

Data Mining: A Brief Overview and Recent IPSI Research

Zaharije Radivojevic, Milos Cvetanovic, Veljko Milutinovic

School of Electrical Engineering, University of Belgrade,
Belgrade, Serbia and Montenegro,

{zaki, cmilos, vm}@etf.bg.ac.yu

Joerg Sievert

Stuttgart, Germany

Joerg.Sievert@3i.com

Abstract—Information extraction from large amount of data is the problem addressed in this paper. Efficient usage of data mining models and algorithms will be presented using the recent IPSI research as a case study. A novel algorithm based on the K-nearest neighbor model has been developed. The algorithm is general enough to be considered as a common solution for a group of similar problems. On the other side, it is a good example on how data mining techniques could be efficiently used for reverse engineering problem types.

Index Terms—Data mining, Data Gathering, Data Gathering Tools, Dynamic Data Analysis, GeForce GPU, K-nearest Neighbor model, Operating Systems, Reverse Engineering, Symbolic Debugging.

I. BRIEF OVERVIEW

Information extraction from large amount of data is a multidimensional problem addressed in various research fields. The need to uncover the hidden knowledge buried in the raw data faces us with a huge NP complete problem domain. Therefore, existence of partial solutions that cover specific problem areas is expected. Instead of that, a completely new methodology has been developed. The newest solution of the problem is data mining. It can be defined as automated extraction of predictive information from different data sources. It is a powerful technology with great potential to help users focus on the most important information. Data mining can answer questions that were too time consuming to resolve in the past. It can also help us predict future trends and behaviors, allowing us to make proactive, knowledge-driven decisions. On the other side, it does not eliminate the need to know your problem as well as to understand your data or analytical methods [15]. Just like any other novelty term, “data mining” is often misused and applied in wrong contexts. For example, it is often frequently identified with OLAP

(On-Line Analytical Processing). OLAP is used for verification of the existing hypothesis; Data mining tries to generate such hypothesis by uncovering the hidden patterns. It is essentially an inductive process. It means that OLAP and data mining should be used together, in order to complement each other. OLAP can be used to verify the results of the data mining process. In other words, data mining should be comprehended as an assistant that can help you classify your data, predict later behavior, extract association rules or detect sequences.

II. DATA MINING PROBLEM TYPES

Before we can use data mining models and algorithms we have to find the most suitable strategy. In order to do that we have to detect the problem type. Usually, data mining project involves a combination of different problem types, which together solve the problem [1].

At the lower end of the scale of the data mining problems is Data description and summarization. It aims at the concise description of characteristics of the data, typically in elementary and aggregated form. This gives us an overview of the structure of the data.

The next data mining problem type is problem of Segmentation. It aims at the separation of data into interesting and meaningful subgroups or classes. All member of a subgroup contain common characteristics. Misunderstanding of term segmentation is caused by it's relation with Classification, which is another data mining problem type. Classification assumes that there is a set of objects that belong to different classes, where some attributes or features characterize each class. The objective is to build classification model which assign correct class label to previously unseen and unlabeled objects (so called predictive modeling). Classification

and Segmentation may introduce new type of data mining problems; it is a Concept description problem. It aims at an understandable description of concepts or classes. The purpose is not to develop complete model with high prediction accuracy, but to gain insights.

Another important problem type that occurs in a wide range of application is Prediction. The aim of Prediction is to find the numerical value of the target attribute for unseen objects. In close connection to the Prediction is another problem type, so called Dependency analysis. It consists of finding a model that describes significant dependencies and associations between data items or events.

III. DATA MINING MODELS

A systematic approach is essential to successful data mining. Many models were designed in order to conduct a sequence of steps that will lead to adequate results.

One of the possible approaches is modeling based upon Neural networks [7]. Many years of experience in this area lead to efficient modeling of large and complex problems. This model can be easily applied for characterization of processed data with single numeric value. Therefore it is usually used for prediction problem types. On the other side this model has numerous drawbacks. It often requires huge amount of data to be preprocessed and beside that it mostly depend on time spent for training of neurons.

Another popular model is based upon Decision trees. It is a way of representing a series of rules that iteratively splits data into discrete groups maximizing distance between them at each split. This model is very flexible when we take growth and stopping rules into consideration.

Similar modeling approach is called Rules induction. It creates a set of rules that are unlikely to form a tree. It differs from the previous model because here rules may not cover all possible situations and rules may sometimes conflict in a prediction.

Beside many other existing models we are going to name one more. It is a K-nearest neighbor and Memory based reasoning model. It is based on usage of knowledge of previously solved similar problems for solving the new problem. When a new data comes it will be classified as a class member of a nearest existing group. It is very important to find a suitable measure for distance between groups.

IV. RECENT IPSI RESEARCH

Methodology and algorithms described in this paper arose as a result from the projects that were conducted on the Electrical Engineering Faculty of Belgrade University in cooperation with IPSI Belgrade. The projects

goal was developing of GeForce graphic card drivers for custom designed operating system, in the following text will be referenced as the GF project. The project started in a spring of 2002, when GeForce graphic chip was inviolable leader on the market. At that moment there was no documentation or source code available, though, retargeting of the existing drivers was not an option, especially if we consider the fact that the particular operating system is based on a completely different paradigm in the contrary to the traditional operating systems. It is actually distributed operating system, which relies on transactions as atomic operations. The transaction management follows the optimistic assumptions with no locking introduced at all, which results in very fast and simple algorithms in the kernel, but it implies additional complexity for the driver programmers. This is actually a case of dynamic reverse engineering research project. In cases like this static analysis is useless because there was no source code to be analyzed. Decompiling of existing drivers would not bring anything useful. The reason for this is dynamic driver loading and run time dependency, so it is impossible to reduce dynamic processes to the static foundations.

Dynamic analysis used during GF project involved following steps: gathering of raw data, data processing, and visualization of extracted information. In the following sections universal methodology for the row data gathering will be explained in detail, and novel algorithm for information resolving will be presented, while the last step, visualization of the results, will not be further considered [2], [13]. The dynamic data gathering is a problem for it self, but once it is solved, it produces mass of unstructured data [12]. This is where data mining algorithm shows it's full strength.

V. DYNAMIC DATA GATHERING METHODOLOGY

Understanding of the processor architecture is necessary precondition for comprehension of the presented methodology. Especially important parts of the architecture are those related to debugging and performance measurement capabilities.

Before the project started some decisions had to be made. Those decisions have influence on the implementation and not on the methodology itself. The first decision was selecting operating system. Windows platform was chosen. Many would be surprised with this decision, because well known fact is the Windows source code is not publicly available, on the contrary to the Linux open source community. The answer is that it is widespread over personal computers in not only medium sized but also in big enterprises. The second reason would be demystification of the Windows secrets

[11]. The last, but not least important reason is personal satisfaction of the team members.

This technique is based on a noninvasive dynamic data gathering as the safest way to show the precise characteristics of the analyzed system. This is true not only for applicative software but also for analyzing of the every possible working mode of the operating system. Those advantages were used during GF project [4], [14].

For the purpose of the GF project it was necessary to gather data about all accesses to the graphic card resources that are done in the run-time on behalf of the operating system. Having in mind that all used resources are memory mapped, it is easy to conclude that presented methodology could be generally used in cases when it is needed to record access to any memory location. The capabilities of the Intel processors (usage of DR0-DR3 registers) were not satisfying because access to numerous memory locations had to be evidenced [8].

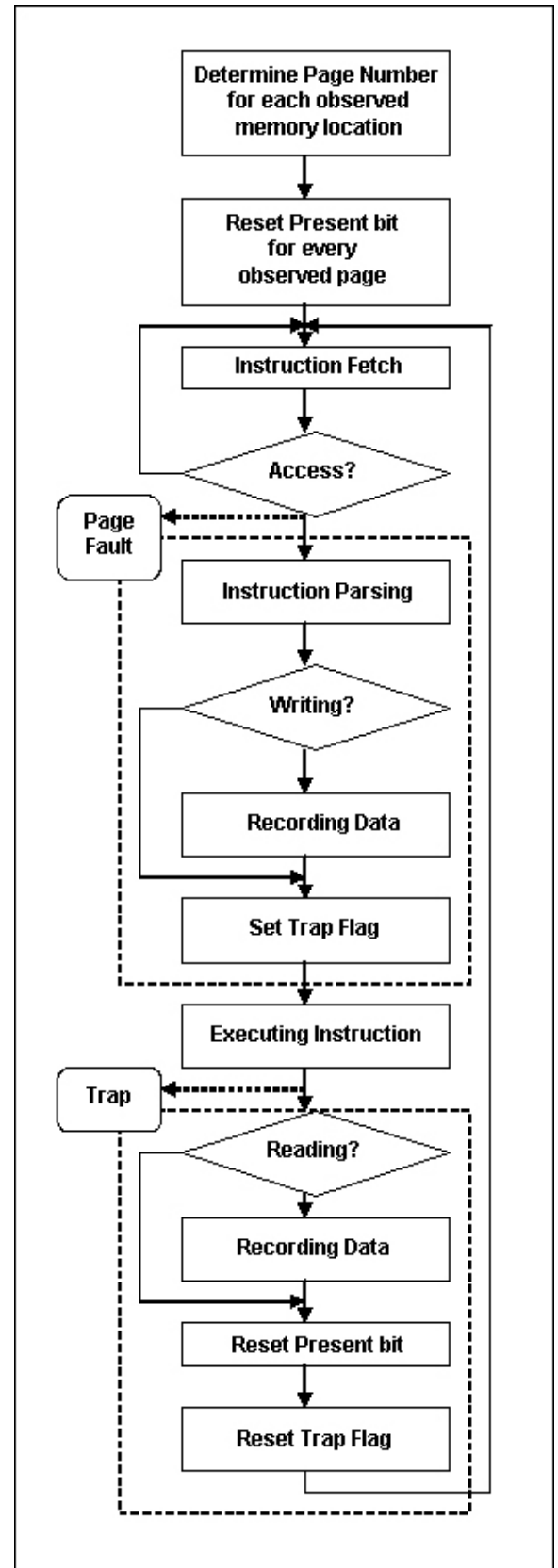
First step of the algorithm is to determine page number (segmented paged virtual memory) that corresponds to each

of the memory mapped location that we wish to observe [9], [10].

The second step is to reset the *present bit* of each page that was selected in the previous step (present bit that is placed in the *page descriptor*). In this particular project we had a situation to deal with the non cacheable pages (memory that is physically located on the graphic card is memory mapped). If the currently executing instruction tries to access observed memory location, *page fault interrupt* will be raised (this is because present bit in the page descriptor was reset). Interrupt handler will do the instruction parsing and record all important data (instruction address, instruction code, access type write/read, data length, data, and data address). In the case of read access we have to wait till end of the instruction to be able to record the data that had been read from the location. In order to obtain all mentioned data we have set *trap flag* in the *program status word* (PSW). This will result in that after the instruction finishes the *trap interrupt* will be raised. Trap handler will read the data (in the case of read access) and reset present bit (in both read and write access) that corresponds to the page that has just been accessed. Before the trap handler completes, trap flag has to be reset. It is obvious that the described algorithm requires that interrupt handlers to be hooked and new interrupt routines to be put on the beginning of interrupt handler lists [3].

Operations that are referred in the algorithm (interrupt handler hooking, present bit resetting, etc.) could be executed only inside of the program with high level

Fig. 1. Data gathering algorithm. Dashed boxes represent steps conducted inside of interrupt handlers (Page Fault and Trap).



of privileges (kernel level). From that reason during the GF project the SoftICE symbolic debugger had been used. The SoftICE is working on the kernel level and it has one very important feature, the possibility to be extended. This algorithm was implemented in the extension had been written in the NASM assembly language.

VI. DATA PROCESSING PHASE – DATA MINING

Data gathering methodology described in the previous section had been used for collecting raw data produced in the series of tests conducted on g different GF2 graphic cards. All graphic cards had been tested on the same hardware configuration with generic driver installed. Mappings of memory and I/O address space could vary between different graphic cards, but relative order of registers inside of the particular memory apertures is constant.

Data processing described here involves all type of problems described in one of the introductory section, and therefore algorithm presented here could be used as a common solution for a group of similar problems.

There are differences between some parts of the gathered data for different graphic cards, and those are implicated only by individual card or operation characteristics. Those parts are referenced as noise. So, data accesses, that include address, order, and data that are card specific for some operations, are also to be treated as noise. On the other side, there are parts of the gathered data that are common for all cards, and they are used for extraction of important memory locations and resources.

A. Mathematical model

Mathematical model that had been used for information resolving from gathered data maps problem space into 5D vector space [6]. More precisely, we have to deal with g independent 4D vector spaces, designated by $V(t, a, rw, d)$. In the $V(t, a, rw, d)$, t stands for time axis, a stands for memory addressing axis, rw for memory access type axis (read/write), and d for data axis.

Construction of a subspace u of the vector space V is defined with the following formula:

$$\forall u(t_u, a_u, rw_u, d_u), V(t, a, rw, d)(u \subseteq V \Leftrightarrow t_u \subseteq t \wedge a_u \subseteq a)$$

The purpose of the formula is to compact the 4D problem space by bounding on the two coordinates (time axis and memory addressing axis). The remaining two coordinates

(rw axis and data axis) serve for similarity function calculation which will be introduced in the following section.

Purpose of the mathematical model is to describe formal method for extracting set of important locations that we denote as set I . It is obvious that set I , can only be used for discovering the important locations and not for giving a meaning to each one of them. We also define I^o as a set of locations important for an operation $o \in O$ where O denotes set of all operations available on the GF chip. We could say that the following formula stands:

$$I^o \subseteq I \wedge \bigcup_{o \in O} I^o = I$$

B. Similarity function

Heart of the presented mathematical model is similarity function $S(u_1, u_2, C)$ where u_1, u_2 represent 4D vector subspaces, $u_1, u_2 \in V(t, a, rw, d)$, and C is restriction criterion.

Purpose of the similarity function is to compare two subspaces and to establish level of conformance.

Similarity function is between $[0, 1]$ and that is the first characteristic of the S function. It could be defined as

$$\forall(u_1, u_2, C)(0 \leq S(u_1, u_2, C) \leq 1)$$

The second characteristic of the similarity function is:

$$\forall(u_1, u_2, C)(u_1 = u_2 \Rightarrow S(u_1, u_2, C) = 1)$$

$$\forall(u_1, u_2, C)(u_1 = \bar{u}_2 \Rightarrow S(u_1, u_2, C) = 0)$$

The third characteristic is:

$$S(u_1, u_2 + u_3, C) = S(u_1, u_2, C) + S(u_1, u_3, C)$$

$$u_2 \cap u_3 = \{\}$$

The fourth characteristic is:

$$S(u_1, u_2, C) = S(u_2, u_1, C)$$

Similarity function defined in this manner is equivalent to the definition of probability function. That implies opportunity to use results gained by theory of information, like information quantity and entropy of the system V .

What we are actually trying to explain is that similarity function $S(u_1, u_2, C)$ aims to maximize pairing of two subspaces u_1 and u_2 . Maximal pairing means finding maximal set of coinciding vectors from u_1 and u_2 . Coinciding of given vectors depends on the restriction criterion C . In the other words, we say that each vector from subspace u_1 coincides to the particular vector from subspace u_2 if it is inside of C -environment of that particular vector. Formal definition tells that vector $x(t_x, a_x, rw_x, d_x)$ is in C -environment of the vector $y(t_y, a_y, rw_y, d_y)$ if and only if:

$$rw_x = rw_y \text{ and } |d_x - d_y| < C_d \text{ and } |a_x - a_y| < C_a$$

Previous definition represents a suitable distance measure between vectors [5]. Distance measure for coordinate d is assumed to be the Hemming's distance, while distance measuring for coordinates a and t are measured relatively from the beginning of the vector subspace. Note that restriction criterion C is vector $C(C_t, C_a, C_{rw}, C_d)$, and it enables different levels of filtering.

C. Algorithm

The goal of the algorithm is to discover set I and sets I^o for the given set O , and to minimize noise (noise minimization is equivalent to information discovery). This could be accomplished in three phases. In all phases we iteratively use similarity function. Each iteration is conducted with different level of filtering (level of filtering is expressed through restriction criterion C). We emphasize that outputs of each phase are used as inputs for the following phases. Results of the first phase are sets I_j , that represent sets of important locations (memory locations) obtained for different restriction criterion C used with similarity function. The second phase of iterations inducts I_j^o for $\forall o \in O$, where j is interpreted according to the results of the previous phase. The purpose of the third phase is to eliminate noise caused by individual card or operation characteristics.

This algorithm is based upon K-nearest neighbor model, with such modification that we have to deal with a lack of information about previous similar problems. This deficiency is surmounted by numerous iterations which purpose is to build the knowledge base. The knowledge is formed upon maximization of similarity function applied to the 4D vector subspace of each test card. This process will be conducted on each possible pair of 4D vector subspaces, which results in algorithm complexity of:

$$O\left(\binom{g}{2} \cdot \|V\|^2\right)$$

Where $\|V\|$ stands for a norm of each 4D subspace, actually it is a number of possible subspaces.

Now that we have explained the essence of the mathematical model we can step into further consideration of each algorithm phase. The first phase of iterations results in sets of approximations of target set I . Those sets are gained under different criterion C , and are denoted as I_j sets. This could be expressed with:

$$I_j = \bigcap_{k \in g} I_{jk}$$

where I_{jk} are sets of important locations for a particular graphic card k . Each one of the I_{jk} sets is obtained as a important locations set for the particular vector space V_k under criterion C_j . The V_k are instances

of 4D vector space V formed upon gathered data for graphic card k .

The purpose of the second phase is to narrow the results of the previous phase by producing I_j^o sets. It is accomplished by looking at only those important locations that are involved in the execution of the particular operation o . Operation set O is predefined and unique among all test cards. For each operation o from set O we know point in time t_0 before which operation did not start. This will result in additional compaction for each vector space V_k . So, initial vector space $V_k(t, a, rw, d)$ is projected to vector subspace denoted as: $V_k'(t', a', rw', d')$ where $t' > t_0, a' \in I_j$.

The final, third phase, tries to resolve noise caused by different card or operation characteristics. In this phase entire problem space (vector space) is narrowed to the level of bit patterns inside of a written or read data from the I_j^o sets. Comparison of the bit patterns results in identification of common characteristics for all test graphic cards.

On the other side, bit patterns and sets of important locations that are constant among all test cards (similarity function $S = 1$) in this phase do not contribute to the overall quantity of information. The way we defined similarity function S , is equivalent to the definition of probability function and therefore from the theory of information we can conclude that system is sustainable with entropy equals to 0 ($H = 0, I = 0$). This means that we are not able to extract the meaning of particular bit patterns, but still they can not be omitted, and they have to be used as they are. These kinds of results are presented in the form of a fix functionality tables.

VII. PROJECT VERIFICATION

In order to verify the given methodology, described in the previous section, part of the gathered data will be presented and explained in details. The process of information extraction and visualization will not be presented since it would require complete understanding of the context in which the data were collected.

The following listing (Figure 2) contains the data gathered during graphic card initialization and graphic mode setting for resolution of 1024*768 in 32 bit color palette with 60Hz of vertical refresh rate [3]. Address range from D8000000(hex) to D8FFFFFF(hex) refer to memory mapped graphic card input/output address space, while range from D0000000(hex) to D7FFFFFF(hex) refer to frame buffer range. The observed graphic card is based on GeForce MX400 chip with 64 MB RAM on board.

The format of the shown listing is as follows. The first column is an address of the instruction that access to observed memory range. The second column tells us the access type (read or write). The next column

Fig. 2. Verification results. This figure presents data gathered during GF project with presented methodology.

| Instr. Address | Access Type | Length | Data | Data Address |
|----------------|-------------|--------|------------|--------------|
| 0xbf96bce | Write | DWORD | 0x0000000 | 0xd8000140 |
| 0x80455b74 | Read | DWORD | 0x11000b2 | 0xd8000000 |
| 0x80455b5c | Read | BYTE | 0x55 | 0xd8700000 |
| 0x80455b5c | Read | BYTE | 0xff | 0xd870c400 |
| 0x80455b74 | Read | DWORD | 0x011000b2 | 0xd8000000 |
| 0xbf96cac2 | Read | BYTE | 0xff | 0xd8300000 |
| 0xbf96ca696 | Read | DWORD | 0x011010de | 0xd8001800 |
| 0xbf96cad5f | Read | DWORD | 0x08c10110 | 0xd8100200 |
| 0x80455c00 | Write | DWORD | 0x00000000 | 0xd8001830 |
| 0x80455bd8 | Write | BYTE | 0x1f | 0xd86013d4 |
| 0x80455b74 | Read | DWORD | 0x02005748 | 0xd8001084 |
| 0x80455b74 | Read | DWORD | 0x011010de | 0xd8001800 |
| 0x80455b74 | Read | DWORD | 0x801d4547 | 0xd8101000 |
| 0x80455b74 | Read | DWORD | 0x02b00007 | 0xd8001804 |
| 0x80455b74 | Read | DWORD | 0x03110111 | 0xd8000200 |
| 0x80455c00 | Write | DWORD | 0x1f000102 | 0xd800184c |
| 0x80455b74 | Read | DWORD | 0x00004603 | 0xd8680504 |
| 0x80455b74 | Read | DWORD | 0x000009ff | 0xd8100228 |
| 0x80455b74 | Read | DWORD | 0x0003be0c | 0xd8680508 |
| 0x80455b74 | Read | DWORD | 0x11000b2 | 0xd8000000 |
| 0x80455b5c | Read | BYTE | 0xeb | 0xd8700003 |
| 0x80455b74 | Read | DWORD | 0x00000000 | 0xd8009200 |
| 0x80455b74 | Read | DWORD | 0x04000000 | 0xd810020c |
| 0x80455c00 | Write | DWORD | 0x03020100 | 0xd8002210 |
| 0x80455c00 | Write | DWORD | 0x00000000 | 0xd8710000 |
| 0x80455c00 | Write | DWORD | 0x00000000 | 0xd8711000 |
| 0x80455c00 | Write | DWORD | 0x00000000 | 0xd8712000 |
| 0x80455c00 | Write | DWORD | 0x00000000 | 0xd8713000 |
| 0x80455c00 | Write | DWORD | 0x00000000 | 0xd8714000 |
| 0x80455c00 | Write | DWORD | 0x00000000 | 0xd8715000 |
| 0x80455c00 | Write | DWORD | 0x00000000 | 0xd8716000 |
| 0x80455c00 | Write | DWORD | 0x00000000 | 0xd8717000 |

presents access data length (byte/word/dword), and it is followed with the column that holds the actual data. The last column presents the address of the memory location (the accessed location).

Explanation and information extraction is not the topic of this paper, but for the illustration purposes will take a look at the instruction on 80455B74(hex). This instruction makes access to frame buffer range mapped with offset 00100000(hex) from the start of the memory mapped input/output graphic card address space (which start on the D8000000(hex) in this particular case). The address of the accessed register is 0000020C(hex) from the start of the frame buffer. The name of this register is NV_PFB_BOOT_0_RAM_AMOUNT, and it holds the amount of onboard RAM memory. The information

about available memory is coded with bit range 27-20. Meaning of those 8 bits is interpreted as follows: 02(hex) (2MB), 04(hex) (4MB), 08(hex) (8MB), 10(hex) (16MB), 20(hex) (32MB), 40(hex) (64MB), and 80(hex) (128MB). It is obvious that in this particular case we had a graphic card with 64 MB RAM on-board, which is verified with the value that was read (04000000(hex)).

From the presented example it is not difficult to understand the overall complexity of this project. The project in which it was necessary to analyze millions of bits and to find the meaning for each one of them, as it was defined by one of our project members.

VIII. CONCLUSION

Usage of data mining techniques for purpose of dynamic reverse engineering is still to be explored, but methodology and algorithm defined here is a platform independent and therefore could be used as a base for all future researches in this domain. Generality of the presented algorithms suggests that they could be used with minimal adjustments for solving a wide range of different problems. Further development of the algorithms in this domain would significantly decrease software engineering expenses and consequently increase total quality of software products.

REFERENCES

- [1] Bruha, I., 'Data Mining, KDD and Knowledge Integration: Methodology and A case Study', SSGRR 2000
- [2] Chikofsky I, Cross JH, "Reverse Engineerig and Design Recovery : A Taxonomy", IEEE Software, vol.7, no.1, Jan. 1990
- [3] Cvetanovic M, Radivojevic Z, "Dinamicki reverzni inzenjering: Metodologija za prikupljanje podataka", YUInfo, Kopaonik, Serbia and Montenegro, March 2003
- [4] Fayyad, U., Shapiro, P., Smyth, P., Uthurusamy, R., *Advances in Knowledge Discovery and Data Mining*, MIT Press, 1996
- [5] Glumour, C., Maddigan, D., Pregibon, D., Smyth, P., "Statistical Themes nad Lessons for Data Mining", *Data Mining And Knowledge Discovery* 1, 11-28, 1997
- [6] Hecht-Nilsen, R., *Neurocomputing*, Addison-Wesley, 1990
- [7] Erdos K., Sneed M., "Partial comprehension of complex programs (enough to perform maintenance)", Proc. 6th International Workshop on Program comprehension, 1998
- [8] Intel Corporation, *Intel Architecture Software Developer's Manual. Volume 1: Basic Architecture*, Intel Corporation, Santa Clara, CA, 1999
- [9] Intel Corporation, *Intel Architecture Software Developer's Manual. Volume 2: Instruction Set Reference*, Intel Corporation, Santa Clara, CA, 1999
- [10] Intel Corporation, *Intel Architecture Software Developer's Manual. Volume 3: System Programming*, Intel Corporation, Santa Clara, CA, 1999
- [11] Microsoft Corporation, *Microsoft Windows 2000 Driver Development Kit*, Microsoft Corporation, Redmond, WA, 2000
- [12] Pyle, D., *Data Preparation for Data Mining*, Morgan Kaufman, 1999
- [13] Tilley S.R., Paul S., and Smith D.B., "Towards a Framework for Program Comprehension", The 4th Workshop on Program comprehension, 1996
- [14] Woods S., Yang Q., "The program understanding problem: analysis and a heuristic approach", Proc. 18th International Conference on Software Engineering, 1996
- [15] Jovanovic, N., Milenkovic, V., and Milutinovic, V., "Issues in Data Mining Infrastructure," on-line tutorial at <http://galeb.etf.bg.ac.yu/~vm/tutorial/tutorial.html>, December, 2003