30-41 (2014)
15. Lee, D., Cyganiak, R., Decker, S.: Open Data Ireland: Best Practice Handbook (2014), http://www.per.gov.ie/open-data/
16. Lóscio, B.F., Burle, C., Calegari, N.: Data on the Web Best Practices. W3C First Public Working Draft (2015), http://www.w3.org/TR/2015/WD-dwbp-20150224/
17. National Audit Office: Implementing transparency (2012), http://www.nao.org.uk/wp-content/uploads/2012/04/10121833.pdf
18. Nečaský, M. Chlapek, D., Klímek, J., Kučera, J., Maurino, A., Rula, A., Konecny, M., Vanova, L.: Deliverable D5.1: Methodology for publishing datasets as open data (2014), http://www.comsode.eu/wp-content/uploads/D5.1-Methodology_for_publishing_datasets_as_open_data.pdf
19. Nečaský, M. Chlapek, D., Klímek, J., Kučera, J., Maurino, A., Rula, A.: Deliverable D5.1: Methodology for publishing datasets as open data. Documentation of practices (2014), http://www.comsode.eu/wp-content/uploads/Annex1_D5.1-Documentation_of_practices.pdf
20. Open Data Institute, The: About the Open Data Certificate, https://certificates.theodi.org/about
21. Open Data Institute, The: Open Data Certificate, https://certificates.theodi.org/
22. Open Data Institute, The: Guides, http://theodi.org/guides
23. Open Knowledge Foundation: The Open Data Handbook (2012), http://opendatahandbook.org/
24. Russell, G.R., Miles, M.P.: The definition and perception of quality in ISO-9000 firms. In: Review of Business, vol. 9, no.3, pp. 13-16 (1998)
25. Share-PSI: Uses of Open Data Within Government for Innovation and Efficiency: Report (2014), https://www.w3.org/2013/share-psi/workshop/samos/report
26. Share-PSI: Encouraging open data usage by commercial developers: Report (2015), https://www.w3.org/2013/share-psi/workshop/lisbon/report
27. Socrata: Open Data Field Guide (2014), http://www.socrata.com/open-data-field-guide/
28. Sunlight Foundation: Ten Principles for opening up government information (2010), http://sunlightfoundation.com/policy/documents/ten-open-data-principles/
29. Tinhold, D.: The Open Data Economy. Unlocking Economic Value by Opening Government and Public Data (2013), https://www.capgemini-consulting.com/ebook/The-Open-Data-Economy/files/assets/downloads/publication.pdf

30. Ubaldi, B.: Open Government Data: Towards Empirical Analysis of Open Government Data Initiatives. OECD Working Papers on Public Governance, vol. 22. OECD Publishing (2013)
31. United Nations: Guidelines on Open Government Data for Citizen Engagement (2013), http://workspace.unpan.org/sites/Internet/Documents/Guidenlines%20on%20OGDCE%20May17%202013.pdf
32. Van Herreweghe, N.: Open Data Dag In Vlaanderen, Conclusions (2014), http://www.w3.org/2013/share-psi/workshop/lisbon/oddv
33. Villazón-Terrazas, B. and Corcho, O.: Methodological Guidelines for Publishing Linked Data (2011), http://delicias.dia.fi.upm.es/wiki/images/7/7a/07_MGLD.pdf
34. White House: Project Open Data, https://project-open-data.cio.gov/
35. World Bank: Open Government Data Toolkit (2014), http://data.worldbank.org/open-government-data-toolkit
36. World Wide Web Foundation, The: Open Data Barometer – Second Edition (2015), http://www.opendatabarometer.org/assets/downloads/Open%20Data%20Barometer%20-%20Global%20Report%20-%202nd%20Edition%20-%20PRINT.pdf

Introduction to Optical Music Recognition: Overview and Practical Challenges

Jiří Novotný and Jaroslav Pokorný

Department of Software Engineering, Faculty of Mathematics and Physics
Charles University, Malostranské nám. 25, Prague, Czech Republic
{novotny, pokorny}@ksi.mff.cuni.cz

Abstract. Music has been always an integral part of human culture. In our computer age, it is not surprising that there is a growing interest to store music in a digitized form. Optical music recognition (OMR) refers to a discipline that investigates music score recognition systems. This is similar to well-known optical character recognition systems, except OMR systems try to automatically transform scanned sheet music into a computer-readable format. In such a digital format, semantic information is also stored (instrumentation, notes, pitches and duration, contextual information, etc.). This article introduces the OMR field and presents an overview of the relevant literature and basic techniques. Practical challenges and questions arising from the automatic recognition of music notation and its semantic interpretation are discussed as well as the most important open issues.

Key words: optical music recognition, document image analysis, machine learning

1 Introduction

Computer perception of music notation forms a constantly growing research field called *optical music recognition* (OMR). The main goal of all OMR systems is to automatically decode and interpret the symbols of music notation from scanned images. Results of the recognition are represented in a digital format suitable to store the semantic information (notes, pitches, dynamics and so on). The main advantage of such representation of music scores is the possibility of different applications such as: audio playback, reediting, musicological analyses, conversions to different formats (e.g. Braille music notation) and the preservation of cultural heritage [23]. More recent applications are for example: concert-planning systems sensitive to the emotional content of music [7] or automatic mapping of scanned sheet music to audio recordings [18].

Over the years, music had been traditionally written down with ink and paper. During the 1980s, early computer music typesetting programs were developed, which revolutionized the way how music can be recorded. Nowadays, the most common approach to transform music data into a computer-readable format (used by professional musicians) combines musical keyboard input (e.g.

MIDI piano) with computer keyboard and mouse. It is a time-consuming procedure, which requires advanced keyboard-playing skills. The musical keyboard is utilized to enter the notes playing voice by voice and then the computer keyboard and mouse is used to correct mistakes and to add another information such as articulation marks, slurs and dynamics.

The majority of music scores exist only in the paper-based form and many contemporary composers and musicians still prefer to use pen and paper as the most efficient way to record their ideas. OMR systems can thus greatly simplify the music data acquisition and save a lot of human time.

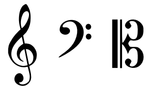
In this article, we survey the area of OMR, its fundamental approaches and problems. Section 1.1 introduces a few aspects of music notation, and Section 1.2 reviews the historical context of OMR. A general framework for the music recognition is presented in Section 2. Challenges making the OMR difficult in practice are discussed in Section 3. Section 4 debates several opened questions of the OMR research and finally, we conclude this paper in Section 5.

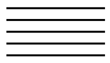
1.1 Music Notation

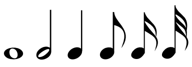
Music notation has evolved over the period of centuries as the composers and musicians tried to express their musical ideas by written symbols [33]. In this article, we focus exclusively on the Western music notation (also known as common music notation — CMN) [10,39], although certain OMR systems are developed to recognize other types of notation (e.g. medieval music notation [13,42]).

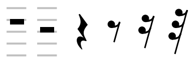
Understanding of any music notation requires knowledge of the information the notation attempts to capture. In the case of CMN, there are four types of information involved [10]: a *pitch*, *time*, *loudness* (also *dynamics*) and *timbre* (tone quality). Figure 1 shows selected music notation marks. Clefs (Fig. 1a) determine the pitches for each line and space of the staff (Fig. 1b), accidentals (Fig. 1e) temporarily modify the pitch of following notes. The pitch of notes itself (Fig. 1c) is indicated by their vertical placement on the staff, and their appearance affects the relative duration. Ornaments (Fig. 1f) change the pitch pattern of individual notes. Rests (Fig. 1d) indicate a relative duration of silence. Dynamics (Fig. 1g) signify the varying loudness. Articulations (Fig. 1h) change the timbre or duration of a note. In practice, certain symbols have almost unlimited variations in representation (e.g. beams connecting notes into note groups or slurs indicating phrasing). The most used CMN symbols and their graphical aspects are listed e.g. in the Essential Dictionary of Music Notation [21].

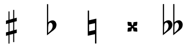
Music Scores. For the music recognition purposes, it is useful to realize, that music scores can be divided into three categories: entirely printed music scores (Figure 2a), scores written by hand over the preprinted staff lines (Figure 2b) and entirely handwritten music scores (Figure 2c). Although the majority of OMR systems operates with printed scores only [35], OMR systems for handwritten music have been researched as well (e.g. [1,17,27,36,37,45]). Also it should be noted, that scores of different visual qualities exist — from the clear music sheets to the degraded ones (mentioned e.g. in [9,11]).

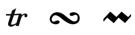

(a) Clefs.


(b) Staff.

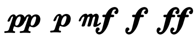

(c) Notes.


(d) Rests.


(e) Accidentals.



(f) Ornaments.



(g) Dynamics.

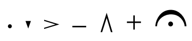
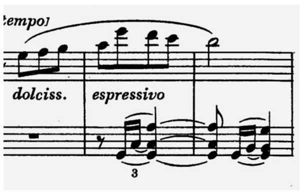
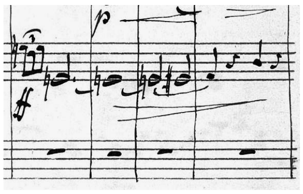

(h) Articulations.

Fig. 1: Selected common music notation symbols.



(a) Entirely printed.


(b) Preprinted staff lines.

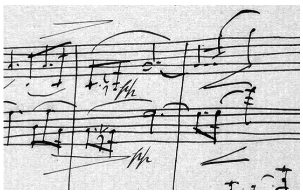

(c) Entirely handwritten.

Fig. 2: Examples of different sheet music categories.

1.2 Historical Background

The OMR research began in 1966, when Pruslin [32] first attempted the automatic recognition of sheet music. His system was able to recognize note heads and chords. In 1970, Prerau [31] introduced the concept of image segmentation to detect primitive elements of music notation. These two OMR founding works were later reviewed by Kassler [24].

With the availability of inexpensive optical scanners, the OMR research expanded in the late 1980s. An interesting contribution was a Japanese keyboard-playing robot WABOT-2 [25], developed in 1984. It was the first robot able to recognize simple music scores and play them on the organ. A critical survey of the OMR systems developed between 1966 and 1990 can be found in [8].

The first commercial OMR products appeared in the early 1990s [23,35]. Also, the first attempts to handle handwritten scores were made (e.g. [37,45]). In 1997, Bainbridge summarized the existing techniques and proposed an extensible music recognition system [2] not restricted to particular primitive shapes and semantic features. Together with Bell [3], they formulated a general framework for OMR systems, which has been adopted by many researchers since then [35].

During the last years, several important studies have been performed: Jones et al. [23] presented a study about music sheet digitization, recognition and restoration. Moreover, they listed available OMR software and provided an evaluation of three OMR systems. Noteworthy contributions to the OMR have been made by Rebelo et al. [30, 34, 36]. In 2012, they published probably the most recent review of the OMR field [35], including an overview of the state-of-the-art techniques and a discussion about the open issues.

2 General Framework

Automatic recognition of music scores is a complex task affecting many areas of computer science. Different OMR systems use various strategies, but the most common algorithms decompose the problem into four smaller tasks [35]:

1. Image Preprocessing
2. Segmentation
3. Object Recognition
4. Semantic Reconstruction

Terminology is not always the same: the segmentation is also called *primitive detection* or *musical object location*, and the recognition phase is sometimes called *musical feature classification* [2, 3].

2.1 Image Preprocessing

The main goal of the preprocessing phase is to adjust the scanned image to make the recognition process more robust and efficient. Different methods are typically used: *enhancement*, *blurring* and *morphological operations* [22] and *noise removal* (e.g. [20, 22, 40, 42]), *deskewing* [17, 20, 22, 27, 42] and *binarization* (e.g. [9, 17, 20, 22, 27, 30, 42]). In the following text, only the binarization is introduced as it is the most crucial step for the vast majority of OMR systems.

Binarization. Binarization algorithms convert the input image into a binary one, where objects of interest (music symbols, staves, etc.) are separated from the background. This is motivated by the fact, that music scores have inherently binary nature (colors are not used to store music information in CMN).

Binarization is usually an automated process driven without special knowledge of the image content. It facilitates the subsequent tasks by reducing the volume of information that is needed to be processed. For example, it is much easier to design an algorithm for staff detection, primitive segmentation and recognition in binary images than in grayscale or color ones.

In general, there are two types of binarization approaches. The first are *global thresholding methods*, which apply one particular threshold to the entire image. The Otsu's method [29] is often assessed to be the best and fastest [38,44]. Global thresholding works well when extracting objects from uniform backgrounds, but usually fails on non-uniform images. Nevertheless, it is used in several OMR research articles (e.g. [22, 34, 42]) because of its simplicity and time efficiency. The second category is represented by *adaptive binarization techniques*, which select a threshold individually to each pixel using information from the local neighborhood. These methods can eliminate non-uniform backgrounds at the expense of longer processing time. One of the most popular adaptive thresholding method is the Niblack's [28] that computes a local threshold from the mean and standard deviation in pixel's surroundings. Adaptive thresholding is also used in some OMR systems (e.g. [40]). Overview of binarization techniques used in OMR can be found in [9,38].

2.2 Segmentation

The segmentation stage parses music scores into the elementary primitives. It is usually initiated by establishing the size of the music notation being processed. This is an important step before any shape recognition. Staff lines are a reliable feature of music notation used to estimate two important reference values: *staff line thickness* and *staff space height*, which are further used to deduce the size of other music symbols. The most common way of their approximation is based on the run-length encoding (RLE), which is a simple form of data compression. For instance, lets assume the binary sequence $\{1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\}$. It can be represented in RLE as $\{3\ 2\ 4\ 4\}$ (supposing 1 starts the original sequence, otherwise the first number in the encoded sequence would be 0). Binarized music scores can be encoded column by column with RLE, then the relative lengths can be easily estimated: the most common black run approximates the staff line thickness and the most common white run estimates the staff space height. However, more robust approximations exist [12].

Staff Lines. Staff line detection is fundamental in OMR, because the staff creates a two dimensional coordinate system essential to understand the CMN. Unfortunately, staff lines are not guaranteed to be perfectly horizontal, straight or of uniform thickness in scanned images (even in printed music scores). Precise staff detection is a tricky problem that still represents a challenge.

The simplest algorithms use horizontal projections [19,20]. A horizontal projection maps a binary image into a histogram by accumulating the number of black pixels in each row. If the lines are straight and horizontal, staff can be detected as five consequent distinct peaks (local maxima) in the histogram. Figure 3 shows an excerpt of music and its horizontal projection. In practice, several horizontal projections on images with slightly different rotation angles are computed to deal with not completely horizontal staff lines. The projection with the highest local maxima is then chosen.

Another strategies use vertical scan lines [13] or Hough Transform or grouping of vertical columns [35]. Although there are many staff line detection techniques, they all have certain limitations. Dalitz [15] surveyed the existing methods and proposed a method based on skeletonization. Handwritten staff lines are usually detected using different kinds of techniques (e.g. [1,43]).

Symbol Segmentation. Once the staff lines have been detected, the music primitives must be located and isolated. This can be performed in two manners:

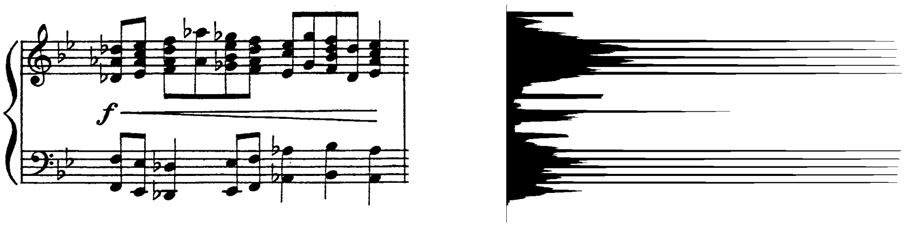


Fig. 3: The horizontal projection of a music score excerpt.

either remove the staff lines or ignore them. Although the majority of researchers remove the staff lines in order to isolate the musical symbols as connected components, there are some authors who suggest the opposite (e.g. [4,22]). The most simple line removal algorithm removes the line piecewise — following it along and replacing the black line pixels with white pixels unless there is evidence of an object on either side of the line [3]. The staff line removal procedure must be careful not to broke any object. Despite that, the algorithms often cause fragmentation, especially to objects that touch the staff lines tangentially.

The score is then divided into regions of interest to localize and isolate the musical primitives. The best approach is hierarchical decomposition [35]. At first, a music score is analyzed and split by staves. Then, the primitive symbols (note heads, stems, flags, rests, etc.) are extracted [22, 27, 34]. Particular procedures vary system to system. For example, some approaches consider note heads, stems and flags to be separate objects, whereas other concepts consider these primitives as a whole object representing a single note. More details can be found in [34].

2.3 Object Recognition

Segmented symbols are further processed and given to the classifier that tries to recognize them (assign them a label from predefined groups). Unfortunately, music shapes are inherently complex — they are often formed by several touching and overlapping graphical components. In addition, the staff line removal can break some objects (they are sometimes already fragmented because of the music score quality itself). Hence, the object recognition phase is very delicate and it is usually combined with the segmentation step [35].

Objects are classified according to their distinctive features. Some authors suggest classification using projection profiles [19], others apply template matching to recognize symbols [22] or propose a recognition process entirely driven by grammars formalizing the music knowledge [14]. Statistical classification methods using support vector machines (SVMs), neural networks (NN), k-nearest neighbours (kNN) and hidden Markov models (HMM) classifiers were investigated by Rebelo et al. [34]. Handwritten music symbols are sometimes segmented and recognized using the mathematical morphology, applying a skeletonization technique and an edge detection algorithm [26]. Despite the number of recognition techniques available, research on symbol segmentation and recognition is still important and necessary, because all OMR systems depend on it [35].

2.4 Semantic Reconstruction

The inevitable task of all OMR systems is to reconstruct the musical semantics from previously recognized graphical primitives and store the information in a suitable data structure. This necessarily requires an interpretation of spacial relationships between objects found in the image. Relations in CMN are essentially two dimensional and the positional information is very critical. For example, a dot can change note's duration if it is placed on the right of a note head, or it can alter the articulation if it is placed above the note.

These musically syntactic rules can be formalized using the grammars [2, 14, 19, 31]. Grammar rules specify semantically valid music notation events and a way, how the graphical primitives should be segmented. Alternative techniques build the semantic reconstruction on a set of rules and heuristics (e.g. [16, 26]).

The last and fundamental aspect of OMR systems is the transformation of semantically recognized scores in a coding format that is able to model and store music information. Many computer formats are available, but none of them has been accepted as a standard. The best known formats are: MIDI (Musical Instrument Digital Interface), NIFF (Notation Information File Format), SMDL (Standard Music Description Language) and MusicXML[1]. MIDI is mainly used as an interchange format between digital instruments and computers. Although its capability of modeling music scores is very limited (e.g. the relationships among symbols cannot be represented), most of the music editors can operate with MIDI files. NIFF was developed in 1994 to exchange data between different music notation software. NIFF is able to describe visual and logical aspects of music, however nowadays it is considered to be obsolete. SMDL strictly separates visual and logical sites and it is rather a standardized formal scheme than a practical file format. MusicXML is designed especially for sharing and archiving of music sheets. It covers the logical structure and also graphical aspects of music scores. It is becoming more and more popular and it targets to be the standard open format for exchanging digital sheet music. A more detailed review and comparison of music notation file formats can be found in [6].

3 Practical Challenges

Despite the fact that OMR systems have been researched thoroughly over the last few decades and even several commercial tools exist, the practical results are still far from ideal [35]. Proposed techniques are typically tailored to different properties of music scores, which makes them difficult to combine in one general OMR system robust enough to overcome all the practical issues. In this section, we focus on reasons that makes the OMR systems challenging in practice and we also discuss some open problems of the research area.

[1] http://www.musicxml.com/

3.1 Preprocessing

Preprocessing is the initial step of all OMR systems, which obviously affects the subsequent stages. However, no goal-directed studies investigating the impact of this phase on the recognition have been carried out [35]. Binarization often produces artifacts and its advantages in the complete OMR process are not clear. There are few attempts to use prior knowledge when performing a binarization process [30]. Such algorithm extracts content-related information from a grayscale image and uses it to guide the binarization. Cardoso et al. [12] encourage the idea of using grayscale images rather then the binary ones. A special care must be also given to highly degraded music scores [9].

In our opinion, it is also worth considering the possibility to analyze the color information when processing handwritten scores with preprinted staff lines, because the color of the composer's ink may slightly vary from the color of the staff lines. It could possibly result in a more efficient staff removal algorithm. This and similar image analysis topics are in our research interest.

3.2 Music Notation Inherent Problems

Music notation itself implies many difficult-to-process variants and possibilities of music representation typically responsible for serious recognition errors. Two different practical troubles are shown in Figure 4. The long curves connecting notes of distinct pitches (*slurs*) can have an arbitrary shape, thus they represent a great challenge for OMR systems. The last bar of the examples presents another difficulty: the notes pass to another staff, while their beams are crossing (moreover, they superimpose the crescendo sign and the slur).

There are plenty of similar problematic properties in CMN, for example: a smaller staff placed above the main staff indicating how a part of music can be alternatively played (*ossia*), simplifications for a better human readability that can be interpreted ambiguously (e.g. omitting the number 3 in triplets or alternating the left and right hands across the staves in the piano literature) or ornamental note groups that do not fit the prescribed meter. It should be also noted, that not all notation formats are able to represent such features. Nevertheless, these kinds of difficulties are nothing exceptional in real music sheets and hence cannot be omitted in a practical OMR system.

3.3 Handwritten Scores

Handwritten music sheets produce specific kinds of problems. Although they are also mentioned in the literature [1, 17, 27, 36, 37, 45], the results are still not usable for practical applications. In general, the major problem of OMR systems are fragmented and connected (touching or overlapping) music symbols. Handwritten music scores contain even more broken and merged symbols — it could be a part of composer's written style or just a consequence of quickly-made strokes. Figure 5 shows a huge variability of written styles of four different composers. These facts make the recognition of handwritten scores complicated.

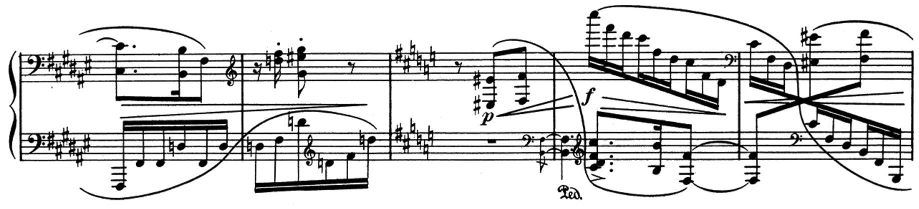


Fig. 4: Example of variations in notation (from Maurice Ravel's Scarbo).

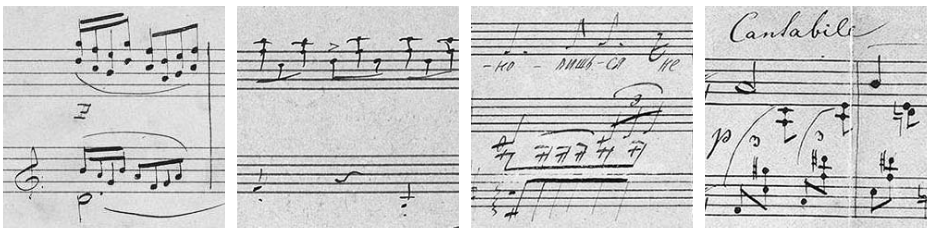


Fig. 5: An example of composer variability in handwritten scores.

4 Open Problems

One of the most important open issues of the OMR research is the lack of an available ground-truth database that could serve as a benchmark. Such a data set would contain a large amount of music scores of different types and qualities (clean scans, photocopies, degraded manuscripts, etc.) together with their ground-truth representation in a uniform notation format. Compilation of such corpus is extremely time-consuming, because the music sheets have to be processed by hand. Available data sets (e.g. [30]) are typically very limited or designed only for specific tasks [35]. Maybe a solution would be to design an automatic method able to procedurally simulate different types of writing styles and paper degradation levels from a given notation file. Available electronic scores then would be easily transformed to images of different qualities.

Another significant problem is the absence of common methodologies and metrics that would compare the results of OMR systems. This is a more complicated issue than it might seem at first glance, because OMR systems can target different goals (audio playback, score archiving, ...) and the outputs can be stored in very unlike formats. However, performance evaluation and related topics have been also studied in the literature. For example, Szwoch [41] proposed a method able to compare and evaluate the results of recognition systems stored in MusicXML format. More on this topic can be found in [5, 11].

In addition, we think, that music knowledge should be incorporated more to support the recognition and reconstruction processes. For example, considering the advanced analysis of music harmony or building a composer-adaptive system (adaptive to the composer's writing style as well as to the music style). To the best knowledge of the authors, no studies concerning these or similar topics exist. Together with the image analysis issues, this is one of the subjects on which we would like to focus our research.

5 Conclusion

During the last decades, OMR has been actively studied and a lot of achievements have been done. Even so, the problem is not solved and represents a great challenge in many ways. Possible OMR applications are still relevant today, which makes the research area constantly growing.

In this article, we have introduced the OMR field, its main goals and practical applications. We have also presented an overview of the most common methodologies including the idea of a generalized framework. The most delicate and challenging problems that all OMR systems have to face have been discussed as well. We hope, that our contribution helps to motivate the researchers, because there are many demanding problems waiting to be solved.

References

[1] Alirezazadeh, F., Ahmadzadeh, M.R.: Effective staff line detection, restoration and removal approach for different quality of scanned handwritten music sheets. Journal of Advanced Computer Science and Technology 3(2), 136–142 (2014)

[2] Bainbridge, D.: Extensible Optical Music Recognition. Ph.D. Thesis, Department of Computer Science, University of Canterbury, Christchurch, NZ (1997)

[3] Bainbridge, D., Bell, T.: The Challenge of Optical Music Recognition. Computers and the Humanities 35(2), 95 – 121 (2001)

[4] Bellini, P., Bruno, I., Nesi, P.: Optical music sheet segmentation. In: Web Delivering of Music, 2001. Proceedings. First International Conference on. pp. 183–190 (Nov 2001)

[5] Bellini, P., Bruno, I., Nesi, P.: Assessing Optical Music Recognition Tools. Comput. Music J. 31(1), 68–93 (Mar 2007)

[6] Bellini, P., Nesi, P.: Modeling Music Notation in the Internet Multimedia Age. In: George, S.E. (ed.) Visual Perception of Music Notation: On-Line and Off Line Recognition, pp. 272–303. IGI Global (2004)

[7] Billinge, D., Addis, T.: Towards Constructing Emotional Landscapes with Music. In: George, S.E. (ed.) Visual Perception of Music Notation: On-Line and Off Line Recognition, pp. 227–271. IGI Global (2004)

[8] Blostein, D., Baird, H.S.: A Critical Survey of Music Image Analysis. In: Baird, H.S., Bunke, H., Yamamoto, K. (eds.) Structured Document Image Analysis, pp. 405–434. Springer Berlin Heidelberg (1992)

[9] Burgoyne, J.A., Pugin, L., Eustace, G., Fujinaga, I.: A Comparative Survey of Image Binarisation Algorithms for Optical Recognition on Degraded Musical Sources. pp. 509–512. Austrian Computer Society (2007)

[10] Byrd, D.: Music Notation by Computer. Ph.D. thesis, Indiana University, Computer Science Department (1984)

[11] Byrd, D., Simonsen, J.G.: Towards a Standard Testbed for Optical Music Recognition: Definitions, Metrics, and Page Images. University of Copenhagen, Copenhagen (2013)
[12] Cardoso, J., Rebelo, A.: Robust Staffline Thickness and Distance Estimation in Binary and Gray-Level Music Scores. In: Pattern Recognition (ICPR), 2010 20th International Conference on. pp. 1856–1859 (Aug 2010)
[13] Carter, N.: Segmentation and Preliminary Recognition of Madrigals Notated in White Mensural Notation. Machine Vision and Applications 5(3), 223–229 (1992)
[14] Coüasnon, B., Camillerapp, J.: Using grammars to segment and recognize music scores. International Association for Pattern Recognition Workshop on Document Analysis Systems pp. 15–27 (1994)
[15] Dalitz, C., Droettboom, M., Pranzas, B., Fujinaga, I.: A Comparative Study of Staff Removal Algorithms. Pattern Analysis and Machine Intelligence, IEEE Transactions on 30(5), 753–766 (May 2008)
[16] Droettboom, M., Fujinaga, I., MacMillan, K.: Optical Music Interpretation. In: Caelli, T., Amin, A., Duin, R., de Ridder, D., Kamel, M. (eds.) Structural, Syntactic, and Statistical Pattern Recognition, Lecture Notes in Computer Science, vol. 2396, pp. 378–387. Springer Berlin Heidelberg (2002)
[17] Fornés, A., Lladós, J., Sánchez, G.: Primitive Segmentation in Old Handwritten Music Scores. In: Proceedings of the 6th International Conference on Graphics Recognition: Ten Years Review and Future Perspectives. pp. 279–290. Springer-Verlag, Berlin, Heidelberg (2006)
[18] Fremerey, C., Müller, M., Kurth, F., Clausen, M.: Automatic Mapping of Scanned Sheet Music to Audio Recordings. In: Bello, J.P., Chew, E., Turnbull, D. (eds.) ISMIR 2008, 9th International Conference on Music Information Retrieval, Drexel University, Philadelphia, PA, USA, September 14-18, 2008. pp. 413–418 (2008)
[19] Fujinaga, I.: Optical music recognition using projections. M.A p. Thesis (1988)
[20] Fujinaga, I.: Staff Detection and Removal. In: George, S.E. (ed.) Visual Perception of Music Notation: On-Line and Off Line Recognition, pp. 1–39. IGI Global (2004)
[21] Gerou, T., Lusk, L.: Essential Dictionary of Music Notation. Alfred Music Publishing (1996)
[22] Göcke, R.: Building a System for Writer Identification on Handwritten Music Scores (2003)
[23] Jones, G., Ong, B., Bruno, I., NG, K.: Optical Music Imaging: Music Document Digitisation, Recognition, Evaluation, and Restoration. Interactive Multimedia Music Technologies pp. 50–79 (2008)
[24] Kassler, M.: Optical Character Recognition of Printed Music: A Review of Two Dissertations. Perspectives of New Music 11 (1972)
[25] Matsushima, T.: Automated recognition system for musical score: The vision system of WABOT-2. Bulletin of Science and Engineering Research Laboratory (1985)
[26] Ng, K.C., Cooper, D., Stefani, E., Boyle, R.D., Bailey, N.: Embracing the Composer: Optical Recognition of Handwritten Manuscripts. In: Proceedings of the International Computer Music Conference. pp. 500–503 (1999)
[27] Ng, K.: Optical Music Analysis for Printed Music Score and Handwritten Music Manuscript. In: George, S.E. (ed.) Visual Perception of Music Notation: On-Line and Off Line Recognition, pp. 108–127. IGI Global (2004)
[28] Niblack, W.: An Introduction to Digital Image Processing. Strandberg Publishing Company, Birkeroed, Denmark, Denmark (1985)
[29] Otsu, N.: A Threshold Selection Method from Gray-Level Histograms. Systems, Man and Cybernetics, IEEE Transactions on 9(1), 62–66 (Jan 1979)

[30] Pinto, T., Rebelo, A., Giraldi, G., Cardoso, J.: Music Score Binarization Based on Domain Knowledge. In: Vitri, J., Sanches, J., Hernndez, M. (eds.) Pattern Recognition and Image Analysis, Lecture Notes in Computer Science, vol. 6669, pp. 700–708. Springer Berlin Heidelberg (2011)
[31] Prerau, D.: Computer pattern recognition of standard engraved music notation. Ph.D. Dissertation, Massachusetts Institute of Technology (1970)
[32] Pruslin, D.: Automatic recognition of sheet music. Sc.D. Dissertation, Massachusetts Institute of Technology (1966)
[33] Read, G.: Music Notation: A Manual of Modern Practice. Taplinger Publishing Company (1979)
[34] Rebelo, A., Capela, G., Cardoso, J.: Optical recognition of music symbols: A comparative study. International Journal on Document Analysis and Recognition (IJDAR) 13(1), 19–31 (2010)
[35] Rebelo, A., Fujinaga, I., Paszkiewicz, F., Marcal, A., Guedes, C., Cardoso, J.: Optical music recognition: state-of-the-art and open issues. International Journal of Multimedia Information Retrieval 1(3), 173–190 (2012)
[36] Rebelo, A.M.: Robust Optical Recognition of Handwritten Musical Scores based on Domain Knowledge. Ph.D. thesis, University of Porto (2012)
[37] Roach, J.W., Tatem, J.E.: Using Domain Knowledge in Low-level Visual Processing to Interpret Handwritten Music: An Experiment. Pattern Recognition 21(1), 33–44 (Jan 1988)
[38] Sezgin, M., Sankur, B.: Survey over image thresholding techniques and quantitative performance evaluation. Journal of Electronic Imaging 13(1), 146–168 (2004)
[39] Stone, K.: Music Notation in the Twentieth Century: A practical guidebook. Norton New York (1980)
[40] Szwoch, M.: Guido: A Musical Score Recognition System. In: Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on. vol. 2, pp. 809–813 (Sept 2007)
[41] Szwoch, M.: Using MusicXML to Evaluate Accuracy of OMR Systems. In: Stapleton, G., Howse, J., Lee, J. (eds.) Diagrammatic Representation and Inference, Lecture Notes in Computer Science, vol. 5223, pp. 419–422. Springer Berlin Heidelberg (2008)
[42] Tardón, L.J., Sammartino, S., Barbancho, I., Gómez, V., Oliver, A.: Optical Music Recognition for Scores Written in White Mensural Notation. J. Image Video Process. 2009, 6:3–6:3 (Feb 2009)
[43] Timofte, R., Van Gool, L.: Automatic Stave Discovery for Musical Facsimiles. In: Lee, K., Matsushita, Y., Rehg, J., Hu, Z. (eds.) Computer Vision ACCV 2012, Lecture Notes in Computer Science, vol. 7727, pp. 510–523. Springer Berlin Heidelberg (2013)
[44] Trier, O., Jain, A.: Goal-Directed Evaluation of Binarization Methods. Pattern Analysis and Machine Intelligence, IEEE Transactions on 17(12), 1191–1201 (Dec 1995)
[45] Wolman, J., Choi, J., Asgharzadeh, S., Kahana, J.: Recognition of Handwritten Music Notation. Proceedings of the International Computer Music Conference pp. 125–127 (1992)

Critical Evaluation of Existing External Sorting Methods in the Perspective of Modern Hardware*

Martin Kruliš

Parallel Architectures/Applications/Algorithms Research Group
Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, Prague, Czech Republic
krulis@ksi.mff.cuni.cz

Abstract. External sorting methods which are designed to order large amounts of data stored in persistent memory are well known for decades. These methods were originally designed for systems with small amount of operating (internal) memory and magnetic tapes used as external memory. Data on magnetic tapes has to be accessed in strictly serial manner and this limitation shaped the external sorting algorithms. In time, magnetic tapes were replaced with hard drives which are now being replaced with solid state drives. Furthermore, the amount of the operating memory in mainstream servers have increased by orders of magnitude and the future may hold even more impressive innovations such as non-volatile memories. As a result, most of the assumptions of the external sorting algorithms are not valid any more and these methods needs to be innovated to better reflect the hardware of the day. In this work, we critically evaluate original assumptions in empirical manner and propose possible improvements.

Keywords: sorting, external sorting, hard disk, parallel

1 Introduction

Sorting is perhaps the most fundamental algorithm which reaches well beyond computer science. Along with relational JOIN operation, it is one of the most often employed data processing steps in most database systems. Despite the fact that the amount of operating memory of current desktop and server computers is increasing steadily, many problems still exist where the sorting operation cannot be performed entirely in the internal memory. In such situations, the data has to be stored in persistent storage such as hard drives and external sorting algorithms must be employed.

External sorting algorithms were designed in rather long time ago, when magnetic tapes were typical representatives of external memory. Even though the magnetic tapes are rarely used at present (except for specialized applications such as data archiving), the fundamental principles of these algorithms are still

* This paper was supported by Czech Science Foundation (GAČR) project P103/14/14292P.

being used. These principles are based on assumptions which no longer hold, most importantly the following:

- External memory has to be accessed sequentially or the sequential access is significantly more efficient.
- There is only a small amount of internal memory and the external memory is several orders of magnitude larger.
- External memory is slower than the internal memory by many orders of magnitude.

In this work, we would like to subject these assumptions to an empirical evaluation in order to critically asses their validity. We have performed a number of tests designed to determine performance properties of mainstream systems, hard drives, and solid state drives. Based on the results of these experiments, we propose a modification of existing methods in order to improve their performance.

In the remainder of the paper, we will assume that the data being sorted are represented as a sequence of items, where all items have the same size. An item has a value called *key*, which is used for sorting. We have used only numerical keys in our experiments; however, we have no additional assumptions about the keys than their relative comparability. Furthermore, we expect that a sorting algorithm produces a sequence of items where the keys are arranged in ascending order.

The paper is organized as follows. Section 2 summarizes related work. Traditional approach to the external sorting is revised in Section 3. Section 4 presents the experimental results that asses the individual operations required for external sorting. Finally, Section 5 proposes modifications to existing methods.

2 Related Work

The external sorting (i.e., sorting of the data that does not fit the internal memory and has to be stored in the external persistent memory) is one of the key problems of data processing. Sorting operations are widely employed in various domains [8] and they also form essential parts of other algorithms. Perhaps the first study of problems that require intensive utilization of external memory was presented in thesis of Demuth [5] more than 50 years ago, which focused mainly on the data sorting.

A rather broad study of various problem instances and of various sorting algorithms was conducted by Knuth [8] in 1970s. This study is presented in the *Art of Computer Programming* books, which are still considered to be one of the best educational materials regarding algorithms and data structures. Volume 3 is dedicated to sorting and searching and it describes commonly used methods of external sorting, such as multiway merging, polyphase merging, and various improvements. Many derived algorithms and methods for external data sorting [16] has been published since then, but they all share the fundamental assumptions layed down by Knuth.

The data communication between fast internal memory and slower external memory is still considered the bottleneck of large data processing [2, 11, 16]. Most algorithms are attempting to avoid this bottleneck by minimizing the number of total input-output operations employing various strategies for internal memory data caching [11]. Another widely utilized technique is to perform the external memory operations asynchronously, so they can overlap with internal data processing [2].

Originally, external sorting employed magnetic tapes as external memory. When magnetic disks arrived, they allowed (almost) random access to the data, so that one disk can store multiple data streams being merged. However, magnetic drives also have their limits and the sequential access to the data is still preferable. Hence, it might be beneficial to employ multiple disks either managed by the sorting algorithm directly or using striping to divide data among the disks evenly (e.g., by RAID 0 technology) [15]. Newest advances in the hardware development, especially the emergence of Solid State Disks (SSD) that employ FLASH memory instead of traditional magnetic recording, have been studied from the perspective of sorting algorithms. The SSD drives are particularly interesting from the energy consumption point of view [1, 14]; however, the introduced papers follow the original approach to external sorting and their main objective is the minimization of I/O operations.

Sorting of large datasets can be also accelerated employing multiple computers interconnected with some networking technology. Clusters of computers may have more combined internal memory than individual servers, thus a distributed solution may even achieve superlinear speedup in some cases. On the other hand, distributed sorting methods introduce new problems, such as communication overhead or load balancing [12]. Furthermore, the distributed sorting is an integral part of Map-Reduce algorithm [4], which is being widely used for distributed data processing on a large scale.

Internal sorting methods are an integral part of the external sorting, since it may be quite beneficial to pre-sort the data partially by chunks that fit the internal memory. Initially, the internal sorting methods have been summarized by Knuth in the Art of Computer Programming (vol. 3) [8]. More recently, Larson et al. [9] made a study of several internal sorting methods and asses their applicability for the initial part of external sorting.

In the past decade, the hardware development has employed parallel processing on many levels. A practical combination of internal sorting methods which takes advantage of parallel features of modern CPUs was proposed by Chhugani [3]. We have also observed an emergence of platforms for massive data parallel processing (like GPGPUs), and so several different algorithms were developed for manycore GPU accelerators [13, 10]. Finally, we would like to point out the work of Falt et al. [6] which proposes an adaptation of Mergesort for in-memory parallel data streaming systems. This work is particularly interesting, since we believe that a similar approach can be adopted for external sorting.

3 Traditional Approach to External Sorting

The traditional algorithms of external sorting have two major parts. The first part employs methods of internal sorting to form *runs* – long sequences of ordered values. The second part merges the runs into one final run, which is also the final result of the sorting problem. The algorithms that employ this approach may vary in details, such as how the runs are generated or how many runs are being merged in each step of the second part.

3.1 Forming Runs

Perhaps the most direct method for creating a run is the utilization of internal sorting algorithms, like Quicksort [7] for instance. The application allocates as large memory buffer as possible and fill it with data from the disk. The data in the buffer are sorted by an internal sorting algorithm and written back to the hard drive as a single run. These steps are repeated until all input data are pre-sorted into runs.

Direct application of internal sorting generates runs, which length corresponds to the internal memory size. Even though the intuition suggests that this is the optimal solution, Knuth [8] describes an improvement, which can be used to increase the average length of the runs. This improvement is based on adjusted version of Heapsort [17] algorithm – i.e., an internal sorting algorithm that employs regular heaps (sometimes also denoted as priority queues).

The algorithm also utilizes as much internal memory as possible. At the beginning, input data are loaded into the internal memory buffer and a d-regular heap is constructed in-place using the bottom-up algorithm. Than the algorithm performs iteratively the following steps:

1. Minimum m (top of the heap) is written to the currently generated run (but not removed from the heap yet).
2. New item x is loaded from the input data file. If $key(m) \leq key(x)$, the x replaces the top of the heap (item m) and the heap is reorganized as if the *increase-key* operation has been performed on the top. Otherwise, the *remove-top* operation is performed on the heap, so its size is reduced by 1, and the x item is stored at the position in the main memory buffer vacated by the heap.
3. If the main heap becomes empty in the previous step, the currently generated run is completed and a new run is started. At the same time, the main memory buffer is already filled with input data values, so a heap structure is constructed from them.

When the input file ends, the algorithm empties the heap to the current run by repeating the *remove-top* step. After that, the remaining data outside of the heap are sorted by internal sorting and saved as another run.

Let us emphasize two important implementation details. Despite the fact the algorithm operates on individual items, the external I/O operations are always performed on larger blocks of items and thus both data input and data output is buffered. Furthermore, the explicit construction of a new heap (in step 3) can be amortized into step 2, so that when an item x is placed outside the heap, one step of bottom-up construction algorithm is performed. This way a secondary heap is constructed incrementally as the first heap shrinks.

If the keys exhibit uniform distribution, the two-heap algorithm can generate runs with average length $2N$, where N is the number of items stored in the internal memory buffer. This fact is explained in detail by Knuth [8] and we have also verified it empirically.

3.2 Merging Runs

The second part of the external sorting performs the merging of the runs. The main problem is that the number of runs being merged simultaneously may be limited. In the past, the computers have limited number of magnetic tapes which can be operated simultaneously. At present, operating systems have limited number of simultaneously opened files and we can perform only limited number of concurrent I/O operations to the same disk. For these reasons, we can read or write only N data streams (i.e., runs) at once. In the following, we will use the term data stream as an abstraction, which could be related to a magnetic tape or to a file on a hard disk, in order to simplify technical details such as opening and closing files.

The most direct approach to this problem is called two-phase merging. $N-1$ data streams are used as inputs and one data stream is used as output. At the beginning, the runs are distributed evenly among the input streams. The algorithm then alternates two phases (depicted in Figure 1), until there is only one run in a single data stream. In the first phase, the runs are merged from the input streams into output stream. In the second phase, the runs from the output stream are distributed evenly among the input streams.

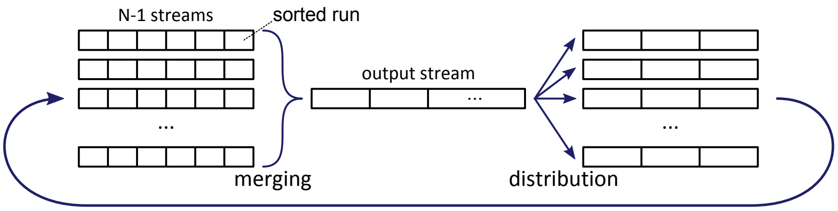


Fig. 1. The principle of two-phase merging

The merging phase takes one run from each fo $N-1$ input streams and merges them together into one run that is written to the output stream. The merging of runs is performed in the internal memory again. It can be implemented hierarchically or by using a priority queue for instance.

The main disadvantage of two-phase merging is the necessity of distributing the runs in the second phase. We can integrate the distributing phase with the merging phase as follows. If the data streams cannot be easily replaced (e.g., such as in case of magnetic tapes), we can reduce the number of input streams to $N/2$ and the remaining $N/2$ streams utilize as output streams. Hence, the merged runs are distributed in a round robin manner among the output streams as they are created. Unfortunately, the number of simultaneously merged runs is reduced from $N - 1$ to $N/2$, wich increases the total number of iterations performed by the algorithm.

If the data streams can be replaced effectively and efficiently (e.g., a file can be closed and another file can be opened), the distribution of the runs can be performed also in the merging phase whilst $N - 1$ input streams are used. Furthermore, it might be possible to utilize more than $N - 1$ input streams, if the streams can be opened/closed or sought fast enough.

3.3 Polyphase Merging

There is yet another way how to amortize the distribution phase in the merging phase. This method is called *polyphase* merging, since it repeats only the merging step and the distribution of the runs is performed naturally. The merging of the runs is also performed from $N - 1$ input streams into one output stream, but the initial distribution of the runs is not even. When one of the input streams is emptied, this stream becomes new output stream and the output stream is included among the input streams.

The distribution of the runs has to be carefully planned in order to achieve optimal workload. The optimal run distribution for the last merging step is that each input stream has exactly one run. One step before the last step, the optimal distribution is that $N - 2$ input streams have two runs and the remaining input stream has one run. Using this pattern, the initial distribution can be planed retrospectively. An example of the merging plan for 21 runs and three streams ($N = 3$) is presented in Table 1.

	beginning	#1	#2	#3	#4	#5	#6
stream 1	13	5	0	3	1	0	1
stream 2	8	0	5	2	0	1	0
stream 3	0	8	3	0	2	1	0

Table 1. Polyphase merging of 21 runs if $N = 3$

At the beginning, the first stream holds 13 runs and the second 8 runs. The third stream is used as the output stream. Each phase merges as many runs as possible and merges them (e.g., 8 runs are merged in the first step). We may observe, that if there are two input streams, the distribution of the runs follow the Fibonacci sequence. If the total number of the runs is not suitable for optimal distribution, some streams may be padded with virtual runs wich do not require merging.

4 Experiments

In this section, we present performance experiments that asses individual operations of external sorting – especially the internal sorting, the performance of the heap data structure, and the disk I/O operations.

The internal sorting experiments were conducted on a high-end desktop PC with Intel Core i7 870 CPU comprising 8 logical cores clocked at 2.93 GHz and 16 GB of RAM. Additional tests were performed on 4-way cache coherent NUMA server with four Intel Xeon E7540 CPUs comprising 12 logical cores clocked at 2.0 GHz and 128 GB RAM. The RHEL (Red Hat Enterprise Linux) 7 was used as operating system on both machines.

The I/O tests were conducted on three storages. Tests denoted *HDD* were measured using one Segate Barracuda HDD (2 TB, 7,200 rpm) connected via SATA II interface. Tests denoted *SSD* were conducted on 6 solid state drives Samsung 840 Pro (512 GB each) in software RAID 0 configuration. Finally, tests denoted *array* used Infortrend ESDS 3060 enterprise disk array which comprised 2 SSD disks (400 GB each) and 14 HDDs (4 TB, 7,200 rpm) connected in RAID 6. The array was connected via dual-link 16 Gbit Fiber Channel. Again, the RHEL 7 was used as operating system and XFS was used as the file system.

4.1 Internal Sorting

In order to asses the benefits of the heap data structure for both generating the runs and for multiway merging, we compare Heapsort with conventional sorting methods. Before we can do that, we need to select optimal heap degree (i.e., branching degree of the tree that represents the heap) for this task.

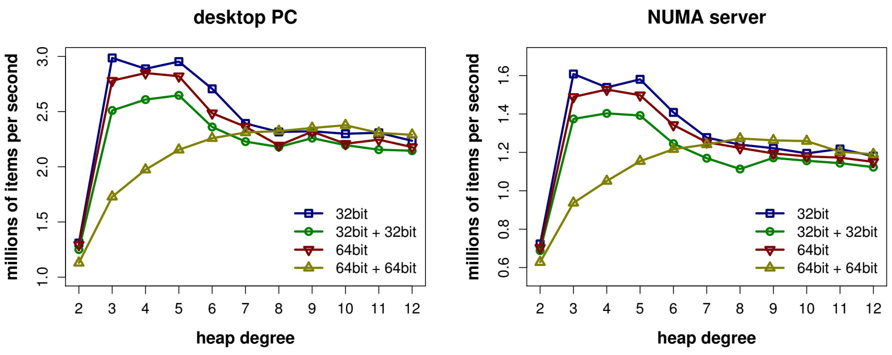


Fig. 2. Heapsort throughput results for various degrees of regular heaps

Figure 2 presents the measured throughput of Heapsort algorithm with various heap degrees on 1 GB of data. The experiments were conducted using four different item/key sizes: 32-bit (integer) keys, 32-bit integers with additional 32-bit payload, 64-bit integer keys, and 64-bit keys with 64-bit payload. The payload simulates additional index or pointer associated with the key which refers to additional (possibly large) data properties of the item.

The results indicate, that 2-regular heaps (which are often used as a default) have poor performance. Much better choice are degrees between 3-5, which reduce the height of the tree. On the other hand, the largest (16 B) items have exhibited a slightly different behaviour, which we were unable to explain so far. A more extensive study of this data structure has yet to be conducted. In the following experiments, we have used the optimal heap degree for each situation.

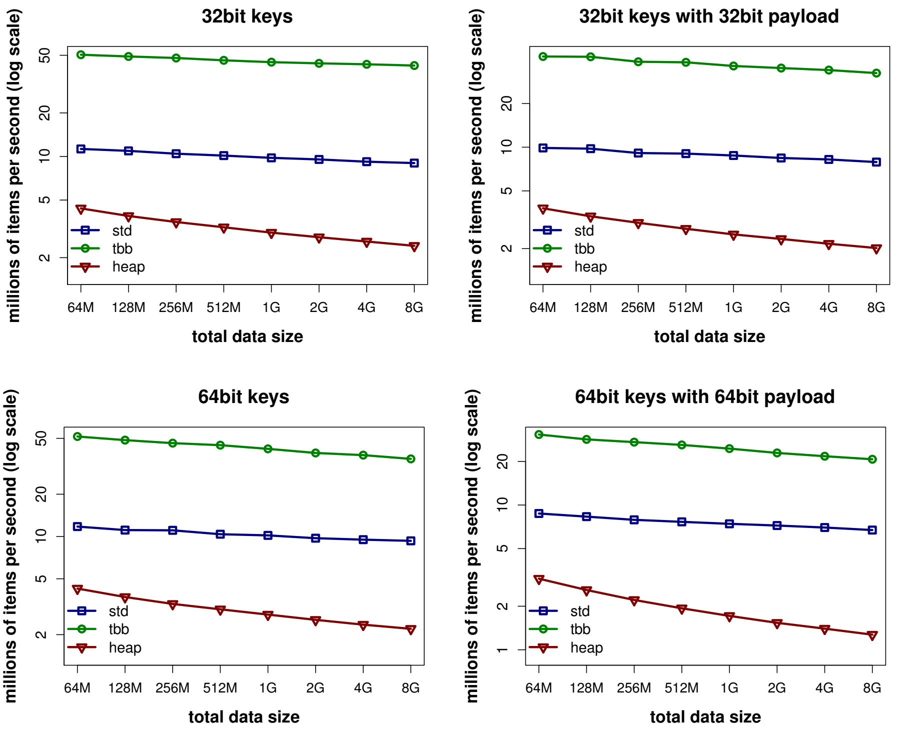


Fig. 3. Internal sorting throughput results

In the internal sorting experiments, we compare the Heapsort algorithm with serial Quicksort implemented in C++ STL library (`std::sort`) and parallel sorting algorithm implemented in the Intel Threading Building Blocks (TBB) library. These algorithms may not be the fastest possible, but they present an etalon, which can be used for further comparisons.

The results depicted in Figure 3 show that the Heapsort is several times slower than the two other methods and its performance degrade more rapidly on larger datasets. Furthermore, the TBB sort is expected to scale with increasing number of cores, but no parallel implementation of Heapsort exists to our best knowledge and we believe that it is not feasible with current CPU architectures.

We have conducted the experiments on the NUMA server as well. The server CPUs have lower serial throughput, which is compensated by the number of cores. Hence, the sorting throughput of TBB algorithm is comparable, but the `std::sort` and the Heapsort exhibit significantly lower performance.

4.2 Sequential Data Reads

The existing external sorting methods are designed to access data sequentially. Hence, our first set of experiments measure the sequential read operations. Let us emphasize, that we use sequential access to the file that holds the data; however, the underlying storage device may fragment the file or otherwise scatter the data.

The following experiments perform sequential reads from binary data files using blocks of fixed size. The data are read using 1-8 threads concurrently, where each thread has its own file in order to avoid explicit synchronization on application level. In every case, the application reads total amount of 64 GB (each thread has an equal share). We have used three APIs for file management – traditional blocking API (*sync*), asynchronous API (*async*), and memory mapped files (*mmap*) – in order to asses their overhead in different situations.

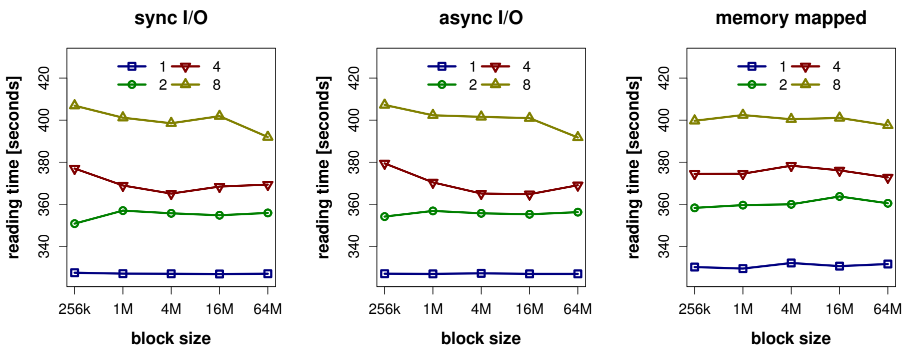


Fig. 4. Reading 64 GB sequentially from a single magnetic hard drive

Figure 4 presents the results measured using commodity hard drive. In all cases, the single-threaded version outperforms the multi-threaded versions, since one thread is capable of sufficiently utilize the drive. Furthermore, the serial reading throughput was reaching 200 MBps for all selected block sizes.

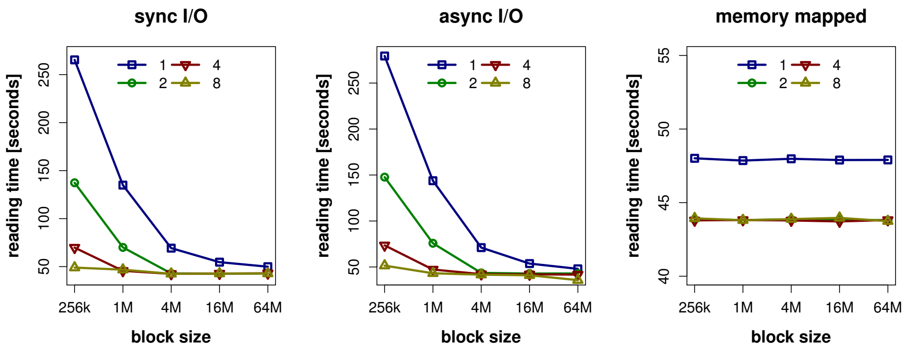


Fig. 5. Reading 64 GB sequentially from 6 SSD disks in RAID 0

Figure 5 presents the results measured on a RAID 0 array of six SSD disks. Since these disks have much greater combined throughput and much lower latency, both sync and async API demonstrate significant overhead on smaller blocks. Furthermore, multiple threads may take advantage of the high processing speed of the disks and achieve better performance. The best reading speed was approaching 1.5 GBps.

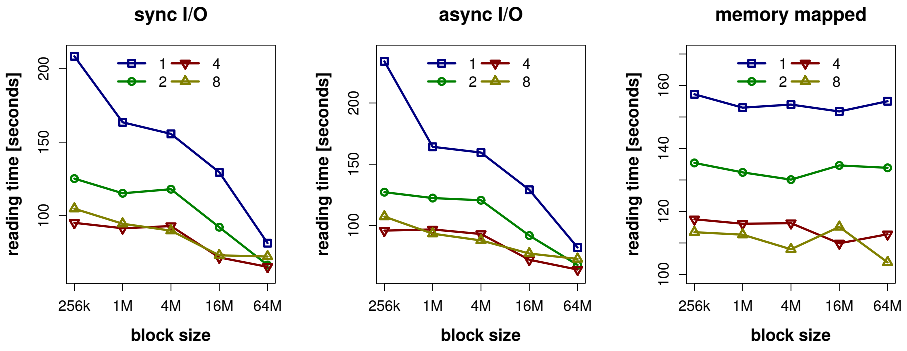


Fig. 6. Reading 64 GB sequentially from an enterprise disk array

The last set of experiments conducted on an enterprise disk array is depicted in Figure 6. The RAID nature of the array provide much better performance than a single drive, but the additional overhead imposed by the data redundancy and internal storage control makes the array less efficient than the SSD drives. The array is also much less predictable, since the I/O scheduling algorithm of the host system is probably clashing with the internal scheduler of the array.

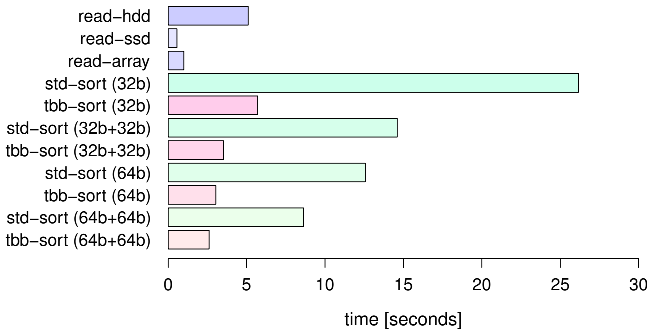


Fig. 7. Comparison of processing times of 1 GB of data (i.e. 256M of 32-bit items, 128M of 32 + 32-bit and 64-bit items, and 64M of 64 + 64-bit items

Finally, let us compare the data reading times with internal sorting times. Figure 7 summarizes the results of reading and sorting operations performed on 1 GB of data. In general, the internal sorting is slower than reading from persistent storage. Furthermore, the reading operation is expected to scale linearly with the data size, while the sorting has time complexity $\Theta(N \log N)$ and so it will get even slower (w.r.t. reading) when larger data buffers are used.

4.3 Random Data Access

In the merging phase, the data streams are usually loaded from the same storage device. In such case, the device must load data from multiple locations in short order. Hence, we would like to compare random access with the sequential access.

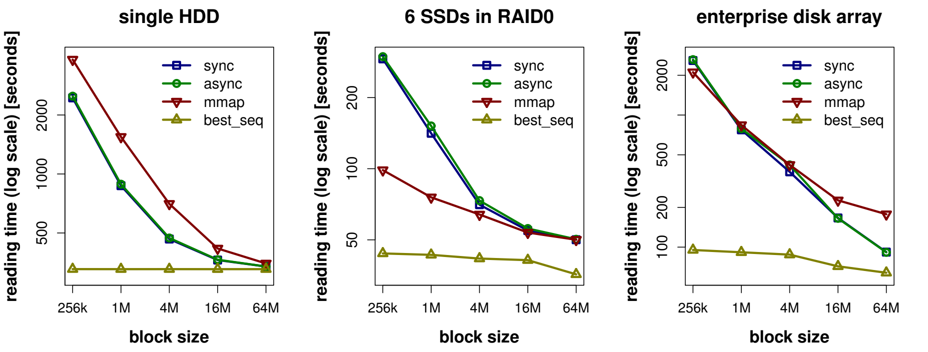


Fig. 8. Reading 64 GB as a random permutation of blocks

Figure 8 presents results of reading times, when the data are loaded in compact blocks, which are accessed in random order. The *best_seq* values are reading times of the best sequential method measured in the previous experiments, so that we can put the random access times in the perspective. The results suggests that in every case, the random access overhead can be minimized if we use sufficiently large blocks. In case of a single HDD, blocks of tens of megabytes are sufficient. In case of faster devices, large blocks are more advisable.

5 Conclusions

The experimental results indicate that using heaps for increasing the length of initial runs has negative impact on the performance. The internal sorting takes more time than data loads, thus the optimization of the internal sorting methods become a priority. Another important fact yielded by the empirical evaluation is that the random access could be nearly as fast as sequential access when the data are transmitted in large blocks. Hence, we can place as many streams as required on the same storage (even in the same file) and access them simultaneously. Based on the evidence, we propose the following:

- The internal sorting used for generating runs should utilize the fastest (parallel) algorithm possible. The length of the runs are no longer first priority.
- Modern servers of the day have hundreds of GB operating memory and tens of TB storage capacity. Hence, if the sorted data fit the persistent storage, the first phase will generate hundreds of runs at most.
- The merging phase should process all generated runs in one step. Current operating systems can work with hundreds separate files and the storage can handle simultaneous (and thus random) access to all these streams without a significant drop in performance.

We have established that traditional methods of external sorting presented by Knuth are obsolete. New methods should focus more on optimizing the internal sorting algorithms and efficient hierarchial merging than on the number of I/O operations. In the future work, we would like to adopt a data stream sorting algorithm of Falt et al. [6] for external merging.

References

1. Beckmann, A., Meyer, U., Sanders, P., Singler, J.: Energy-efficient sorting using solid state disks. Sustainable Computing: Informatics and Systems 1(2), 151–163 (2011)
2. Bertasi, P., Bressan, M., Peserico, E.: psort, yet another fast stable sorting software. Journal of Experimental Algorithmics (JEA) 16, 2–4 (2011)
3. Chhugani, J., Nguyen, A.D., Lee, V.W., Macy, W., Hagog, M., Chen, Y.K., Baransi, A., Kumar, S., Dubey, P.: Efficient implementation of sorting on multicore simd cpu architecture. Proceedings of the VLDB Endowment 1(2), 1313–1324 (2008)
4. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
5. Demuth, H.B.: Electronic data sorting. Dept. of Electrical Engineering (1956)
6. Falt, Z., Bulánek, J., Yaghob, J.: On parallel sorting of data streams. In: Advances in Databases and Information Systems. pp. 69–77. Springer (2013)
7. Hoare, C.: Quicksort. The Computer Journal 5(1), 10 (1962)
8. Knuth, D.E.: Sorting and Searching. Addison-Wesley (2003)
9. Larson, P.A.: External sorting: Run formation revisited. Knowledge and Data Engineering, IEEE Transactions on 15(4), 961–972 (2003)
10. Merrill, D.G., Grimshaw, A.S.: Revisiting sorting for gpgpu stream architectures. In: Proceedings of the 19th international conference on Parallel architectures and compilation techniques. pp. 545–546. ACM (2010)
11. Nyberg, C., Barclay, T., Cvetanovic, Z., Gray, J., Lomet, D.: Alphasort: A cache-sensitive parallel external sort. The VLDB Journal – The International Journal on Very Large Data Bases 4(4), 603–628 (1995)
12. Rasmussen, A., Porter, G., Conley, M., Madhyastha, H.V., Mysore, R.N., Pucher, A., Vahdat, A.: Tritonsort: A balanced large-scale sorting system. In: Proceedings of the 8th USENIX conference on Networked systems design and implementation. pp. 3–3. USENIX Association (2011)
13. Satish, N., Harris, M., Garland, M.: Designing efficient sorting algorithms for many-core gpus. In: Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. pp. 1–10. IEEE (2009)
14. Vasudevan, V., Tan, L., Kaminsky, M., Kozuch, M.A., Andersen, D., Pillai, P.: Fawnsort: Energy-efficient sorting of 10gb. Sort Benchmark final (2010)
15. Vengroff, D.E., Scott Vitter, J.: Supporting i/o-efficient scientific computation in tpie. In: Parallel and Distributed Processing, 1995. Proceedings. Seventh IEEE Symposium on. pp. 74–77. IEEE (1995)
16. Vitter, J.S.: External memory algorithms and data structures: Dealing with massive data. ACM Computing surveys (CsUR) 33(2), 209–271 (2001)
17. Williams, J.W.J.: Algorithm-232-heapsort. Communications of the ACM 7(6), 347–348 (1964)

Procedural Code Representation in a Flow Graph

Michal Brabec and David Bednárek

Parallel Architectures/Algorithms/Applications Research Group
Department of Software Engineering
Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, Prague, Czech Republic
{brabec,bednarek}@ksi.mff.cuni.cz

Abstract. Modern scientific computing often combines extensive calculation with complex structure of data; however, the programming methodologies and languages of high-performance computing significantly differ from those of databases. This impedance mismatch leads many projects to the use of either primitive (like JSON) or overly general (like distributed file systems) methods of data access, ignoring the decades of development in database technology. In this paper, we investigate the possibility to represent procedural code fragments using a network of operators similar to query plans used in relational database systems. Such a unified representation forms the necessary step towards an integrated computational-database platform. We propose a flow graph representation that allows us to analyze, transform and optimize applications more efficiently and without additional data. Along with the graph, we designed an algorithm that transforms a procedural code into the graph.

Keywords: compiler, graph, optimization, parallelism

1 Introduction

Modern data processing often combines complex data layouts with intensive calculations. Despite the ongoing effort in the area of no-SQL databases, the traditional relational paradigm, especially in its column-based version [2], still offers unmatched maturity and efficiency up to multi-terabyte ranges. However, the database systems were not designed with general computing in mind.

Systems based on the MapReduce paradigm allow the programmer to integrate general procedural code with a distributed data storage more easily. Despite of their popularity, MapReduce implementations may still be outperformed by parallel databases even in brute-force tasks where the sophisticated database approach has seemingly no advantage [20]. However, the same experiments also show that the performance dominance of parallel databases is limited to workloads implemented by built-in functions; as soon as user-defined functions are required, the performance falls rapidly.

This observation shows that the runtime stages of modern parallel database systems are extremely efficient even under brute-force computing load. However, this efficiency is degraded by the inability of the relational optimizers to efficiently handle procedural code fragments contained in user-defined functions [15]. Nevertheless, with a different front-end, a parallel database system may become a suitable runtime for parallel computing.

Such a front-end would have to compile procedural code into physical execution plans used in database systems. Modern databases, as well as streaming systems, use graph-based execution plans whose nodes are not limited to relational algebra operators, as shown by the successful adaptation of many relational engines to XML or RDF [17].

In this paper, we present *flow graphs* – an intermediate code capable to represent procedural code with its complex control-flow. Unlike typical intermediate representations used in compilers, the flow graphs are designed to be similar to pipeline-based models used in many database, streaming, and general parallel platforms.

Besides the introduction of flow graphs, we also describe the key algorithms which take part in the transformation of procedural code into flow graphs. The algorithms described here are applied after language-specific phases like parsing and semantic analysis and they also make use of analytical algorithms which are already frequently used in compilers.

The rest of the paper is organized as follows: After reviewing the related work, flow graphs are defined in Section 3. Section 4 presents the transformation algorithms; in Section 5, we revise possible optimizations of the flow graph.

2 Related Work

The flow graph described in this paper is similar but not identical to other modeling languages, like Petri nets [21] or Kahn process networks [14]. The main difference is that the flow graph was designed for automatic generation from the source code, where the other languages are generally used to model the application prior to implementation [16] or to verify a finished system [9]. The flow graph is similar to the graph transformation system [8], which can be used to design and analyze applications, but it is not convenient for execution. There are frameworks that generate GTS from procedural code like Java [7], though the produced graphs are difficult to optimize. The flow graph has similar traits to frameworks that allow applications to be generated from graphs, like UML diagrams [12] [4], but we concentrate both on graph extraction and execution.

The flow graph is closely related to graphs used in compilers, mainly the dependence and control flow graphs [3], where the flow graph merges the information from both. The construction of the flow graph and its subsequent optimization relies on compiler techniques, mainly *points-to* analysis [23], *dependence* testing [18] and *control-flow* analysis [22]. In compilers, graphs resulting from these techniques are typically used as additional annotation over intermediate code.

The flow graph is not only a compiler data representation, it is a processing model as well, similar to KPN graphs [13]. It can be used as a source code for specialized processing environments, where frameworks for pipeline parallelism are the best target, since these frameworks use similar models for applications [19]. One such a system is the Bobox framework [10], where the flow graph can be used to generate the execution plan similarly to the way Bobox is used to execute SPARQL queries [11].

3 Flow Graph

The *Flow graph* was designed as a compact format for representation of procedural code. Once constructed, it contains the code along with the information about its structure, including control flow and data flow, and it can be transformed back into procedural code. In this section, we define the flow graph and we explain its relation to other processing models.

A *flow graph* is a directed graph, where nodes represent operations and edges represent data exchange among the operations. The direction of the edge indicates the direction of data flow (source and sink operations). Figure 1 shows an unoptimized flow graph for a function without branches (see Listing 1 for source code). The gray nodes denote dead code and they will be removed during later optimization steps. The flow graph can become complicated once control flow is introduced – see Figure 5 for an example of a more complex graph that implements a program with a loop.

```
void Statements() {
  int a = 3;
  int b = 5;
  int c = 0;
  c = a + b;
  print(c);
}
```

Listing 1. Simple function without branches

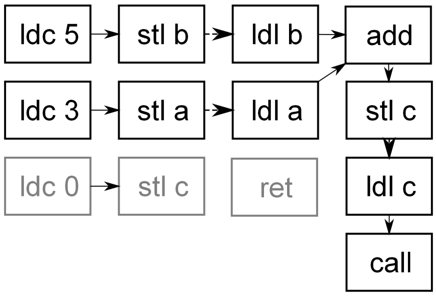


Fig. 1. Simple function transformed to an unoptimized flow graph

For a particular domain of application, a *specific flow graph language* is defined that contains a set of *basic operations* and a set of *data types*. We construct flow graphs based on such platform specific language. Both nodes and edges contain information about the represented operation or data type respectively. Each edge is connected to a specific input or output of the node according to the represented operation (the data type must be compatible).

As our research is focused on C#, we use CIL [1] instructions to specify node operations. In this paper, there are four most common instruction types: *ldl x* (load variable or constant), *stl x* (write to a variable), *ble* (conditional branch), *add* etc. (mathematical operations). We omit data types for the edges, because they are not important for the graph construction.

3.1 Execution Model

The *flow graph execution model* defines the way nodes process data and communicate. Nodes represent operations and edges represent unbounded queues (FIFO).

Operations have three states - *waiting for input*, *processing* and *inactive*. Each operation starts waiting for input, it *fires* once input is available, processes the input and produces an output. Once the input is processed, the operation again waits for data. Nodes without input (loading constants) fire at the beginning of the execution, produce data and then they become inactive.

The queues transport single values of the assigned data type (based on the edge). Nodes must always consume their input, they cannot simply check the data and leave them in the queue. For example a simple node, which adds two numbers, fires when there are data in both its incoming queues, it consumes both numbers and then is stores the result in its outgoing queue.

3.2 Special Nodes

Loops and branches of the source language are transformed into a graph of platform independent *special nodes* which interact with the data-flow carried through the *basic nodes* that perform the basic operations.

An *extended primary node* is a special version of any basic operation without parameters, like load constant value. The extended version has a single input and it restarts whenever the input contains data.

A *broadcast node* has a single input and a variable number of outputs. It represents a simple operation that creates a copy of the input for each output. This node is created whenever an operation must pass its result to multiple operations and the number of receivers defines the number of outputs. A *loop feedback node* is a special type broadcast node that distributes the positive result of a conditional branch to all extended primary nodes.

A *merge node* has a single output and three inputs. It represents an operation that accepts data from two sources and passes them to a single operation and it is used to merge data flow after a conditional branch. The node has an extra input used to get feedback from another node, generally a conditional branch. The node fires when all three inputs contain data.

A *loop merge node* is a special version of a merge node with two inputs for conditional branches, it is used to merge data in loops. The input is split into two pairs, where each contains one data input and one branch input. The node fires, whenever both inputs in a pair contain data. The node is either positive or negative, where the conditional inputs are required to be either positive or negative, for passing data into a loop or outside a loop.

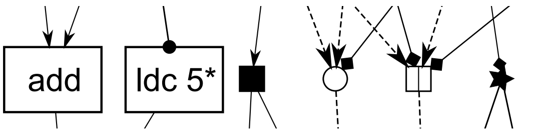


Fig. 2. Node types: basic, extended primary, broadcast, merge, loop merge, feedback

4 Flow Graph Construction

In this section, we present a two-phase algorithm that transforms a procedural code to a flow graph. The input to the algorithm is an intermediate code; in our case, the CIL. For simplicity, we assume that the CIL code was compiled from a C# source without unsafe code and goto. We also assume that the code is first subjected to a points-to analysis [23] which resolves potential aliasing problems.

The basic idea of the algorithm is that each CIL instruction is transformed into a node. Edges are generated according to the data used by each instruction. Edges correspond to the inputs / outputs of the instructions. When the basic graph is ready, we add special nodes for broadcasting and merging data according to branches and loops in the code. For simplicity, we ignore function calls in this description – see Section 4.1 for more information on function handling.

Basic Graph Construction The first step of the algorithm is to create basic nodes according to the instructions of the source code. We use basic operations in the first part, the special operations are introduced in the second part. Algorithm 1 contains all the necessary steps and it produces the basic nodes N and edges E of the graph.

We can create a node for every CIL instruction, because we defined the operations based on the instruction set (lines 1 to 3 in Algorithm 1).

Next, we analyze how instructions exchange data, either using registers or stack, and we create edges that connect the source and sink instruction. CIL instructions communicate using virtual stack and we use a stack simulator to analyze the way the instructions exchange data. Then we create edges between nodes representing instructions that exchange data (lines 4 to 6 in Algorithm 1), along with the appropriate data type. The analysis is similar when the instructions communicate through registers, only instead of stack simulator, we have to connect instructions that use the same register.

Then, we have to take into account the data passing through variables. We have to create edges between nodes representing variable access, from write to read. This step is more complex, because we have to connect every variable read with the nearest write, along all the possible execution paths, which means analyzing the control flow.

We generate basic blocks for the input code. In every basic block, we locate all the instruction that access any variable (lines 7 to 11 in Algorithm 1). First we create edges inside every block, where we connect variable writes to reads if

```
.method void  Statements() cil
  ldc.i4.3
  stloc.0
  ldc.i4.5
  stloc.1
  ldloc.0
  ldloc.1
  add
  stloc.2
  ldloc.2
  call     void print(int32)
  ret
```

Listing 2. CIL code of Statements function

```
void BranchElse() {
  int a = 3;
  int c = 0;
  if (a > 0)
    c = 4;
  else
    c = -4;
  print(c);
}
```

Listing 3. Simple branch

both access the same variable but only if the read is after the write and there is no other write between them (lines 12 to 16 in Algorithm 1). Results of this step are illustrated in Figure 1 which shows the intermediate flow graph based on the CIL code in Listing 2, generated from the source code in Listing 1.

Next, we have to connect variables between basic blocks. We locate all reads before the first write in every block, for every variable. We call them accessible reads. We connect the last write in a basic block to all the accessible reads in the blocks that follow the original, until one of them contains an instruction that writes to the same variable (lines 17 to 26 in Algorithm 1). Basically, we perform an exhaustive search through the block graph, where we stop at nodes that change the studied variable. This way the data is propagated through the control flow.

Figure 3 shows a flow graph generated from the function in Listing 3. There is one conditional jump that produced the two edges that lead to the node *ldl c*. The branch instruction must decide which input is used (Section 4). It is important that the initialization of c to 0 is identified as unused code.

Control Flow Management In this section, we present algorithms necessary to make the graph produced by the Algorithm 1 deterministic and compliant to the flow graph definition. We must make sure that the nodes have a proper number of inputs and outputs and that the branching conditions deterministically decide what value is used at every time, especially in loops. Algorithm 2 contains all the transformations for handling control flow.

We start this algorithm by locating loops in the basic block graph. We locate all the blocks L of the inner-most loop and we locate the block L_b that contains the backward conditional jump b that restarts the loop (it is sufficient to locate the backward jump, because we consider a restricted C#). We duplicate the entire block L_b as L_{b_i}, the copy drives the first iteration which is different, since it uses data initialized before the loop started (initialization of variables). See

Algorithm 1. Basic graph construction

```
Require: $I$ – set of instructions $i$
    $B$ – basic blocks
    $M$ – set containing all memory locations (variables)
    $C_i$ – all instructions consuming the output of the instruction $i$
    $R_i$ – variables read by instruction $i$
    $W_i$ – variables read by instruction $i$
Ensure: $N$ – nodes of the flow graph
    $E$ – edges of the flow graph
 1: for all $i \in I$ do
 2:     $N = N \cup \{N_i : operation(N_i) = O_i\}$ – nodes based on instructions
 3: end for
 4: for all $N_i \in N$ do
 5:     $E = E \cup \left\{E_{N_i N_j} : j \in C_i\right\}$ – edges based on instruction communication (stack)
 6: end for
 7: for all $b \in B$ do
 8:     $Load_{bm} = \{i \in b : m \in R_i \wedge w \in b < i \implies m \notin W_w\}$ – read before update
 9:     $Load_{bmj} = \{i \in b : m \in R_i \wedge j < i \wedge w \in [j, i] \implies m \notin W_w\}$ – read after write
10:     $Store_{bm} = \{i \in b : m \in W_i \wedge w \in b > i \implies m \notin W_w\}$ – last update in block
11: end for
12: for all $b \in B$ do
13:     for all $m \in M$ do
14:         $E = E \cup \left\{E_{N_j N_i} : i \in Load_{bmj}\right\}$ – edges based on variable access
15:     end for
16: end for
17: for all $m \in M$ do
18:     for all $b \in B$ do
19:         for all $n_0 \in B : next(b, n_0)$ do
20:             $E = E \cup \left\{E_{N_j N_i} : j \in Store_{bm} \wedge i \in Load_{bn_0}\right\}$ – edges between blocks
21:             if $Store_{bn} = \emptyset$ then
22:                 recursion for $\{n_1 \in B : next(n_0, n_1)\}$
23:             end if
24:         end for
25:     end for
26: end for
```

Listing 4 where i is first compared while it still has the value of 1. We locate all nodes ni inside the loop with more incoming edges than inputs (line 1 in Algorithm 2). We create loop merge nodes m for all ni (line 2 in Algorithm 2), redirect the incoming edges to the merge nodes (line 3 in Algorithm 2). We add edge E_{mni}. Finally we connect the merge nodes to the conditional branches and we pair the input coming from outside the loop to the duplicate branch in b_i in L_{b_i} and the other with the branch b in L_b (line 4 in Algorithm 2). This ensures that the first iteration takes the data from outside and all the rest take the internal data. See Figure 5 for complete loop with merge nodes.

When the loops are fitted with merge nodes, we add a loop feedback node that restarts all the nodes without input. We replace all nodes without input in

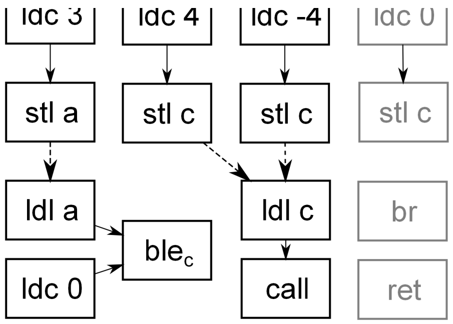


Fig. 3. Basic graph of a conditional branch

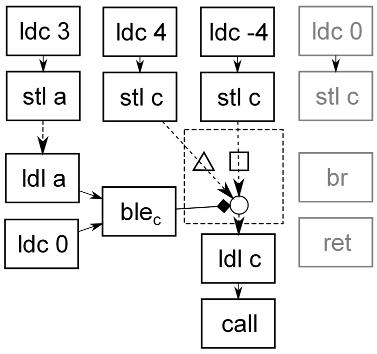


Fig. 4. Completely transformed branch

```
void SingleLoop()
{
    for (int i = 1;
         i <= 5;
         i++)
    {
        print();
    }
}
```

Listing 4. Simple loop function

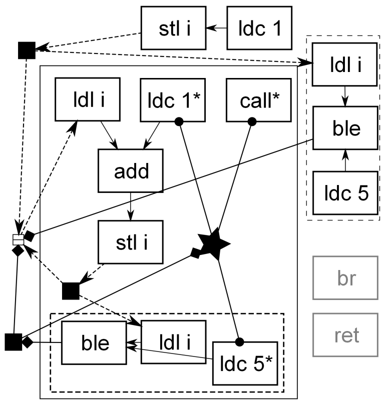


Fig. 5. Flow graph of a loop

a loop by their extended version, see Section 3.2. We connect the feedback node to the branch in Br_l and to all the extended nodes in the loop (lines 6 to 10 in Algorithm 2). Figure 5 shows a complete loop, where the Br_l and Brc_l are outlined by a dashed rectangle and the entire loop by a solid rectangle.

When all loops are transformed, we must locate all remaining nodes n with more incoming edges than inputs (line 11 in Algorithm 2), the multiple inputs are the result of conditional branches. Figure 3 shows the situation where two edges lead to a node with a single input (*ldl c*). We solve this situation by introducing a merge node m along with the edge E_{mn} (lines 12 to 13 in Algorithm 2). Then we redirect the two input edges to the m (line 14 in Algorithm 2). Finally, we have to locate the conditional branch responsible for the merge and connect it to the merge node. We can do this by following the paths from source nodes, where we locate the branch just before the paths join, ble_c (branch if $a \leq 0$) in Figure 3. The result of this algorithm is in Figure 4, where the edge with a square is the positive input and the triangle is negative.

The final step is to locate all nodes n with more outgoing edges than outputs. This is solved simply by using a broadcast node. We create a new broadcast node b, we add an edge E_{nb} and we change the outgoing edges to start in b (lines 16 to 20 in Algorithm 2).

Algorithm 2. Iteration over stripes

```
Require: $B$ – basic blocks
Ensure: $N$ – nodes of the flow graph
    $E$ – edges of the flow graph
 1: for all $\{ni \in N : \exists L \subset B \wedge ni \in L \wedge |E_{xn}| > inputs(ni)\}$ do
 2:     $N = N \cup \{m\}$ – add loop merge node
 3:     $\forall E_{xni} : x \in N \implies E = (E \setminus \{E_{xni}\}) \cup \{E_{xm}\}$
 4:     $E = E \cup \{E_{mni}\} \cup \{E_{bm}\} \cup \{E_{b_i m}\}$ – $b$ and $b_i$ are conditional jumps of the loop
 5: end for
 6: for all $\{\forall\ L \subset B : loop(L)\}$ do
 7:     $N = N \cup \{f\}$ – add loop feedback node
 8:     $E = E \cup \{E_{bf}\}$ – where $b$ is conditional jump of the loop
 9:     $\forall e \in L : extended(e) \implies E = E \cup \{E_{fe}\}$
10: end for
11: for all $\{n \in N : |E_{xn}| > inputs(n)\}$ do
12:     $N = N \cup \{m\}$ – add merge node
13:     $E = E \cup \{E_{mn}\} \cup \{E_{bm}\}$ – where $b$ is the conditional jump
14:     $\forall E_{xni} : x \in N \implies E = (E \setminus \{E_{xn}\}) \cup \{E_{xm}\}$
15: end for
16: for all $\{n \in N : |E_{nx}| > outputs(n)\}$ do
17:     $N = N \cup \{b\}$ – add broadcast node
18:     $E = E \cup \{E_{nb}\}$
19:     $\forall E_{nx} : x \in N \implies E = (E \setminus \{E_{nx}\}) \cup \{E_{bx}\}$
20: end for
```

4.1 Functions and Methods

Methods are analyzed starting with the main function and then the graph is constructed as additional methods are called. Whenever a method call is located, we create a flow graph for the called method, treating its parameters as local variables. We connect the parameters to their actual values (source variables or constants). This approach is equivalent to complete procedure integration and it is applicable only for programs whose call graph is acyclic and contains reasonable number of paths. Using additional special nodes, any program might be transformed; however, at the cost of additional operations which correspond to the call-return pairs in conventional program execution. In the supposed application domain, the general approach is probably unnecessary.

4.2 Objects and Arrays

A variable of complex data type (object or array) can contain a number of memory locations (members or elements) that can be accessed using special instructions. We treat member data of objects as separate variables where the same members of two independent objects are different variables. Arrays can be viewed as objects with a single member - data, where the data contains multiple independent values. Arrays are treated this way by many compiler algorithms [23].

5 Graph Optimizations

A flow graph produced by the algorithm presented in Section 4 is generally very big, since we transform every instruction into a single node, which is not very convenient for execution, but it can be efficiently analyzed and optimized. In this section, we shortly introduce optimizations that can produce more compact graphs.

Each node in a flow graph, as defined in Section 3, represents a basic or special operation. We introduce the *merge rules*, to allow optimizations of the graph. The merge rules define the way the operations are combined to produce complex operations. They define the behavior (source code) of the complex operation and its inputs and output along with their data types. The merge rules are added to the definition of the specific flow graph language.

A complex operation is created by merging other operations, either basic or complex, according to the merge rules defined along with the graph. The merge rules for CIL instructions contain for instance chaining of the instructions. For example, when merging the addition instr. in $A + B + C$, we create a complex operation that is equal to $\sum_1^3 In_i$.

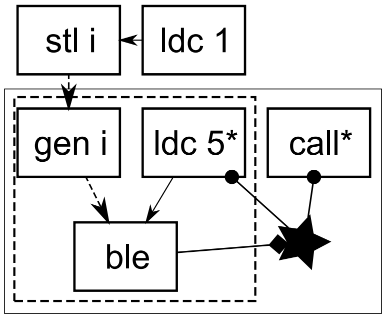


Fig. 6. Merged simple loop

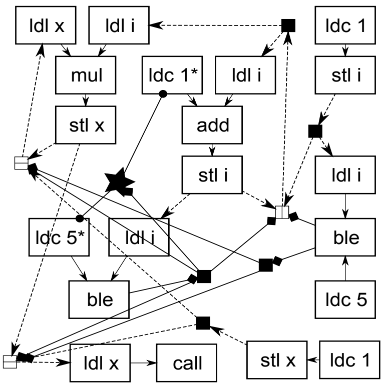


Fig. 7. Factorial computation graph

Another transformation is aimed at simple loops controlled by a single variable. Figure 5 shows a very simple loop that is controlled by the variable i, updated in every iteration. The loop creates many unnecessary dependences.

When we locate such a simple situation, we can merge the entire loop into a single node that just generates appropriate values in this case $\{1, 2, 3, 4\}$. We can utilize algorithms used in compilers [18] to locate the loops, because the graph contains all the information found in the original source code. Figure 6 shows how the loop from Figure 5 is optimized. This optimization is essential for improving the efficiency of the flow graph, compare the optimized graph to an unoptimized graph implementing the computation of a factorial, Figure 7.

6 Conclusions

We designed the flow graph to represent a procedural code along with important information about its structure and behavior. We designed an algorithm that allows us to create a flow graph for an application implemented in a subset of C# and compiled to CIL. This transformation becomes a part of a toolchain that allows the transformation of C# programs into a stream-based parallel computing platform [5]. The algorithm can be modified for other languages, like Java bytecode [6].

The flow graph is a powerful tool for application analysis and optimization. Besides generating pipeline-based execution plans, the flow graph can be used for automatic parallelization. For such use, the original flow graph may be too fine-grained – in this case, it has to be transformed using a set of merge rules to make the final parallel application efficient.

Acknowledgements

This paper was supported by Czech Science Foundation (GAČR) project P103-13-08195, and by the Grant Agency of Charles University Grant Agency (GAUK) project 122214.

References

1. TG3. Common Language Infrastructure (CLI). Standard ECMA-335, June 2005
2. Abadi, D., Boncz, P.A., Harizopoulos, S., Idreos, S., Madden, S.: The design and implementation of modern column-oriented database systems. Foundations and Trends in Databases 5(3), 197–280 (2013)
3. Allen, R., Kennedy, K.: Optimizing compilers for modern architectures. Morgan Kaufmann San Francisco (2002)
4. Balasubramanian, D., Narayanan, A., van Buskirk, C., Karsai, G.: The graph rewriting and transformation language: Great. Electronic Communications of the EASST 1 (2007)
5. Brabec, M., Bednárek, D., Malý, P.: Transformation of pipeline stage algorithms to event-driven code. In: Kurkova, V., Bajer, L., Svátek, V. (eds.) Proceedings of the 14th Conference on Information Technologies - Applications and Theory, Jasna, Slovakia, 2014. CEUR Workshop Proceedings, vol. 1214, pp. 13–20. CEUR-WS.org (2014), http://ceur-ws.org/Vol-1214

6. Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R.R., Bradshaw, R., Weizenbaum, N.: FlumeJava: easy, efficient data-parallel pipelines. In: ACM Sigplan Notices. vol. 45, pp. 363–375. ACM (2010)
7. Corradini, A., Dotti, F.L., Foss, L., Ribeiro, L.: Translating java code to graph transformation systems. In: Graph Transformations, pp. 383–398. Springer (2004)
8. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Graph transformation systems. Fundamentals of Algebraic Graph Transformation pp. 37–71 (2006)
9. Ezpeleta, J., Colom, J.M., Martinez, J.: A Petri net based deadlock prevention policy for flexible manufacturing systems. Robotics and Automation, IEEE Transactions on 11(2), 173–184 (1995)
10. Falt, Z., Kruliš, M., Bednárek, D., Yaghob, J., Zavoral, F.: Locality aware task scheduling in parallel data stream processing. In: Camacho, D., Braubach, L., Venticinque, S., Badica, C. (eds.) Intelligent Distributed Computing VIII, Studies in Computational Intelligence, vol. 570, pp. 331–342. Springer International Publishing (2015)
11. Falt, Z., Čermák, M., Dokulil, J., Zavoral, F.: Parallel SPARQL query processing using Bobox. International Journal On Advances in Intelligent Systems 5(3 and 4), 302–314 (2012)
12. Geiger, L., Zündorf, A.: Graph based debugging with fujaba. Electr. Notes Theor. Comput. Sci. 72(2), 112 (2002)
13. Geilen, M., Basten, T.: Requirements on the execution of Kahn process networks. In: Programming languages and systems, pp. 319–334. Springer (2003)
14. Gilles, K.: The semantics of a simple language for parallel programming. In: Information Processing: Proceedings of the IFIP Congress. vol. 74, pp. 471–475 (1974)
15. Guravannavar, R., Sudarshan, S.: Rewriting procedures for batched bindings. Proceedings of the VLDB Endowment 1(1), 1107–1123 (2008)
16. Josephs, M.B.: Models for data-flow sequential processes. In: Communicating Sequential Processes. The First 25 Years, pp. 85–97. Springer (2005)
17. Mayer, S., Grust, T., Van Keulen, M., Teubner, J.: An injection with tree awareness: adding staircase join to postgresql. In: Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. pp. 1305–1308. VLDB Endowment (2004)
18. Muchnick, S.S.: Advanced compiler design implementation. Morgan Kaufmann Publishers (1997)
19. Navarro, A., Asenjo, R., Tabik, S., Cascaval, C.: Analytical modeling of pipeline parallelism. In: Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on. pp. 281–290. IEEE (2009)
20. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. pp. 165–178. ACM (2009)
21. Peterson, J.L.: Petri nets. ACM Comput. Surv. 9(3), 223–252 (Sep 1977), http://doi.acm.org/10.1145/356698.356702
22. Reps, T.: Program analysis via graph reachability. Information and software technology 40(11), 701–726 (1998)
23. Sridharan, M., Bodík, R.: Refinement-based context-sensitive points-to analysis for Java. ACM SIGPLAN Notices 41(6), 387–400 (2006)

Biased k-NN Similarity Content Based Prediction of Movie Tweets Popularity

Ladislav Peška and Peter Vojtáš

Faculty of Mathematics and Physics
Charles University in Prague
Malostranske namesti 25, Prague, Czech Republic
peska@ksi.mff.cuni.cz, vojtas@ksi.mff.cuni.cz

Abstract. In this paper we describe details of our approach to the RecSys Challenge 2014: User Engagement as Evaluation. The challenge was based on a dataset, which contains tweets that are generated when users rate movies on IMDb (using the iOS app in a smartphone). The challenge for participants is to rank such tweets by expected user interaction, which is expressed in terms of retweet and favorite counts.

During experiments we have tested several current off-the-shelf prediction techniques and proposed a variant of item biased k-NN algorithm, which better reflects user engagement and nature of the movie domain content-based attributes. Our final solution (placed in the third quartile of the challenge leader board) is an aggregation of several runs of this algorithm and some off-the-shelf predictors.

In the paper we will further describe dataset used, data filtration, algorithm details and settings as well as decisions made during the challenge and dead ends we explored.

Keywords: recommender systems, content based similarity, social network, semantic web and linked data, hybrid biased k-NN, ensemble learning, user engagement, RecSys Challenge 2014, SemWexMFF team, data structures for similarity search and indexing

1 INTRODUCTION

Recommending on the web is both an important commercial application and popular research topic. The amount of data on the web grows continuously and it is virtually impossible to process it directly by a human. Various tools ranging from keyword search engines to binary intra e-shop search or product aggregators were adopted to fight against information overload. Although such tools are definitely useful, they can be used only if the user is able to specify in detail what he/she wants. Recommender systems are complementary to this scenario as they are mostly focused on serendipity – showing surprisingly interesting items the user was not aware of and thus couldn't search for them by keywords.

Many recommender systems, algorithms or methods have been presented so far. Initially, the majority of research effort was spent on the collaborative systems and explicit user feedback. Although collaborative recommender systems are generally trusted to be more accurate, they suffer from three well known problems: cold start, new object and new user problem.

New user / object problem is a situation, where recommending algorithm is incapable of making relevant prediction because of insufficient feedback about current user / object. The cold start problem refers to a situation short after deployment of recommender system, where the system cannot provide any relevant recommendation, because of insufficient data generally.

Using attributes of objects and hence content based or hybrid recommender systems can speed up learning curve and reduce the cold start problem. Moreover, content-based recommender systems can compute similarity of a new object based on its features effectively eliminating the new object problem.

Our deep belief is that quality of data used for recommendation are often more important than the algorithm which processes them. In another words rather than designing a brand new algorithm we focus on enhancing our datasets and using state-of-the-art or slightly modified algorithms to improve predictions.

The task of 2014 RecSys Challenge[1] ([17], [20]) was to predict user engagement on Twitter for movie rating tweets automatically posted from IMDb[2] (from users, who connected their IMDb and Twitter accounts). The user engagement of each tweet was defined as a sum of retweets and favorites of this tweet. Other tweet data was also made available for use, especially user rating of the movie, statistics about the user, date and time when the tweet was posted and URL to the IMDb page with the movie (data are available at Github[3]).

The dataset covers the period from February 2013 to March 2014 and contains in total almost 213,000 tweets from 24,000 users about 15,000 movies. The dataset was divided into training, test and validation subsets based on the timestamp when the tweet was created. Evaluation of the task was based on nDCG metric considering top 10 tweets for each user.

The rest of the paper is organized as follows: review of some related work is in section 2. In section 3 we provide some insight on the task and how it affected our solution [14]. Section 4 describes which recommending algorithms were used and their results. Finally section 5 concludes the paper, describes lessons learned during the challenge and points to some future work.

2 RELATED WORK

Due to the space reasons, we can provide only a short review of the related work. For general information and introduction to the recommender systems, we suggest

[1] http://2014.recsyschallenge.com/

[2] http://www.imdb.com

[3] https://github.com/sidooms/MovieTweetings/tree/master/recsyschallenge2014

Recommender Systems Handbook [19]. Several state-of-the-art recommending algorithms was used in the experiments namely Factor Wise Matrix Factorization [1], Bi-Polar Slope One [7], Item-based k-NN [8], Decision trees, Support Vector Machines4 etc. Individual results of these methods can be found in Section 4. For the majority of the algorithms we use their implementation in RapidMiner Studio5, or its Recommender extension [9].

We would like to mention also our own previous work, which affected our approach: In [12] we first considered using external semantic content to enhance secondhand bookshop recommender systems. The paper corroborated improvement of success metrics while using DBPedia content and although the following experiments shown that content-based recommending algorithm can be substantially improved, we kept using the item-item similarity method described there. In the following work on the same domain [16] we experimented with Content-boosted Matrix Factorization (CBMF) [4], which outperformed methods from [12]. Similar approach was also used in ESWC RecSys Challenge 2014 [13], however CBMF suffered from too high time complexity with rising number of examples and content attributes which detracts its usability. On the other hand the challenge winning method by Risotski et al. [18] has shown that using relatively simple recommenders combined together may provide surprisingly good results.

Our work is also related to the area of linked data. An inspiration to our previous work was the research by Ostuni et al. [10], whose aim was to develop content-based recommender system for a movie domain based sole on (multiple) LOD datasets and A. Passant [11], who developed dbRec – the music recommender system based on DBPedia dataset. Their point of view is however slightly different as they aim to develop a recommender system based solely on the semantic web datasets, but in our work (both previous and contemporary) we need to integrate external knowledge into the already known structure of the domain, thus our recommending techniques are not based on graph structure of linked data, but we aim to convert LOD into attribute-value structure.

The area of recommending on social networks is currently well covered in the research. We would like to mention e.g. Hannon et al. [6] work on recommending inter-user relationships on twitter or Esparza et al. [5] work on categorizing tweets. Nonetheless due to the specific nature of the challenge task, most of the common social network based research is not applicable.

3 ANALYSIS OF THE TASK

This section aims on discussion and initial analysis of the challenge task, focusing mainly on design choices implied by the nature of the task.

The dataset provided by the challenge organizers consisted of user, tweet and item identification, timestamp when the tweet was scraped, user rating of the movie and all information available from the tweet API excerpt from the text of the tweet (see Fig-

[4] http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/
[5] http://www.rapidminer.com

ure 1). The textual information would be extremely important for prediction of user engagement in other datasets, but automatically generated tweets from IMDb contain only template text and thus are not much relevant. The same reason makes also tweet topic categorization (e.g. CatStream [5]) irrelevant. The tweet API contains e.g. date and time of the tweet posting and statistics about the user (number of friends, followers, tweets etc.).

The user engagement is generally low throughout the dataset. The average user engagement in the training set is 0.216, over 95% of the tweets have zero user engagement and almost 80% of users received zero engagement for all of their tweets. The situation seems to be similar to the number of purchases on an e-commerce site, where most of the users only browse items, but do not buy one. Our experiments on such domain [15] suggested using extended observation of user behavior and content of the items to improve recommendation. As the monitoring of user behavior is not possible in this scenario, we focused on using available content. Another interesting question is how to interpret zero user engagement in situations where no engagement was shown also in other tweets of the same user.

3.1 Content-based Movie Datasets, Collaborative vs. Content-based

Prior to the experiments, we have conducted a small survey of available movie datasets. The IMDb, DBPedia[6], and Freebase[7] were examined concluding that IMDb contains most of the relevant information available in the other two datasets. Due to 100% coverage of items (each tweet was based on single IMDb object) and availability of querying API[8] we decided to use sole IMDb for dataset enhancements. The movie features used in our solution can be distinguished into three classes:

- Attributes describing popularity (average rating, number of awards, IMDb metascore)
- Attributes describing widespread of the movie (number of ratings)
- Attributes describing content (movie name, release date, genre, country, language, director, actors)

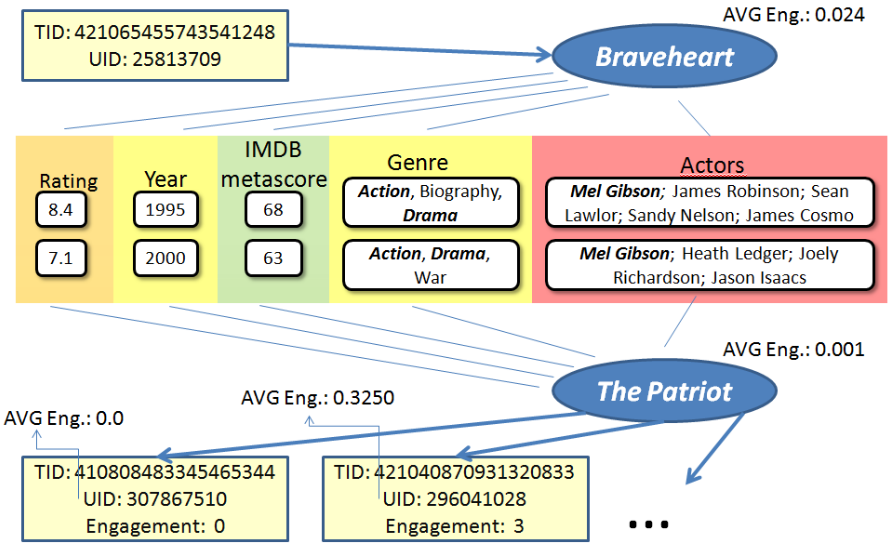
There is however some space left for further improvements especially by employing DBPedia features like dcterms:subject or widespread metrics like ingoing / outgoing links or number of Wikipedia language editions.

The test dataset contains large number of new movies unseen in the training data, so we expect that purely collaborative recommenders will not provide very good predictions. Another possible limitation is large number of zero user engagement. This caused problems to some algorithms (e.g. decision trees) as they almost constantly predicted zero for all tweets. The problem can be bypassed e.g. by filtering out (some) examples with zero engagement or by copying other tweets. The task is also not well

[6] http://www.dbpedia.org
[7] http://www.freebase.com
[8] http://www.omdbapi.com

suited for the purely content-based recommenders as there are new users in the test dataset and also for many users we have only a few tweets available.

Fig. 1. In this figure we illustrate two tweets in training data (evaluating The Patriot) and one tweet in testing data (evaluating Braveheart). Automatically generated tweets could look like “I rated The Patriot 9/10 http://www.imdb.com/title/tt0187393/#IMDb” and “I rated Braveheart 8/10 http://www.imdb.com/ title/tt0112573/#IMDb”. We download additional data about the movie and we calculate average engagement for the user and also for the movie. The task is to predict engagement for tweet in testing data.



3.2 What Does User Engagement Depend on?

We are deeply convinced that crucial for any recommending task is to estimate on which variables the final success may depend. In the current case each tweet contains almost the same text except for the name of the movie and the user rating, so we do not expect that the tweet itself can affect user engagement.

Important variable determining user engagement is probably composition of user friends and followers. Unfortunately the dataset contains only total numbers of friends and followers for each user, not the variables describing them, but we can at least employ user bias defining average engagement for each user.

Another important component is, according to our assumption, features of the movie that the tweet refers to. The sole movie ID may not be enough as there are numerous new movies in the test set and some movies are not rated with enough users. Thus we need to define content-based similarity between movies under assumption that similar movies will be treated similarly.

Also the date when the tweet was posted may be interesting since the structure of friends or followers might change over time and also popularity of the movie may evolve, but we expect that relation of previous components should be stronger and so we did not pursue this direction and leave it for the future work. The same applies also for the dependence between user rating and user engagement.

4 RECOMMENDING ALGORITHMS

In our approach we worked with two main hypotheses:

- Engagement of similar movies should be similar.
- Engagement depends on neighborhood of the current user.

In order to define inter-movie similarity, we used IMDb querying API to generate content-based attributes. We also considered using DBPedia or Freebase, but IMDb contains most of the relevant information and furthermore offers guaranteed 100% item coverage. Three types of attributes were downloaded: attributes describing popularity (average rating, number of awards, IMDb metascore), attributes describing widespread of the movie (number of ratings), attributes describing content (movie name, release date, genre, country, language, director, actors).

The second hypothesis reflects our expectation that composition of user friends and followers would greatly affect observed engagement. The twitter API contains only aggregated information (total numbers of friends and followers for each user), so we decided to use simple user bias instead of machine learning over user's friends.

Table **1.** Results of the off-the-shelf algorithms. The best achieved result for each algorithm is shown.

Method	nDCG@10
Random predictions	*0.7482*
Bi-Polar Slope One [7]	0.7652
Slope One [7]	0.7557
Factor Wise Matrix Factorization [1]	0.7556
Item-Item K Nearest Neighbors [8]	0.7604
Decision Tree	0.7494
AdaBoost with Decision Stump	0.7505
Support Vector Machines (SVM)	**0.8057**

Prior to the design of our own recommending algorithm we evaluated several off-the-shelf algorithms using RapidMiner and its Recommender extension. As expected, results of both collaborative-filtering and standard machine learning algorithms were except for SVM rather unsatisfying. Table 1 contains the best achieved results for each algorithm over different settings. Some dataset transformations (e.g. omitting records with zero engagement from the training set, transformation of user engagement in training set etc.) were also examined, but they did not significantly improve the results.

4.1 Hybrid Biased k-NN

According to the assumptions and hypothesis formulated in Section 3, we decided to pursue especially content-based movie similarity. We implemented a variant of well known k-nearest-neighbors as our main individual recommender (see Algorithm 1), where similarity of tweets is determined as content-based similarity of the respective movies. The similarity is defined as average of attributes similarities. Attribute similarity is defined according to attribute type. Similarity of *numeric* attributes (*average rating, number of ratings, number of awards, IMDb metascore* and *release date*) is defined as their difference normalized by maximal allowed distance (1).

$$sim_{x,y,maxDist} = \max\left(0, \frac{maxDist - |x - y|}{maxDist}\right) \quad (1)$$

For string attributes (movie name) the similarity is defined as inverse of relative Levenshtein distance (2). This allows us to define as similar e.g. movie series.

$$sim_{x,y} = 1 - \left(\frac{levenshtein(x,y)}{\max(lenght(x),\ lenght(y))}\right) \quad (2)$$

Finally, similarity of set attributes (genre, country, director and actors) is defined as Jaccard similarity (3). Note that nominal attributes can be dealt as sets of size 1.

$$sim_{\mathbf{x,y}} = |(\mathbf{x} \cap \mathbf{y})| \,/\, |(\mathbf{x} \cup \mathbf{y})\,| \quad (3)$$

Differences between audiences of users will be considered in the form of user bias (average value of engagement per user). The bias of current user is not important, as the evaluation is on per user base, however the bias of other users should be considered within the k-NN algorithm.

Algorithm 1: Hybrid biased k-NN algorithm: for tweet tIDte from the test set, its movie mIDte and fixed k, the algorithm first computes similarities to other movies in training set and selects k most similar movies. Then for each tweet about a similar movie the predicted ranking $\hat{e}$ is increased according to similarity s, user engagement e and bias of the tweeting user. The bias of the current movie is added in the final $\hat{e}$ prediction too (see Figure 1).

```
function HybridBiasedKNN($tID_{te}$ ∊ $TestSet$, $k$){ $\hat{e}$ = 0; extract
$mID_{te}$ from tID$_{te}$, extract uID$_{te}$ , extract $mID_{te}$ content from IMDb
  /*compute similarity for all movies */
foreach($mID$ ∊ $TrainSet$){
    S[$mID$] = similarity($mID_{te}$, $mID$);      }
/*get k most similar movies */
$\bar{S}$ = getKMostSimilar($mID_{te}$,S,k);
/*get all tweets about movies in $\bar{S}$ */
foreach({uID, mID, e, s}:
{uID, mID, e} ∊ TrainSet && $\bar{S}$[mID]=s){$\hat{e}$ += s * e / bias($uID_{te}$);}
$\hat{e}$ = bias($mID_{te}$) + ($\hat{e}$ / sum(s))
return $tID_{te}$, $\hat{e}$;                     }
```

Several meta modeling techniques were used to derive final predictions based on hybrid k-NN and off-the-shelf algorithms predictions. We have experimented with stacking with random trees, linear regression in cross-validation like setting and also tried several variants of averaging selected predictions (omitting portion of highest and lowest predictions for each tweet).

4.2 Results and Discussion

Table 2 contains results of several variants of hybrid k-NN algorithm as well as best aggregated predictions (for the sake of clarity we show only a fraction of results expressing different aspects of the data). Generally spoken, the best performing individual recommender was SVM followed by several variants of hybrid k-NN. Almost all experimented settings of hybrid k-NN outperformed other standard machine learning methods (Table 1). Surprisingly, stacking based ensemble did not predict well, probably due to dependence of the results on user, which is hard to express with decision trees. Also linear regression did not improve results, but averaging results of selected algorithms provided a significant improvement over the best individual recommenders.

While evaluating Hybrid Biased k-NN we focused mainly on the utility of each attribute, using of user bias and also ways to combine results from multiple algorithm settings. Only a fraction of our results can be shown due to the space reasons. We can state that most of the attributes used as sole similarity measure provided good results (especially IMDb metascore, director, country and language) – see Table 2.

Table 2. Results (nDCG@10) of Hybrid biased k-NN algorithm using only single content-based attribute to compute similarity.

Avg rating	0.7918	Movie name	0.7947	Language	0.8005
Awards	0.7652	Date	0.7962	Director	0.8029
Metascore	**0.8057**	Genre	0.7919	Actors	0.7930
# of ratings	0.7964	Country	0.7984		

One of our research questions was which value to use as user engagement e. The experiments showed that if using directly sum of retweets and favorites, the algorithm is highly dependent on using user bias. Another option was to use rank of the tweet in the list of current user's tweets ordered by user engagement (rank of the tweet should better reflect considered success metric). Under this setting was algorithm less dependent on using bias, however overall results were slightly worse.

Another question was how to interpret if all tweets of a user have constantly zero user engagement. Omitting those users from the training set however resulted into the decrease of performance so we suppose that even these tweets carries some negative evidence. Comparing with off-the-shelf algorithms, almost all variants of Hybrid k-NN achieved better results.

The neighborhood size k between 50 and 100 provided good results. We also tried numerous variants of combining attribute similarities within the Hybrid k-NN algorithm (omitting some attributes, weighting schemas) and ensemble methods (stacking, linear regression), but so far the best results was achieved by simple average of single attribute predictions omitting single top and bottom result – see Table 3.

Table 3. Results of Hybrid biased k-NN algorithm. *No bias* stands for omitting user and item bias from the algorithm.

Method	nDCG
Hybrid k-nn (Metascore, Language, Director, Country, Date, # ratings)	0.7927
Hybrid k-nn(Metascore, Language, Director, Country, Date, # ratings),no bias	0.7792
Linear Regression (Metascore, Language, Director, Country, Date, # ratings)	0.7913
AVG (Metascore, Language, Director, Country, Date, # of ratings), omit best and worst prediction	**0.8134**

5 CONCLUSIONS AND FUTURE WORK

In this paper, we presented our solution to the RecSys Challenge 2014. After analysis of the task, available data and current prediction techniques, we proposed a variant of k-NN algorithm leveraging content-based similarity of movies. The algorithm performed comparably with the best examined prediction techniques and the best results were achieved after averaging results of multiple runs of hybrid k-NN and SVM. Our solution was placed ninth in the challenge leader board. Some of our ideas didn’t work as we expected, namely using more advanced ensemble techniques and using ranks of the tweet instead of its user engagement resulted in worse predictions.

There are several directions of the future work. In our research so far we did not pursue temporal dependence at all which could also affect user engagement. The defined movie similarity should be also examined and tuned. We could also try to employ tweet similarity instead of movie similarity (we didn’t so far for the sake of computation effectivity). Also other procedures to aggregate results from multiple algorithms should be examined. Last but not least enhancing current dataset with e.g. DBPedia popularity measures should be considered.

Concerning data structures for similarity search and indexing – the query object is usually multimodal ([2]). Our objects have simple attributes and metrics is easy to compute. Our query is initiated by the whole user’s history, in contrast with [21], [3]. Moreover the metrics is dynamically changing because of bias is changing. It is a challenge to consider index structure for fast k-NN for online usage.

ACKNOWLEDGMENTS
The work was supported by the grant SVV-2014-260100, GAUK-126313 and P46. Our source code is available on http://www.ksi.mff.cuni.cz/~peska/hybrid_knn.zip.

6 REFERENCES

1. Bell, R.; Koren, Y. & Volinsky Ch.: Modeling relationships at multiple scales to improve accuracy of large recommender systems. *In KDD '07*, ACM, 2007, 95-104
2. Budíková P. Towards Large-Scale Multi-Modal Image Search, Doctoral thesis Masaryk University, 2013
3. Alan Eckhardt, Tomás Skopal, Peter Vojtás: On Fuzzy vs. Metric Similarity Search in Complex Databases. In FQAS 2009, Springer LNCS, Volume 5822, (2009) 64-75
4. Forbes, P. & Zhu, M. Content-boosted matrix factorization for recommender systems: experiments with recipe recommendation. *In RecSys 2011, ACM,* 2011, 261-264
5. Esparza, S. G.; O'Mahony, M. P. & Smyth, B. CatStream: Categorizing Tweets for User Profiling and Stream Filtering. *In IUI 2013, ACM,* 2013, 25-36
6. Hannon, J.; Bennett, M. & Smyth, B. Recommending Twitter Users to Follow Using Content and Collaborative Filtering Approaches. *In RecSys 2010, ACM,* 2010, 199-206
7. Lemire, D. & Maclachlan, A.: Slope One Predictors for Online Rating-Based Collaborative Filtering. *In SIAM Data Mining (SDM 2005)*
8. Linden, G.; Smith, B. & York, J.: Amazon.com recommendations: item-to-item collaborative filtering, *Internet Computing, IEEE*, 2003, 7, 76 - 80
9. Mihelčić, M., Antulov-Fantulin, N., Bošnjak, M., Šmuc, T., Extending RapidMiner with recommender systems algorithms, *In RCM 2012*, Budapest, Hungary, 2012
10. Ostuni, V. C.; Di Noia, T.; Di Sciascio, E. & Mirizzi, R. Top-N recommendations from implicit feedback leveraging linked open data, *In RecSys 2013, ACM,* 2013, 85-92
11. Passant, A. dbrec - Music Recommendations Using DBpedia *In ISWC 2010, Springer, LNCS,* 2010, 209-224
12. Peska, L.; Vojtas, P.: Enhancing Recommender Systems with Linked Open Data. *In FQAS 2013, Springer, LNCS* 8132, 2013, 483-494
13. Peska, L.; Vojtas, P.: Hybrid Recommending Exploiting Multiple DBPedia Language Editions, *In ESWC 2014 Linked Open Data-enabled Recommender Systems Challenge*, 2014
14. Peska L., Vojtas P. Hybrid Biased k-NN to Predict Movie Tweets Popularity, poster, http://2014.recsyschallenge.com/program/SemWexMFF_short_09-21.pdf
15. Peska, L. & Vojtás, P.: Recommending for Disloyal Customers with Low Consumption Rate. *In SOFSEM 2014, Springer, LNCS 8327,* 2014, 455-465
16. Peska, L.; Vojtas, P.: Using Linked Open Data to Improve Recommending on E-Commerce. *In SerSy Worlshop at RecSys 2013, Hong Kong*
17. RecSys Challenge 2014: User Engagement as Evaluation. Complete dataset. https://github.com/sidooms/ MovieTweetings/tree/master/recsyschallenge2014
18. Ristoski, P.; Mencia, E.L. & Paulheim, H.: A Hybrid Multi-Strategy Recommender System Using Linked Open Data, *In ESWC 2014*, 2014
19. Ricci F.; Rokach L.; Shapira B. & Kantor P.B., editors, Recommender Systems Handbook, Springer Science + Business Media, LLC 2011
20. Said A., Dooms S., Loni B., Tikk D. Proceedings of the 2014 Recommender Systems Challenge, http://dl.acm.org/citation.cfm?id=2668067
21. Skopal T., Bustos B.: On nonmetric similarity search problems in complex domains. ACM Comput. Surv. 43(4): 34 (2011)

UnifiedViews: Towards ETL Tool for Simple yet Powerfull RDF Data Management*

Tomáš Knap, Petr Škoda, Jakub Klímek, and Martin Nečaský

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
{knap, skoda, klimek, necasky}@ksi.mff.cuni.cz

Abstract. We present UnifiedViews, an Extract-Transform-Load (ETL) framework that allows users to define, execute, monitor, debug, schedule, and share ETL data processing tasks, which may employ custom plugins (data processing units, DPUs) created by users. UnifiedViews differs from other ETL frameworks by natively supporting RDF data and ontologies. In this paper, we introduce UnifiedViews and discuss future features of the tool towards simplicity of use for non-RDF experts. We are persuaded that UnifiedViews helps RDF/Linked Data publishers and consumers to address the problem of sustainable RDF data processing; we support such statement by introducing a list of projects and other activities where UnifiedViews is successfully exploited.

Keywords: RDF data processing, ETL, Linked Data

1 Introduction and Basic Concepts of UnifiedViews

The advent of Linked Data [1] accelerates the evolution of the Web into an exponentially growing information space where the unprecedented volume of data offers information consumers a level of information integration that has up to now not been possible.

Suppose a consumer building a data mart integrating information from various RDF and non-RDF sources. There are lots of tools used by the RDF/Linked Data community[1], which may support various phases of the data processing; e.g., a consumer may use *any23*[2] for extraction of non-RDF data and its conversion to RDF data, *Virtuoso*[3] database for storing RDF data and executing SPARQL (Update) queries [2,3], *Silk* [6] for RDF data linkage, or *Cr-batch*[4] for RDF data fusion. Nevertheless, the consumer who is preparing a *data processing task* producing the desired data mart typically has to (1) configure every such tool properly (using a different configuration for every tool), (2) implement a script for downloading and unpacking certain source data, (3) write his own script holding the set of SPARQL Update queries refining the data, (4) implement

* This work was partially supported by a grant from the European Union's 7th Framework Programme number 611358 provided for the project COMSODE

[1] http://semanticweb.org/wiki/Tools

[2] https://any23.apache.org/

[3] http://virtuoso.openlinksw.com/

[4] https://github.com/mifeet/cr-batch

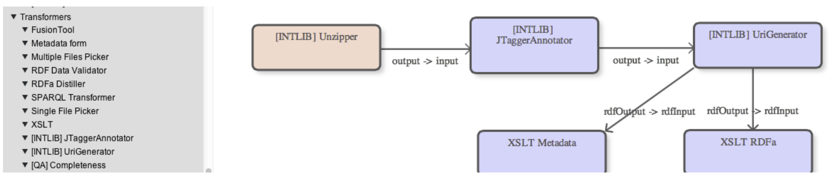


Fig. 1. UnifiedViews framework – Definition of a data processing task

custom transformers which, e.g., enrich processed data with the data in his knowledge base, (5) write his own script executing the tools in the required order, so that every tool has all desired inputs when being launched, (6) prepare a scheduling script, which ensures that the task is executed regularly, and (7) extend his script with notification capabilities, such as sending an email in case of an error during task execution.

Maintenance of such data processing tasks is challenging. Suppose for example that a consumer defines tens of data processing tasks, which should run every week. Further, suppose that certain data processing task does not work as expected. To find the problem, the consumer typically has to browse/query the RDF data outputted by a certain tool; to realise that, he has to manually launch the required tool with the problematic configuration and load the outputted RDF data to the store, such as Virtuoso, supporting browse/query capabilities. Furthermore, when other consumers would like to prepare similar data processing tasks, they cannot share the tools' configurations already prepared by the consumer.

The general problem RDF/Linked Data publishers and consumers are facing is that they have to write most of the logic to define, execute, monitor, schedule, and share the data processing tasks themselves. Furthermore, they do not get any support regarding the debugging of the tasks. To address these problems, we developed UnifiedViews, an Extract-Transform-Load (ETL) framework, where the concept of data processing task is a central concept. Another central concept is the native support for RDF data format and ontologies.

A *data processing task* (or simply task) consists of one or more data processing units. A *data processing unit* (DPU) encapsulates certain business logic needed when processing data (e.g., one DPU may extract data from a SPARQL endpoint or apply a SPARQL query). Every DPU must define its required/optional inputs and produced outputs. UnifiedViews supports an exchange of RDF data between DPUs. Every tool produced by RDF/Linked Data community can be used in UnifiedViews as a DPU, if a simple wrapper is provided[5].

UnifiedViews allows users to define and adjust data processing tasks, using graphical user interface (an excerpt is depicted in Figure 1). Every consumer may also define their custom DPUs, or share DPUs provided by others together with their configurations. DPUs may be drag&dropped on the canvas where the data processing task is constructed. A data flow between two DPUs is denoted as an edge on the canvas (see Figure 1); a label on the edge clarifies which outputs of a DPU are mapped to which

[5] `https://grips.semantic-web.at/display/UDDOC/Creation+of+Plugins`

inputs of another DPU. UnifiedViews natively supports exchange of RDF data between DPUs; apart from that, files and folders may be exchanged between DPUs.

UnifiedViews takes care of task schedulling, a user may configure UnifiedViews to get notifications about errors in the tasks' executions; user may also get daily summaries about the tasks executed. UnifiedViews ensures that DPUs are executed in the proper order, so that all DPUs have proper required inputs when being launched. UnifiedViews provides users with the debugging capabilities – a user may browse and query (using SPARQL query language) the RDF inputs to and RDF outputs from any DPU. UnifiedViews allows users to share DPUs and tasks as needed.

The code of UnifiedViews is available under a combination of GPLv3 and LGPLv3 license[6] at `https://github.com/UnifiedViews`.

2 Related Work

There are plenty of ETL frameworks for preparing tabular data to be loaded to data warehouses, some of them are also opensource[7] – for example Clover ETL (community edition)[8]. In all these frameworks custom DPUs may be created in some way, but the disadvantage of these non-RDF ETL frameworks is that there is no support for RDF data format and ontologies in the framework itself. As a result, these non-RDF ETL frameworks are, e.g., not prepared to suggest ontological terms in DPU configurations, a feature important when preparing SPARQL queries or mappings of the table columns to RDF predicates. Furthermore, these frameworks do not have a native support for exchanging RDF data between DPUs; also the existing DPUs do not support RDF data format, URIs for identifying things according to Linked Data principles. Therefore, further, we discuss the related work in the area of RDF ETL frameworks.

ODCleanStore (Version 1)[9], was the original Linked data management framework, which was used as an inspiration for ODCleanStore (Version 2)[10], the student's project implemented at Charles University in Prague and defended in March 2014 . UnifiedViews is based on ODCleanStore (Version 2). Linked Data Manager (LDM)[11] is a Java based Linked (Open) Data Management suite to schedule and monitor required ETL tasks for web-based Linked Open Data portals and data integration scenarios. LDM was developed by Semantic Web Company in Austria[12]. They currently decided to replace LDM, used by their clients, with UnifiedViews and further continue to maintain UnifiedViews together with Charles University in Prague, the Czech Linked Data company Semantica.cz s.r.o.[13], and Slovak company EEA s.r.o.[14].

6 `http://www.gnu.org/licenses/gpl.txt`, `http://www.gnu.org/licenses/lgpl.txt`

7 `http://sourceforge.net/directory/business-enterprise/enterprise/data-warehousing/etl/`

8 `http://www.cloveretl.com/products/community-edition`

9 `http://sourceforge.net/projects/odcleanstore/`

10 `https://github.com/mff-uk/ODCS/`

11 `https://github.com/lodms/lodms-core`

12 `http://www.semantic-web.at`

13 `http://semantica.cz/en/`

14 `http://eea.sk/`

DERI Pipes[15] is an engine and graphical environment for general Web Data transformations. DERI Pipes supports creation of custom DPUs; however, an adjustment of the core is needed when a new DPU is added, which is not acceptable; in UnifiedViews, it is possible to reload DPUs as the framework is running. DERI Pipes also does not provide any solution for library version clashes; on the other hand, in UnifiedViews, DPUs are loaded as OSGi bundles, thus, it is possible to use two DPUs requiring two different versions of the same dependency (library) and no clashes arise. In DERI pipes, it is not possible to debug inputs and outputs of DPUs.

Linked Data Integration Framework (LDIF)[5] is an open-source Linked Data integration framework that can be used to transform Web data. The framework consists of a predefined set of DPUs, which may be influenced by their configuration; however, new DPUs cannot be easily added[16]. LDIF provides a user interface to monitor results of executed tasks.; however, when compared with UnifiedViews, LDIF does not provide any graphical user interface for defining and scheduling tasks, managing DPUs, browsing and querying inputs from and output to the DPUs, and managing users and their roles in the framework. LDIF also does not provide any possibility to share pipelines/DPUs among users. On the other hand, LDIF provides possibility to run tasks using Hadoop[17].

3 Impact of the UnifiedViews Framework

The goal of the *OpenData.cz initiative*[18] is to extract, transform and publish Czech open data in the form of Linked Data, so that the initiative contributes to the Czech Linked (Open) Data cloud. For this effort, UnifiedViews framework is successfully used since September 2013; so far we published tens of datasets, hundreds of milions of triples; Figure 2 depicts an excerpt of the datasets (blue circles) published with UnifiedViews and the integration of these datasets (links are depicted by blue arrows, pointing from the linking dataset to the linked dataset).

Project INTLIB[19] aims at extracting (1) references between legislation documents, such as decisions and acts, (2) entities (e.g., a citizen, a president) defined by these documents and (3) the rights and obligations of these extracted entities. UnifiedViews is used in INTLIB to extract data from selected sources of legislation documents, convert it to RDF data, and provide it as Linked Data.

COMSODE FP7 project[20] has the goal to create a publication platform for publishing (linked) open data. UnifiedViews is used there as the core tool for converting hundreds of original datasets to RDF/Linked Data.

UnifiedViews framework is being integrated to the stack of tools produced by the *LOD2 project*[21]. As a result, anybody using tools from LOD2 stack, such as Virtuoso

[15] `http://pipes.deri.org/`

[16] `http://ldif.wbsg.de/`

[17] `http://hadoop.apache.org/`

[18] `http://opendata.cz`

[19] `http://www.isvav.cz/projectDetail.do?rowId=TA02010182`

[20] `http://www.comsode.eu/`

[21] `http://lod2.eu/`

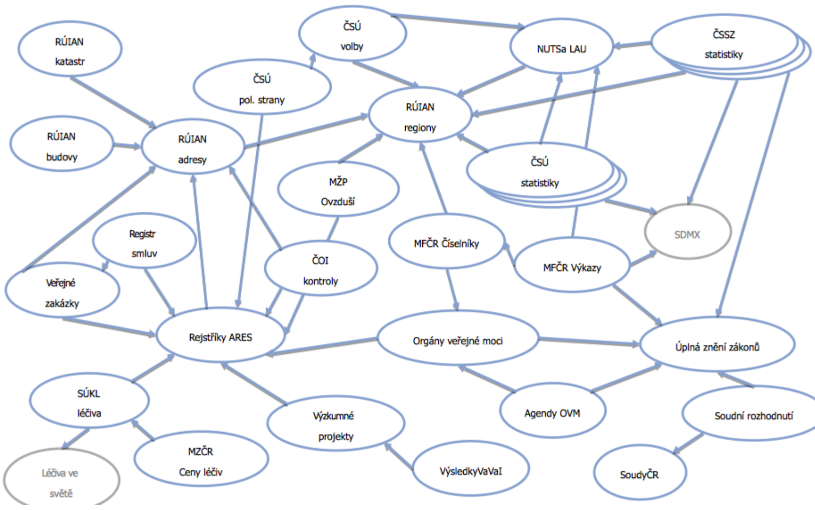


Fig. 2. Datasets published by `opendata.cz` initiative – an excerpt

and Silk, has also the possibility to use UnifiedViews. UnifiedVIews will be also used in the recently starting EU H2020 project called YourDataStories[22].

UnifiedViews framework is intended to be used for commercial purposes by companies Semantica.cz s.r.o., Czech Republic, EEA s.r.o., Slovak Republic, Semantic Web Company, Austria, TenForce, Belgium, to help their customers to prepare and process RDF data.

4 Ongoing and Future Work Towards Simplicity of Use

In this section, we introduce the ongoing and future work on UnifiedViews towards simplicity of use of the tool for non-RDF experts. Each section below describes the planned feature. In the sections below, we are talking about a *task designer* – a person who creates new or adjusts existing data processing tasks. All sections contain motivation, goals to be achieved, and at least outline how the goals will be realised.

4.1 Automated Schema Alignment and Object Linkage

Motivation. Suppose that a data processing task is created to daily extract tabular data provided by Czech Hydrometeorological Institute about the air pollution in various cities of the Czech Republic and publish them as Linked Data. Publishing data as Linked Data involves (1) alignment of the schema used for the published data with well-known schemas (RDF vocabularies) used in the Linked Data community, e.g., in Linking Open Data cloud[23] and, (2) linkage of the published objects with the objects

[22] `https://www.insight-centre.org/content/your-data-stories`
[23] `http://lod-cloud.net/`

already available in the Linked Data space, so that common objects in the datasets have the same identifiers across the datasets. As a result, Linked Data applications work on top of integrated datasets (thanks to (2)), which use common schema elements (thanks to (1)).

If the task designer is a Linked Data expert, he is able to manually integrate the data, e.g., link the RDF representation of the cities introduced in the source tabular data to the generally accepted representation of the cities – identifiers used by the LAU codes dataset[24]; based on that, Linked Data applications may not only show pollution in the particular city, but also, e.g., level of carbon emissions in that city, demographic statistics for the population in the city, number of child inhabitants in that city, etc.

Linked data experts may also ensure that published data is using well-know RDF vocabularies used in Linked Open Data community to publish certain types of data; for example, the task designer may ensure that instead of automatically generated predicate `ex:firstName` holding first names of persons, the predicate `foaf:givenName` is used. Such adjustments of the RDF data related to alignment of the schemas or object linkage are not trivial and cannot be easily done by non-experts.

Goal to be achieved. UnifiedViews should simplify Linked Data publishing for non-RDF experts by:

1. Automatically discovering that certain columns in the processed tabular represent certain types of data (e.g., cities of the Czech Republic) and automatically mapping values in this column to URIs taken from the preferred dataset for the given type of data (e.g., from the dataset with LAU codes). As a result, all datasets use the same identifiers for the same types of data, which realises the data integration and avoids costly and adhoc application integration.
2. Automatically suggest the mappings of the used RDF vocabulary terms (e.g., predicated) to well-known vocabulary terms (e.g., predicates), which increases the understandability of the data and reuse of the data by various applications.

To realise 1), first, it is necessary to identify that certain columns contain certain types of values; such identification is always probabilistic and typically based on the comparison of the name of the column with the list of names of the RDF classes and/or based on matching sample data from the considered column against known codelists, such as list of Czech cities; experiments are needed to decide the particular algorithm for identification of types among input data. Second step to realise 1) is to apply predefined Silk [6] rules for the given identified type of data within the column of the input tabular data. To realise 2), various schema matching techniques has to be experimented [4].

4.2 Hiding Sparql Queries

Motivation. Linked data expert is able to query RDF data using SPARQL query language [2,3], so for an expert it is enough to have one generic DPU, which is able to execute arbitrary SPARQL query on top of processed RDF data. Nevertheless, using

[24] `http://opendata.cz/linked-data`

SPARQL query language for rather typical and simple operations with the data, such as renaming predicates or replacing predicate's value based on a regular expression, may be considered as too heavy-weight and difficult for RDF beginners and as tedious for experienced users.

Goal to be achieved. UnifiedViews should simplify work with SPARQL query language by providing a set of DPUs for executing rather typical and simple operation on top of RDF data; such DPUs may be configured via a configuration dialog, SPARQL query behind is completely hidden from the task designer.

To realise the goal, list of typical operation should be written down and DPUs should be prepared. Discussion with the users – task designers – is crucial to focus on the most typically used operations on top of RDF data.

4.3 Autocompleting Terms from Well-known Vocabularies

Motivation. In many cases, e.g., when aligning vocabulary terms as described in Section 4.1 or when configuring DPU hiding complexity of SPARQL query language as described in Section 4.2, task designer has to define certain vocabulary terms. Since the number of well-known vocabularies is quite high, task designer may easily use wrong vocabulary term or misspell the term.

Goal to be achieved. As the task designer is configuring DPUs, UnifiedViews should suggest and autocomplete vocabulary terms from well-known Linked Data vocabularies. Task designer should be not only provided with the suggested term, but also with the description of the term, its formal definition, its recommended usage etc.

To realise this goal, data processing tasks should be prepared to populate RDF database regularly with the well-known Linked Data vocabularies – such knowledge base is then used for suggesting and autocompleting the vocabulary terms. Selected components of the DPUs' configurations, e.g., text fields, should by design support suggesting of terms from well known vocabularies – so any DPU developer may use such vocabulary autocomplete aware text field when defining configuration dialog for his DPU.

4.4 Sustainable RDF Data Processing

Motivation. As the task designer updates the task, the interconnections among and configurations of the DPUs comprising that task are adjusted. As a result, task designer may introduce errors in the definition of the task yielding in erroneous or no data produced by the task.

Goal to be achieved. UnifiedViews should address the problem of sustainable RDF data processing by allowing task designer to define for each DPU a set of SPARQL queries, which tests that the output data produced by the given DPU satisfies certain conditions. Such set of SPARQL queries for testing data outputted by the DPU plays similar role as standard JUnit tests – to test that any change to the DPU configuration did not change the produced data of that DPU in an unexpected way. Task designer should be supported with the autocomplete feature (described in 4.2) as he is specifying the SPARQL unit tests.

To realise this goal, every DPU detail should be extended with the possibility to define set of SPARQL ASK queries to realise unit testing.

4.5 Wizards for Simple Definition of Data Processing Tasks

Motivation. Defining data processing tasks typically requires detailed knowledge of the DPUs that are available in the deployed UnifiedViews instance; task designer has to know which DPUs are suitable for the task at his hand, how it should be configured and interconnected with other DPUs.

Goal to be achieved. It should be possible to define simple tasks without the knowledge of the DPUs, its configurations. UnifiedViews will contain so-called wizards, which provides task designers step by step guides for defining new data processing tasks – at least for typical types of data processing tasks, e.g, extracting tabular data and publishing it as Linked Data, or extracting data from relational databases and publishing it as Linked Data.

To realise the goal, list of typical types of data processing tasks should be written down and wizards should be prepared for such tasks. Discussion with the users – task designers – is crucial to focus on the most typical types of tasks. The idea of incorporating wizards to existing UnifiedViews frontend is as follows: when a task designer creates new pipeline, he may either manually define the task or start the wizard which will guide him through the process of task preparation; task designer may then manually finetune the definition of the task.

4.6 Assessing Quality of Produced Data, Recommendation of Cleansing DPUs

Motivation. As the goal of a task designer is to produce high quality data, the task designer should be informed about any problems in the data, e.g., w.r.t. syntactic/semantic accuracy of the produced Linked Data or completeness of the published dataset. Furthermore, if such problems may be corrected, they should be corrected.

Goal to be achieved. UnifiedViews should provide a set of DPUs assessing the quality of the produced data and set of DPUs being able to cleanse the problems in the data. UnifiedViews should also automatically recommend cleansing DPUs for data processing tasks based on the problems revealed in the data.

To realise the goal, list of quality assessment and cleansing DPUs should be implemented, being inspired by the list of data quality dimensions and metrics relevant for Linked Data [7]. The recommendation of cleansing DPUs should be based on the types of quality assessment DPUs which reported problems.

4.7 Evolution of DPUs

Motivation. DPUs may evolve as the time goes, different tasks use different versions of the same DPU. When the version of the DPU is updated, configuration used in the tasks must be also updated without the needed to reconfigure the DPU by the task designer.

Goal to be achieved. UnifiedViews must be able to cope with the changing versions of the DPUs; each new version of the DPU may bring changes to the DPU's configuration. UnifiedViews must be able to automatically convert outdated configuration, so that it may be used in the latest version of the DPU.

To realise the goal, interface of the DPUs should be extended, so that DPU developers may provide a migration method converting previous configuration to the current DPU configuration. As a result, if outdated version of the configuration is encountered and should be updated to the correct version, a sequence of these migration methods may be automatically executed by UnifiedViews (if these methods are properly provided by the DPU developer).

5 Conclusions

We presented UnifiedViews, an ETL framework with a native support for processing RDF data. The framework allows to define, execute, monitor, debug, schedule, and share data processing tasks. UnifiedViews also allows users to create custom plugins - data processing units.

We discussed future intended features of the tool w.r.t simplicity of use of the tool for non-RDF experts or those not familiar with all Linked Data vocabularies, datasets etc. For each intended feature we discussed its motivation, goals and its realisation.

We are persuaded that UnifiedViews is a matured tool, which addresses the major problem of RDF/Linked Data consumers – the problem of sustainable RDF data processing; we support such statement by introducing a list of projects where UnifiedViews is successfully used and mention two commercial exploitations of the tool.

References

1. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1 – 22, 2009.
2. S. H. Garlik, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 Query Language. W3C Recommendation, 2013. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`, Retrieved 20/03/2014.
3. P. Gearon, A. Passant, and A. Polleres. SPARQL 1.1 Update. Technical report, W3C, 2013. Published online on March 21st, 2013 at `http://www.w3.org/TR/2013/REC-sparql11-update-20130321/`, Retrieved 20/03/2014.
4. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, Dec. 2001.
5. A. Schultz, A. Matteini, R. Isele, C. Bizer, and C. Becker. LDIF : Linked Data Integration Framework. In *Proceedings of the Second International Workshop on Consuming Linked Data (COLD)*, Bonn, Germany, 2011. CEUR-WS.org.

6. J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk - A Link Discovery Framework for the Web of Data. In *Proceedings of the WWW2009 Workshop on Linked Data on the Web (LDOW)*, Madrid, Spain, 2009. CEUR-WS.org.
7. A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment for linked data: A survey. *Semantic Web Journal*, 2015.

Aspect-oriented User Interface Design for Android Applications[1]

Jiří Šebek, Karel Richta

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague, Karlovo nám. 13,
121 35 Praha 2, Czech Republic
{sebekji1,richta}@fel.cvut.cz

Abstract. This paper deals with the design of an effective Android framework that will allow a developer to create Android applications easily in a short time with the help of aspect-oriented approach. Our solution enables to deal with separated aspects like security, layout, input validation, data binding and presentation independently. Our presented framework is compared to conventional development approach of mobile applications and also is compared to the framework Aspect Faces that is also uses aspect-oriented approach, but is designed for Java EE applications. Each aspect of framework was tested and it was proven that our framework is effective in the following areas. It does not slow down the developed application according to the same application created with XML, it makes the code to be more readable, and it makes development faster, and reduces the number of code lines that developer has to write down.

Keywords aspect-oriented approach, aspect-driven design, entity inspection based approach, run-time aspect model, reduced maintenance and development efforts

1 Introduction

The main aim of this paper is to present a design of a new Android framework that will allow a developer to create Android applications with minimal effort, in a short time, and with the help of an aspect-oriented approach.

In the conventional approach, we are mixing a code of all miscellaneous aspects together in one big code. The aspect-oriented approach in a software development means that we are focusing on separated aspect. These aspects are: security, layout,

[1] This work has been partially supported by the Grant Agency of CTU No. SGS15/210/OHK3/3T/13 and partially also by the AVAST Foundation

input validation, data binding and presentation. As a result, the developer using aspect-oriented approach can write less amount of code, which is moreover reusable. We also avoid a spaghetti code (code that has tangled structure), and a redundancy of code, and other bad habits in the programming.

2 Background

Within all operation systems (OS) for a mobile device, the Android is the most expanded as shown in Figure 1. Because of that, applications targeted for Android are very desirable. As you can see from Figure 1, the ratio of applications targeted for the Android OS for mobile device is steadily growing up. This is the reason why developers cannot omit the Android in their analysis.

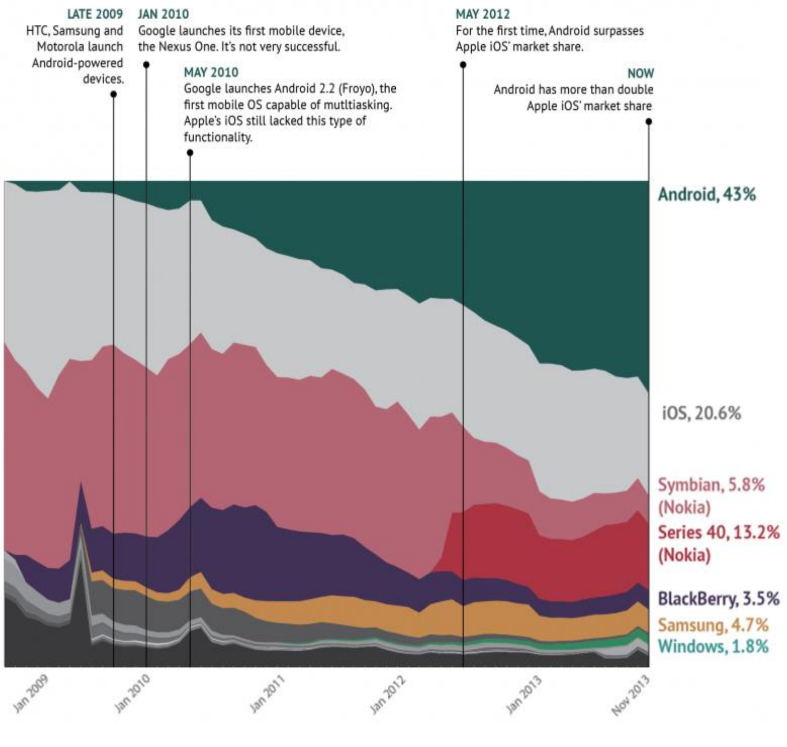


Fig 1. Distribution of all OS for mobiles (adopted from [11])

2.1 Conventional Approach

In conventional development approach of Android applications, every screen has two parts. The first part is Java class that extends Activity class and where you can place your logic and you have to connect your Java class to view. That view is a second part of the screen description and can be done in XML or generated by a program. Usually, XML choice is better because you can separate, at least, layout from remaining code. The readability depends on the developer and sometimes it's hard to maintain resulting code.

2.2 Aspect-Oriented Approach

Aspect-oriented programming (AOP) approach is a paradigm whose main goal is increasing modularity. Usually the code of applications can be separated into logical sections. These logical sections are called aspects. AOP supports separating development of any aspects option, resulting in a better code. In OOP designs, we use classes to describe only instances and their attributes. That is the representation for data only. Therefore, the best way to do things in aspect way approach is to add these additional pieces of information to the same representation (class). This is called Rich Entity Aspect/Audit Design (READ) [2,3]. Above any instance, attribute or method you can place annotation containing additional information. By this procedure, we can add all required aspects.

3 Related Works

Approach in the articles [2,3] is not the only way how to generate User Interfaces (UI) from the model. The topic of UI generated from domain objects is mentioned in [6,7]. The framework is called Meta-widget and it is based on Model driven development (MDD). The user just creates objects and puts them to Meta-widget's framework. The UI is generated according to the model. Meta-widget supports a lot of technologies from Android, Google Web Toolkit (GWT), HTML 5 (POH5), JavaScript to JSF and JSP. Meta-widget works in three basic steps. First, Meta-widget comes with a UI component native to your existing front-end. Second, Meta-widget inspects, either statically or in the run-time, your existing back-end architecture. Third, Meta-widget creates native UI subcomponents matched to the back-end. In articles [2,3], the other aspects were added based on annotations. Meta-widget adds this information based on existing back-end of any applications.

Model driven development (MDD) is based on the idea that the model should be primary centralized place for all information. This model is then compiled or transformed by another way into the deployed application code. The benefits are reduction of information in application and concentration of the structure of information into one place. The disadvantages can be adaptation and evolution management [8]. This approach does not go well with OOP, because we need to maintain the interconnection between the models with the back-end of the application. There exist another tools, how to describe an additional information. These tools are called in the MDD the Domain-Specific Languages (DSL). Sometimes, they are informally called mini-languages, because they describe the additional information inside the other language. There are a wide variety of DSL. Domain-specific languages can be a visual diagramming language, programmatic abstractions, declarative language (OCL) or even whole languages like XSLT. As we can see, some of them evolve into the programming tools that are frequently used (XSLT).

Generative programming (GP) is a specific type of a programming that generates the source code from domain-specific code. The goal is to improve productivity of developer, make the way between application code and domain model, support reuse, adaptation, and simplify management of components [12].

Meta-programming (MP) is a technique, which allows the developer to modify the structure and the behavior of the applications at the run-time. The reflection is one of the options how to implement the MP [5]. The developers can inspect the classes, the fields, the methods at the compile time and they do not even have to know their names at the compile time. The MP allows developers to adapt the application to the different situations. The bottleneck of this solution is the performance. The applications are significantly slower with the MP and are harder to test or debug then the applications without the MP. To deal with this problem the developer can use some cache.

4 Design of Aspect-Oriented Framework

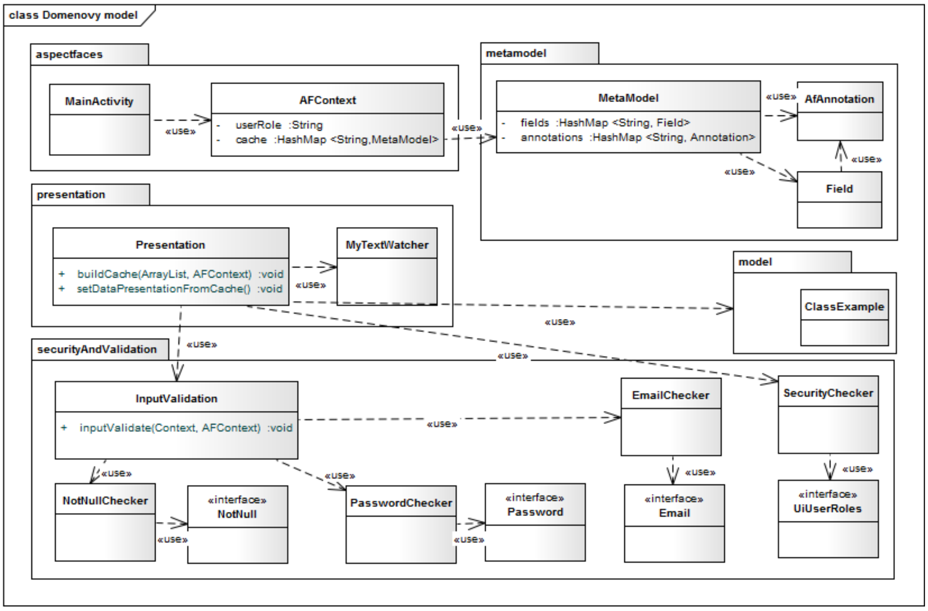


Fig. 2. Analytic model of classes

The analytic model of classes is a diagram which captures a general static view of the application. The purpose of this is to illustrate types of objects, variables and their relationships. Figure 2 shows class diagram of our framework. It does not contain all of the files (classes) because there would be much more objects and the diagram would not be easy to read, but it contains all packages and main functionality.

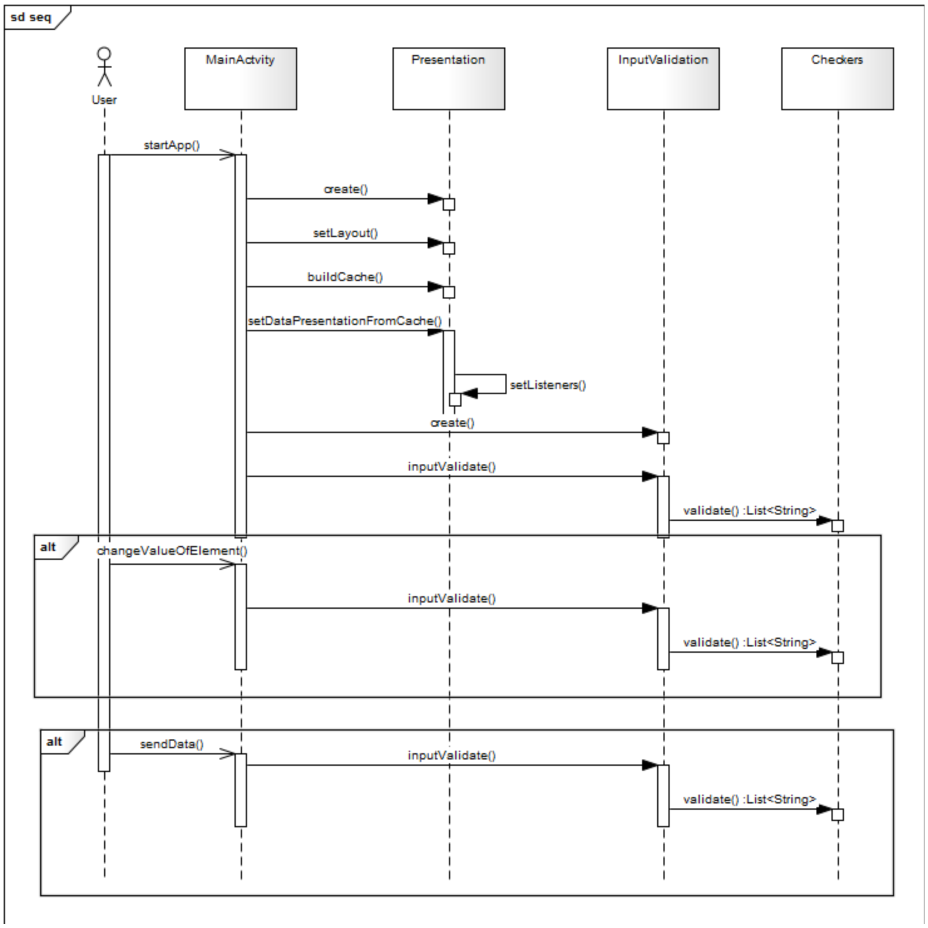
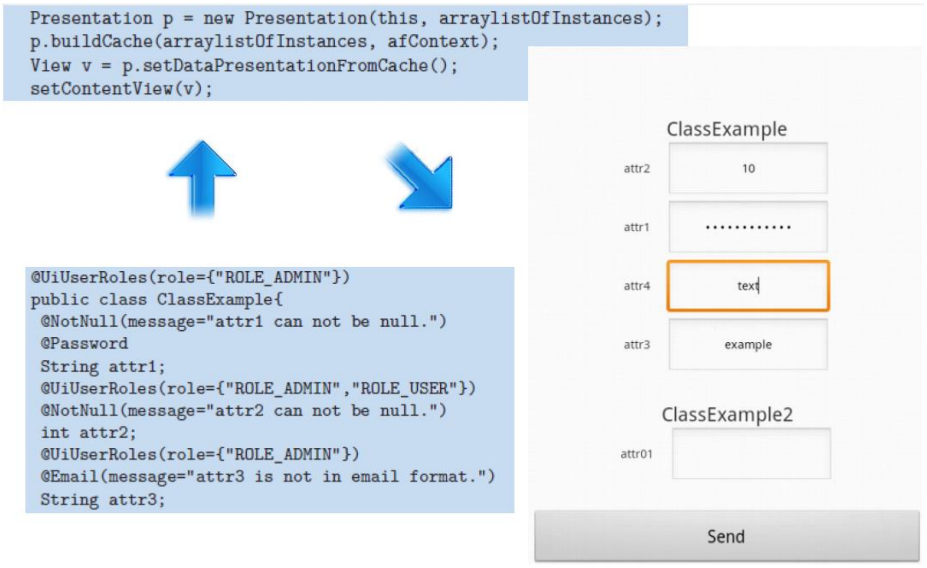


Fig. 3. Sequence diagram of the framework

The sequence diagram is used for a visualization of interactions between processes (objects). It also displays the right order of these interactions. It is a behavioral type of a diagram. Therefore, it is the best choice for showing how the framework works. The diagram includes parallel vertical lines called lifelines and horizontal arrows that represent the messages exchanged between them in the right order as they appear. In figure 3, there is shown basic sequence diagram of our framework. It shows what happens from the start of the application using the framework.

The mandatory action is the creation of a new *Presentation* object. Then, the main activity calls methods *buildCache()*, and *setDataPresentationFromCache()* of this object. *BuildCache()* method creates new cache from given instances. UI can be created much faster from cache, which includes information in hashmaps. The time to retrieve this information is then constant. It makes final application much faster, for example in the case of a fragment style application, where user is often sliding between screens that he/she already visited. When cache is already created, it is not created again. *SetDataPresentationFromCache()* method creates whole UI from cache. It means layout, data presentation and data binding. The other options of the framework are voluntary, like on the diagram. If the developer wants to validate default data in created instances, he/she just creates the *InputValidation* object and call *inputValidate()*. The framework will care about the rest via created rich entity (normal instance with attributes and with annotations). In this diagram, it is also captured when the user changes the value of element it will call the listener which will call the *inputValidate()*. If the user sends the form with data, it will also call *inputValidate()*.



```
Presentation p = new Presentation(this, arraylistOfInstances);
p.buildCache(arraylistOfInstances, afContext);
View v = p.setDataPresentationFromCache();
setContentView(v);
```

```
@UiUserRoles(role={"ROLE_ADMIN"})
public class ClassExample{
 @NotNull(message="attr1 can not be null.")
 @Password
 String attr1;
 @UiUserRoles(role={"ROLE_ADMIN","ROLE_USER"})
 @NotNull(message="attr2 can not be null.")
 int attr2;
 @UiUserRoles(role={"ROLE_ADMIN"})
 @Email(message="attr3 is not in email format.")
 String attr3;
```

ClassExample

attr2: 10

attr1: ••••••••••••

attr4: text

attr3: example

ClassExample2

attr01:

Fig. 4. Usage of the framework

4.1 Usage of the Framework

In the Figure 4, it is shown how our framework works all together. The basic element is entity (Java object) enriched by other information in form of annotations. Then we have to call the framework and also give over *arrayList* of instances. Everything is generic; framework does not need to know exact type of object which was inserted into *arrayList*. The last part in the figure 4 shows the result after launching the application.

5 Comparison of Aspect-Oriented Approach and Conventional Approach for Android Platform

In the conventional approach, the presentation layer was implemented in XML, the data binding in Java, input validation in Java, layout in XML and security in Java. The project, which was created in conventional way, was developed to implement one form that is the same as the first one in the example of aspect-oriented approach with four attributes (created with our framework). All aspects was coded in common way and not stored in annotations like in the aspect-oriented approach. If we want to do another form, we will have to write the similar amount of code. This code is redundant and is making application hard to maintain. The big difference against AOP is in security, where in aspect-oriented approach we just do not create particular element. Here, we create all elements and then we are changing visibility if the user has particular user role. The advantages of the aspect-oriented approach are shown in Table 1.

Table 1. Comparison of AOP and conventional approach

Feature	AOP	Conventional approach
Reuse	yes	no
Run-time approach	Yes	no
Reduce code	Yes	no
Better to maintain	Yes	no
Separated each aspects	Yes	no
Readable code	Yes	No (depends on developer)
Time to launch the form (average)	119,5ms	193,1ms
Standard deviation (std)	5,35ms	14,7ms
Lines of code (LOC)	29	495

Here, we can see that AOP is definitely better in reuse, reduction of the code, maintenance of the code, separation of aspects, and readability of code. The reuse in conventional approach means copy, paste and edit. That is not a good approach. From Table 1, we can also see the difference in lines of code (LOC). The AOP has 4 lines of Java code and 25 lines of Java class code. The conventional approach has 16 LOC of Java class, 377 lines of Java code and 102 lines of XML code. LOC where counted by regular expression in search function in eclipse IDE. LOC is counted from the view of developer so the body of the framework is not counted. As we know, this is example that has only 4 attributes. If this number rises for example to eight, the LOC for conventional approach will also rise by similar amount of lines. On the other hand, AOP will increase only by about ten LOC (four lines plus some annotation).

The time to launch the form is also an interesting item, because as you can see in Table 1, AOP is faster than conventional approach. The reason of this is the creation of view by XML which is slower than the view created by a program. This time is calculated when the form is first launched. When user is returning to this activity it is even faster for AOP because it is using a cache with the constant asymptotic complexity of access to data. The average time to launch any form was calculated from the list of ten data that was taken as you can see in Table 2.

Table 2. Table of launching times

Test number	Launch time with AOP (ms)	Launch time with Conventional approach (ms)
1	128	196
2	114	171
3	121	210
4	117	222
5	110	183
6	126	175
7	119	201
8	121	188
9	115	196
10	124	189

6 Comparison of Aspect-Oriented Programming (AOP) for Android Platform and Java EE

The aspect-oriented framework, called Aspect Faces for Java EE [2,3], was created on a similar idea as for Java EE, but they are not implemented in a same way as presented framework for Android. The framework for Java EE [2,3] is called by the tag in a view part as we can see the usage in Listing 1. Mostly it is placed in JSP page or in some xhtml page. In Listing 1, there is an example how to create two forms.

Listing 1. Usage of Aspect Faces in Java EE

```
<!-- Form1 generated via Aspect Faces -->
<af:ui instance="#{bean.entity1}" edit="true"/>
<!-- Form2 generated via Aspect Faces -->
<af:ui instance="#{bean.entity2}" edit="true"/>
```

Instead of this approach, the framework for Android is called in Java activity class that user creates. The reason for this is simple. In Android application structure, the Java classes are mandatory unlike in Java EE where if you just want the UI you do not have to create Java bean or some logic behind. Furthermore, the XML files that represent the UI are optional, because you can create UI by a program. If you create a button to another screen in Android application, you are not connecting the button to call another view, but first, you call the Java activity class and in this class, we can choose how to create UI. In Listing 2, there is the basic example how to create form by our framework in Android.

Listing 2. Usage of framework in Android

```
Presentation p = new
Presentation(this,listOfInstances);
p.buildCache(arraylistOfInstances, afContext);
View v = p.setDataPresentationFromCache();
setContentView(v);
```

Both frameworks are using meta-models to save information about instances that are rendered in forms and also both of them are also working in the run-time. One big difference is in the future potential of the development. A native application has the advantage against web application, that it is compatible with the devices hardware such as motion sensors, environmental sensors, position sensors, and camera. The motion sensors include accelerometers, gravity sensors, gyroscopes and rotational vector sensors. The environmental sensors include barometers, photometers and thermometers. The position sensors include orientation sensors and magnetometers. The web applications are limited in this way. Aspect Faces on Java EE can get a position from GeoIP, but nothing more. This information from sensors can be used by framework and can react to that information.

7 Conclusion and Future Work

This paper results from diploma thesis [9] and contains background of aspect-oriented approach, and describes the design of the new framework for an Android application development. We compare our framework with the conventional approach to Android application development, and to Java EE framework called Aspect Faces. Our framework seems to be fast, clear, easy scalable, readable, reusable, improves the maintenance and it was tested. The results of tests show us, that our framework enables faster development than standard conventional approach to Android application development. It is true that, when we comparing programming approach with the approach using XML, XML has the disadvantage in the speed of launching.

In the future work we will focus on extending the framework context with the hardware devices such as motion sensors, environmental sensors, position sensors, and camera. The application can then easily react to change devices position etc. Also the framework will be tested not only against XML approach, but also against another programmatic approach. Another aim of future work will be to test our framework not only by the launching time, but also by time, when screens are just changing. The expectation is, of course, that our framework will be much faster due to the cache system.

References

1. Czarnecki, K. and Eisenecker, U. W.: Generative programming: methods, tools, and applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (2000)
2. Černý, T., Donahoo, M. J., and Song, E.: Towards effective adaptive user interfaces design. In Proceedings of the 2013 Research in Adaptive and Convergent Systems (RACS '13). ACM, New York, NY, USA, 373-380. DOI=10.1145/2513228.2513278, http://doi.acm.org/10.1145/2513228.2513278 (2013)
3. Černý, T., Čemus, K., Donahoo, M. J., and Song, E.: Aspect-driven, Data-reflective and Context-aware User Interfaces Design. In: Applied Computing Review, Vol. 13, Issue 4, ACM, New York, NY, USA, 53-65. ISSN 559-6915, http://www.sigapp.org/acr/Issues/V13.4/ACR-13-4-2013.pdf (2013)
4. Černý, T. and Song, E.: UML-based enhanced rich form generation. In: Proceedings of the 2011 ACM Symposium on Research in Applied Computation (RACS '11). ACM, New York, NY, USA, 192-199. DOI=10.1145/2103380.2103420, http://doi.acm.org/10.1145/2103380.2103420 (2011)
5. Forman, I. R. and Forman, N.: Java Reflection in Action (In Action series). Manning Publications Co., Greenwich, CT, USA (2004)
6. Kennard, R. and Leaney, J.: Towards general purpose architecture for UI generation. Journal of Systems and Software, 83(10) http: / / metawidget . sourceforge . net / media / downloads / Towards a General Purpose Architecture for UI Generation.pdf (2010) 1896-1906
7. Kennard, R. and Robert, S.: Application of software mining to automatic user interface generation. In SoMeT'08. http: / / metawidget . sourceforge . net / media / downloads / Application of Software Mining to Automatic User Interface Generation.pdf (2008) 244 - 254
8. Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F. and Solberg. A.: Models@run.time to support dynamic adaptation. Computer, 42(10) (Oct. 2009) 44-51
9. Šebek, J.: Aspect-oriented user interface design for Android applications, diploma thesis. Department of Computer Science, CTU FEE, Prague (2014)

Internet resources

10. Android Activity Lifecycle. Android Activity Lifecycle [online]. 12/22/2011, [cit. 2014-04-28]. http: / / www . mikestratton . net / 2011 / 12 / android-activity-lifecycle (2011)
11. Android vs iOS. Android vs iOS [online]. November 11 2013 3:22 PM [cit. 2014-04-28]. http: / / www . ibtimes . com / android-vs-ios-whats-most-popular-mobileoperating-system-your-country-1464892 (2013)
12. Introduction To Android Mobile Operating System. Android Development, Tutorials [online]. August 1, 2011 [cit. 2014-04-29]. http://www.blogsaays.com/tutorial-part1-introduction-android-mobile-operating-system (2011)
13. Jak vypadá Android uvnitř. Android developers [online]. 31. December 2011 [cit. 2014-04-28]. http://www.androidmarket.cz/android/jak-vypada-android-uvnitr-anebco-je-rom-kernel-bootloader-a-dalsi (2011)

A Survey on Music Retrieval Systems Using Microphone Input

Ladislav Maršík[1], Jaroslav Pokorný[1], and Martin Ilčík[2]

[1] Dept. of Software Engineering, Faculty of Mathematics and Physics
Charles University, Malostranské nám. 25, Prague, Czech Republic
{marsik, pokorny}@ksi.mff.cuni.cz
[2] The Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Favoritenstraße 9-11, Vienna, Austria
1040 Vienna, Austria
ilcik@cg.tuwien.ac.at

Abstract. Interactive music retrieval systems using microphone input have become popular, with applications ranging from whistle queries to robust audio search engines capable of retrieving music from a short sample recorded in noisy environment. The availability for mobile devices brought them to millions of users. Underlying methods have promising results in the case that user provides a short recorded sample and seeks additional information about the piece. Now, the focus needs to be switched to areas where we are still unable to satisfy the user needs. Such a scenario can be the choice of a favorite music performance from the set of covers, or recordings of the same musical piece, e.g. in classical music. Various algorithms have been proposed for both basic retrieval and more advanced use cases. In this paper we provide a survey of the state-of-the-art methods for interactive music retrieval systems, from the perspective of specific user requirements.

Keywords: music information retrieval, music recognition, audio search engines, harmonic complexity, audio fingerprinting, cover song identification, whistling query

1 Introduction

Music recognition services have gained significant popularity and user bases in the recent years. Most of it came with the mobile devices, and the ease of using them as an input for various retrieval tasks. That has led to the creation of Shazam application[3] and their today's competitors, including SoundHound[4] or MusicID[5], which are all capable of retrieving music based on a recording made with a smartphone microphone. Offering these tools hand in hand with a convenient portal for listening experience, such as Last.fm[6] or Spotify[7], brings a

[3] http://www.shazam.com
[4] http://www.soundhound.com
[5] http://musicid2.com
[6] http://www.last.fm
[7] http://www.spotify.com

whole new way of entertainment to the users' portfolio. In the years to come, the user experience in these applications can be enhanced with the advances in music information retrieval research.

1.1 Recent Challenges in Music Retrieval

With each music retrieval system, a database of music has to be chosen to propel the search, and if possible, satisfy all the different queries. Even though databases with immense numbers of songs are used, such as the popular Million Song Dataset [1], they still can not satisfy the need to search music in various genres. At the time of writing of this paper, the front-runners in the field as Shazam Entertainment, Ltd., are working on incorporating more Classical or Jazz pieces into their dataset, since at the moment their algorithm is not expected to return results for these genres [22].

Let us now imagine a particular scenario – the user is attending dance classes and wishes his favorite music retrieval application to understand the rhythm of the music, and to output it as a result along with other information. Can the application adapt to this requirement?

Or, if the user wishes to compare different recordings of the same piece in Classical music? Can the resulting set comprise of all such recordings?

There are promising applications of high-level concepts such as music harmony to aid the retrieval tasks. De Haas et al. [2] have shown how traditional music theory can help the problem of extracting the chord progression. Khadkevich and Omologo [9] showed how the chord progression can lead us to an efficient cover identification. Our previous work [13] showed how music harmony can eventually cluster the data by different music periods. These are just some examples of how the new approaches can solve almost any music-related task that the users can assign to the system.

1.2 Outline

In this work we provide a survey of the state-of-the-art methods for music retrieval using microphone input, characterized by the different user requirements. In Section 2 we describe the recent methods for retrieving music from a query created by sample song recording using a smartphone microphone. In Section 3 we show the methods for the complementary inputs such as humming or whistling. We also look at the recent advances in cover song identification, in Section 4. Finally, we form our proposals to improve the recent methods, in Section 5.

2 Audio Fingerprinting

We start our survey on music retrieval systems with the most popular use case – queries made by recording a playback sample from the microphone and looking for an exact match. This task is known in music retrieval as *audio fingerprinting.* Popularized by the Shazam application, it became a competitive field in both academic and commercial research, in the recent years.

2.1 Basic Principle of Operation

Patented in 2002 by Wang and Smith [21], the Shazam algorithm has a massive use not only because of the commercial deployment, but mainly due to its robustness in noisy conditions and its speed. Wang describes the algorithm as a „combinatorially hashed time-frequency constellation analysis“ of the audio [22]. This means reducing the search for a sound sample in the database to a search for a graphical pattern.

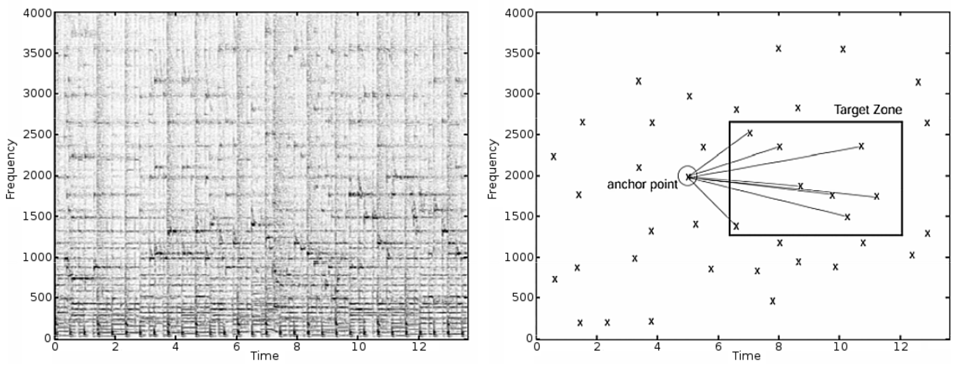


Fig. 1. On the left, time-frequency spectrogram of the audio, on the right, frequency peaks constellation and combinatorial hash generation. Image by Wang and Smith [22].

First, using a discrete-time Fourier transform, the time-frequency spectrogram is created from the sample, as seen on Figure 1 on the left. Points where the frequency is present in the given time are marked darker, and the brightness denotes the intensity of that particular frequency. A point with the intensity considerably higher than any of its neighbors is marked as a peak. Only the peaks stay selected while all the remaining information is discarded, resulting in a *constellation* as depicted on Figure 1 on the right. This technique is also used in a pre-processing step to extract the constellation for each musical piece in the database.

The next step is to search for the given sample constellation in the space of all database constellations using a pattern matching method. Within a single musical piece, it is the same as if we would match a small transparent foil with dots to the constellation surface. However, in order to find all possible matches, a large number of database entries must be searched. In our analogy, the transparent foil has the „width“ of several seconds, whereas the width of the surface constellation is several billion seconds, when summed up all pieces together. Therefore, optimization in form of combinatorial hashing is necessary to scale even to large databases.

As seen on Figure 1 on the right, a number of chosen peaks is associated with an „anchor“ peak by using a combinatorial hash function. The motivation behind using the fingerprints is to reduce the information necessary for search. Given the frequency f_1 and time t_1 of the anchor peak, the frequency f_2 and time t_2 of the peak, and a hash function h, the fingerprint is produced in the form:

$$h(f_1, f_2, t_2 - t_1)|t_1$$

where the operator | is a simple concatenation of strings. The concatenation of t_1 is done in order to simplify the search and help with later processing, since it is the offset from the beginning of the piece. Sorting fingerprints in the database, and comparing them instead of the original peak information results in a vast increase in search speed. To finally find the sample using the fingerprint matching, regression techniques can be used. Even simpler heuristics can be employed, since the problem can be reduced to finding points that form a linear correspondence between the sample and the song points in time.

2.2 Summary of Audio Fingerprinting and Benchmarking

Similar techniques have been used by other authors including Haitsma and Kalker [6] or Yang [23]. The approach that Yang uses is comparison of indexed peak sequences using Euclidean distance, and then returning a sorted list of matches. His work effectively shows how exact match has the highest retrieval accuracy, while using covers as input result in about 20% decrease in accuracy. As mentioned earlier, there are many other search engines besides Shazam application, each using its own fingerprinting algorithm. We forward the reader to a survey by Nanopoulos et al. [14] for an exhaustive list of such services.

To summarize the audio fingeprinting techniques, we need to highlight three points:

1. Search time is short, 5-500 milliseconds per query, according to Wang.
2. Algorithms behave greatly in the noisy environment, due to the fact that the peaks remain the same also in the degraded audio.
3. Although it is not the purpose of this use case, an improved version of the search algorithms could abstract from other characteristics, such as the tonal information (tones shifted up or down without affecting the result, we suggest Schönberg [16] for more information about tonality). However, the algorithms depend on the sample and the match being exactly the same in most of the characteristics, including tempo.

In the end, the algorithms are efficient in the use case they are devoted to, but are not expected to give results other than the exact match of the sample, with respect to the noise degradation.

Interestingly enough, a benchmark dataset and evaluation devoted to audio fingerprinting has only commenced recently[8], although the technology has been around for years. We attribute this to the fact that most of the applications were developed commercially.

[8] http://www.music-ir.org/mirex/wiki/2014:Audio_Fingerprinting

2.3 New Use Cases in Audio Search

There are other innovative fields emerging, when it comes to audio search. Notable are: finding more information about a TV program or advert, or recommendation of similar music for listening. Popularized first by the Mufin internet radio[9] and described by Schonfuss [17], these types of applications may soon become well-known on the application market.

3 Whistling and Humming Queries

Interesting applications arose with the introduction of „whistling“ or „humming“ queries. In this scenario, the user does not have access to the performance recording, but remembers the melody of the music she wants to retrieve. The input is whistling or humming the melody into the smartphone microphone.

3.1 Basic Principle of Operation

In their inspiring work, Shen and Lee [18] have described, how easy it is to translate a whistle input into MIDI format. In MIDI, musical sound commencing and halting are the events being recorded. Therefore, it is easily attainable from human whistle due to its nature. Shen and Lee further describe, that whistling is more suitable for input than humming, with the capture being more noise-resistant. Whistling has a frequency ranging from 700Hz to 2.8kHz, whereas other sounds fall under much smaller frequency span. String matching heuristics are then used for segmented MIDI data, featuring a modification of the popular *grep* Unix command-line tool, capable of searching for regular expressions, with some deviations allowed. Heuristics exist also for extracting melody from the song, and so the underlying database can be created from real recordings instead of MIDI. The whole process is explained in a diagram on Figure 2.

The search for the song in the database can be, as well as in Section 2, improved by forming a fingerprint and creating an index. Unal et al. [20] have formed the fingerprint from the relative pitch movements in the melody extracted from humming, thus increasing the certainty of the algorithm results.

3.2 Benchmarking for Whistling and Humming Queries

Many algorithms are proposed every year for whistling and humming queries. There is a natural need in finding the one that performs the best. The evaluation of the state-of-the-art methods can be found on annual benchmarking challenges such as MIREX[10] (Music Information Retrieval Evaluation Exchange, see Downie at al. [3] for details). The best performing algorithm for 2014 was the one from Hou et al. [8]. The authors have used Hierarchical K-means Tree (HKM) to enhance the speed and dynamic programming to compute the minimum edit

[9] http://www.mufin.com

[10] http://www.music-ir.org/mirex/wiki/MIREX_HOME

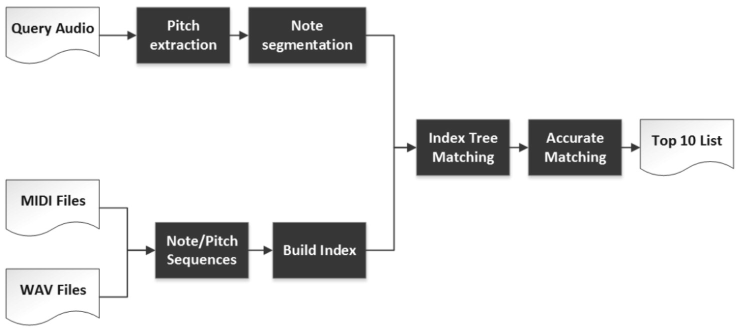


Fig. 2. Diagram of Query by Humming/Singing System, by Hou et al. [8].

distance between the note sequences. Another algorithm that outperformed the competition in the past years, while also being commercially deployed was MusicRadar[11].

Overall, whistling or humming queries are another efficient way of music retrieval, having already a number of popular applications.

4 Cover Song Identification Methods

In the last years, focus has switched to more specific use cases such as efficient search for the cover song or choosing from the set of similar performances. As described earlier, the exact-match result is not satisfying if we, for example, search for the best performance of Tchaikovsky's ballet, from a vast number of performances made. Although not geared on a microphone input (we are not aware of applications for such use case), this section provides an overview of recent cover song identification methods.

4.1 Methods Based on Music Harmony

The task requires a use of high-level concepts. Incorporation of music theory gives us the tool to analyze the music deeper, and find similarities in its structure from a higher perspective. The recent work of Khadkevich and Omologo [9] summarizes the process and shows one way how we can efficiently analyze the music to obtain all covers as the query result. The main idea is segmenting music to chords (musical elements in which several tones are sounding together). The music theory, as described e.g. by Schönberg [16] provides us with the taxonomy of chords, as well as the rules to translate between chords. Taking this approach, Khadkevich and Omologo have extracted „chord progression“ data from a musical piece, and used Levenshtein's edit distance [11] to find similarities between

[11] http://www.doreso.com

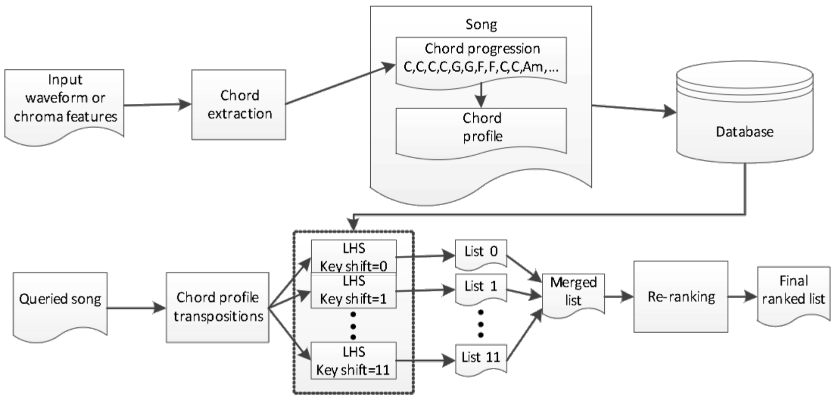


Fig. 3. Diagram of cover song identification by Khadkevich and Omologo [9].

the progressions, as depicted in Figure 3. A method of locality sensitive hashing was used to speed up the process, since the resulting progressions are high dimensional [5].

Another method was previously used by Kim et al. [10] at the University of Southern California. The difference between the approaches lay in the choice of fingerprints. Kim et al. have used a simple covariance matrix to mark down the co-sounding tones in each point of the time. Use of such fingerprints has, as well, improved the overall speed (approximately 40% search speed improvement over conventional systems using cross-correlation of data without the use of fingerprints). In this case, the fingerprints also improved the accuracy of the algorithm, since they are constructed in the way that respect music harmony. They also made the algorithm robust to variations which we need to abstract from, e.g. tempo. This can be attributed to the use of beat synchronization, described by Ellis and Poliner [4].

4.2 Benchmarking for Cover Song Identification

Same as in Section 3, cover song identification is another benchmarking category on annual MIREX challenge, with around 3-5 algorithms submitted every year. The best performing algorithm in the past few years was from The Academia Sinica and the team around Hsin-Ming Wang, that favored the use of extracting melody from song and using melody similarity [19]. Previous algorithm that outperformed the competition was the one made by Simbals[12] team from Bordeaux. The authors used techniques based on local alignment of chroma sequences (see Hanna et al. [7]), and have also developed techniques capable of identifying plagiarism in music (see Robine et al. [15]). On certain datasets, the mentioned

[12] http://simbals.labri.fr

algorithms were able to perform with 80-90% precision of identifying the correct covers.

5 Proposals for Improving Music Retrieval Methods

We see a way of improvement in the methods mentioned earlier. Much more can be accomplished if we use some standardized high-level descriptors. If we conclude that low-level techniques can not give satisfying results, we are left with a number of high-level concepts, which are, according to music experts and theoreticians, able to describe the music in an exhaustive manner. Among these the most commonly used are: *Melody*, *Harmony*, *Tonality*, *Rhythm* and *Tempo*. For some of these elements, it is fairly easy to derive the measures (e.g. Tempo, using the peak analysis similar to the one described in Section 2). For others this can be a difficult task and there are no leads what is the best technique to use. As a consequence, the advantage of using all of these music elements is not implemented yet in recent applications.

In our previous work we have defined the descriptor of Harmonic complexity [13], and described the significance of such descriptors for music similarity. The aim was to characterize music harmony in specific time of its play. We have shown that aggregating these harmony values for the whole piece can improve music recognition [12]. The next step, and possible improvement can be comparing the time series of such descriptors in music. Rather than aggregated values we can compare the whole series in time and obtain more precise results. Heuristics such as dynamic time warping can be used easily for this task. We now analyze the method and its impact on music retrieval. As the future work, experiments will take place to prove the proposed method.

Also, we see the option of combining general methods for cover song identification described in Section 4, with the use case of short recorded audio sample from the microphone. One of the possible ways is abstracting from tonal information and other aspects, as described briefly in Section 2.2. Recent benchmarking challenges for cover song identification are focusing on analyzing the whole songs, rather than a short sample. We believe that a combination of methods described in previous sections can yield interesting results and applications.

6 Summary and Conclusion

We have provided a survey of recent music retrieval methods focusing on: retrieving music based on audio input from recorded music, whistling and humming queries, as well as cover song identification. We described how the algorithms are performing efficiently in their use cases, but we also see ways to improve with new requirements coming from the users.

In the future work we will focus on the use of high-level descriptors and we propose stabilizing these descriptors for music retrieval. We also propose combining the known methods, and focusing not only on the mainstream music, but analyzing other genres, such as Classical, Jazz or Latino music.

Acknowledgments. The study was supported by the Charles University in Prague, project GA UK No. 708314.

Bibliography

1. Bertin-Mahieux, T., Ellis, D.P., Whitman, B., Lamere, P.: The Million Song Dataset. In: *Proceedings of the 12th International Society for Music Information Retrieval Conference.* ISMIR 2011 (2011)
2. De Haas, W.B., Magalhães, J.P., Wiering, F.: Improving Audio Chord Transcription by Exploiting Harmonic and Metric Knowledge. In: *Proceedings of the 13th International Society for Music Information Retrieval Conference.* ISMIR 2012 (2012)
3. Downie, J.S., West, K., Ehmann, A.F., Vincent, E.: The 2005 Music Information retrieval Evaluation Exchange (MIREX 2005): Preliminary Overview. In: *Proceedings of the 6th International Conference on Music Information Retrieval.* ISMIR 2005 (2005)
4. Ellis, D.P.W., Poliner, G.E.: Identifying 'Cover Songs' with Chroma Features and Dynamic Programming Beat Tracking. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing.* ICASSP 2007 (2007)
5. Gionis, A., Indyk, P., Motwani, R.: Similarity Search in High Dimensions via Hashing. In: *Proceedings of the 25th International Conference on Very Large Data Bases.* VLDB '99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)
6. Haitsma, J., Kalker, T.: A Highly Robust Audio Fingerprinting System. In: *Proceedings of the 3rd International Society for Music Information Retrieval Conference.* ISMIR 2002 (2002)
7. Hanna, P., Ferraro, P., Robine, M.: On Optimizing the Editing Algorithms for Evaluating Similarity Between Monophonic Musical Sequences. Journal of New Music Research 36(4) (2007)
8. Hou, Y., Wu, M., Xie, D., Liu, H.: MIREX2014: Query by Humming/Singing System. In: *Music Information Retrieval Evaluation eXchange.* MIREX 2014 (2014)
9. Khadkevich, M., Omologo, M.: Large-Scale Cover Song Identification Using Chord Profiles. In: *Proceedings of the 14th International Society for Music Information Retrieval Conference.* ISMIR 2013 (2013)
10. Kim, S., Unal, E., Narayanan, S.S.: Music Fingerprint Extraction for Classical Music Cover Song Identification. In: *Proceedings of the IEEE International Conference on Multimedia and Expo.* ICME 2008 (2008)
11. Levenshtein, V.I.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Soviet Physics-Doklady 10/8 (1966)
12. Marsik, L., Pokorny, J., Ilcik, M.: Improving Music Classification Using Harmonic Complexity. In: *Proceedings of the 14th conference Information Technologies - Applications and Theory.* ITAT 2014, Institute of Computer Science, AS CR (2014)
13. Marsik, L., Pokorny, J., Ilcik, M.: Towards a Harmonic Complexity of Musical Pieces. In: *Proceedings of the 14th Annual International Workshop on Databases, Texts, Specifications and Objects (DATESO 2014). CEUR Workshop Proceedings*, vol. 1139. CEUR-WS.org (2014)
14. Nanopoulos, A., Rafailidis, D., Ruxanda, M.M., Manolopoulos, Y.: Music Search Engines: Specifications and Challenges. Information Processing and Management: an International Journal 45(3) (2009)

15. Robine, M., Hanna, P., Ferraro, P., Allali, J.: Adaptation of String Matching Algorithms for Identification of Near-Duplicate Music Documents. In: *Proceedings of the International SIGIR Workshop on Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection*. SIGIR-PAN 2007 (2007)
16. Schönberg, A.: Theory of Harmony. University of California Press, Los Angeles (1922)
17. Schönfuss, D.: Content-Based Music Discovery. In: *Exploring Music Contents*, *Lecture Notes in Computer Science*, vol. 6684. Springer (2011)
18. Shen, H.C., Lee, C.: Whistle for Music: Using Melody Transcription and Approximate String Matching for Content-Based Query over a MIDI Database. Multimedia Tools and Applications 35(3) (2007)
19. Tsai, W.H., Yu, H.M., Wang, H.M.: Using the Similarity of Main Melodies to Identify Cover Versions of Popular Songs for Music Document Retrieval. Journal of Information Science and Engineering 24(6) (2008)
20. Unal, E., Chew, E., Georgiou, P., Narayanan, S.S.: Challenging Uncertainty in Query by Humming Systems: A Fingerprinting Approach. IEEE Transactions on Audio, Speech, and Language Processing 16(2) (2008)
21. Wang, A.L., Smith, J.O.: Method for Search in an Audio Database. Patent (February 2002), WO 02/011123A2
22. Wang, A.L.: An Industrial-Strength Audio Search Algorithm. In: *Proceedings of the 4th International Society for Music Information Retrieval Conference*. ISMIR 2003 (2003)
23. Yang, C.: Macs: Music Audio Characteristic Sequence Indexing for Similarity Retrieval. In: *IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics*. WASPAA 2001 (2001)

Author Index