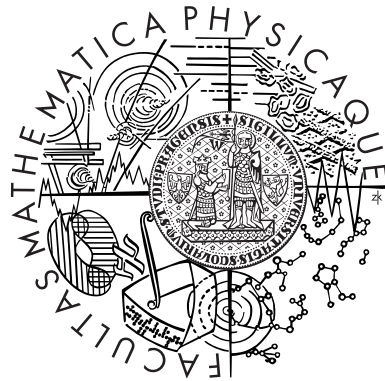


Informatická sekce
Matematicko–fyzikální fakulta
Univerzita Karlova v Praze



MIS 2010

16.–23. ledna 2010, Josefův Důl

Sborník semináře

matfyzpress

PRAHA 2010

Všechna práva vyhrazena. Tato publikace ani žádná její část nesmí být reprodukována nebo šířena v žádné formě, elektronické nebo mechanické, včetně fotokopí, bez písemného souhlasu vydavatele.

© (Eds.) David Obdržálek, Martin Plátek 2010

© MATFYZPRESS, vydavatelství Matematicko-fyzikální fakulty
Univerzity Karlovy v Praze, 2010

ISBN 978-80-7378-148-4

ÚVOD

Tento sborník je sestaven z příspěvků 27. ročníku Malého Informatického Semináře (**MIS'10**), který proběhl 16. - 23. ledna 2010 v Josefově Dole. Podobně jako v předchozích letech se tohoto semináře zúčastnili členové kateder informatické sekce MFF UK, studenti, zvaní hosté a další aktivní účastníci. Stablními účastníky se také stali mladší studenti doktorského studia.

Malý informatický Seminář je organizován již řadu let, jeho sborník letos vychází po osmé. Do sborníku bylo v tomto roce vybráno 6 příspěvků. Všechny příspěvky prošly recenzí.

Příspěvky tematicky spadají do oblasti výzkumu vedeného jednotlivými katedrami informatické sekce MFF UK. Někteří autoři také využili dodatečné možnosti rozšíření svého textu oproti původní verzi. Editoři proto doufají, že tento sborník poslouží i jako doplněk učebních textů pro příslušné obory.

Programový i organizační výbor semináře pracoval ve složení: Martin Plátek jako předseda, David Obdržálek jako člen; oba jsou pak editory sborníku.

Příspěvky recenzovali především pracovníci informatické sekce MFF UK a Ústavu Informatiky AV ČR.

Děkujeme všem, kteří se podíleli na zdu Malého Informatického Semináře v roce 2010, a to jak těm, kteří jej pomohli zorganizovat, tak i všem účastníkům. Tradiční poděkování přísluší i pánům Lumíru Kymrovi a Václavu Šilháčkovi za trvalé pochopení při zajišťování ubytování účastníků našeho semináře.

Praha 2010

M. Plátek
D. Obdržálek

OBSAH

Úvod	1
Obsah	2
Tomáš Haničinec, Martin Holeňa: Genome encoding in genetic and EDA algorithms for rule extraction	3
Tomáš Holan: Evoluce komunikace umělých bytostí	17
Petr Homola, Jernej Vičič, Vladislav Kuboň: On the Role of Statistical Methods in Machine Translation Between Related Languages	25
Lucie Kárná: Metodika kvantitativního hodnocení detekčních kódů	33
Jana Katreniaková, Jiří Dokulil: Vizualizácia modelu v systéme Bobox	45
Martin Kruliš, Jakub Yaghob: Obecné výpočty na grafických kartách – použitelnost, vlastnosti, praktické zkušenosti	55

Genome encoding in genetic and EDA algorithms for rule extraction

Tomáš Haničinec

Martin Holeňa

KTIML MFF UK

ÚI AV ČR

Malostranské nám. 25, Prague

Pod Vodárenskou věží 2, Prague

tomas.hanicinec@centrum.cz

martin@cs.cas.cz

Abstract. Rule extraction forms one of the most important areas of data mining. Foremost advantages of this approach are namely good comprehensibility of the rules for humans and the consequent possibility to closely analyze and tune the relations and dependencies found by one of the commonly used rule extraction algorithms. This article explains two groups of such rule extraction algorithms - genetic algorithms and EDA algorithms. Each group is first introduced in its basic form. Then, the ways and specifics of its application on the domain of rule extraction are shown in detail along with the techniques used to boost the algorithms performance.

Keywords: rule extraction, genetic algorithm, EDA algorithm, genome encoding

1 Introduction to rule extraction

Rule extraction is one of mainstream areas of data mining. Its' task is to find significant relationships and dependencies in an input data set in the form of human-readable rules.

An **input data set** is a set of n *data instances*. Each instance is a vector of m attribute values and a target attribute value. *Attributes* can be of different types. For the purposes of this article we differentiate attributes into *discrete attributes* (like gender, nationality or blood_type) and *continuous attributes* (like weight, dimension or time). *Target attribute* is the attribute we are trying to predict.

An example of **a rule** is given

IF ($age > 60$) **&** ($smoker = yes$) **&** ($weight \geq 100$) **THAN** ($cancer_risk = high$)

A rule is composed of a conjunction of individual *conditions* (like $age > 60$). The conjunction of conditions is called the *antecedent*. The assertion about the target attribute (like $cancer_risk = high$) is called the *consequent*. The semantics of a rule is simple - if a data instance satisfy the antecedent, than it should have the target attribute according to the consequent.

usually defined as a fixed number of generations. Other options are possible - for example reaching sufficiently good solution or reaching the point where the changes in the population between consequent generations are sufficiently small. **Fitness evaluation** is often the only problem dependent part of the algorithm. Each individual is assigned a value corresponding with its quality as a solution to a problem we want to solve. More about genetic algorithms and the underlying theory can be found for example in [9].

3 Genetic algorithms for rule extraction

Genetic algorithms can be successfully used to extract rules from data. The domain of rule extraction has however some specific features that make the standard version of genetic algorithm slow and inefficient. It is therefore necessary to introduce some domain specific techniques to improve the algorithms' performance. Some of these techniques are introduced in the following sections. For further reading, there is an excellent monograph [5] covering most of the presented techniques in more detail.

3.1 Individual representation

The first question we have to answer is what will form the individual itself. The most obvious solution is to encode one rule into an individual (so called Michigan approach). Michigan approach is easy, straightforward and fast. However, as an output of the algorithm we would usually like to get a set of rules that reasonably covers the input data set. With one rule per individual we can either return several top individuals (rules) or to run the algorithm repeatedly and get the best individual/rule per each run (Both cases will very likely produce almost identical rules. To prevent that, the technique of sequential covering is used. Whenever we select a rule, the input data set is stripped of the instances covered by the rule and the next rule is selected with respect to this reduced input data set). In both cases we are only guaranteed to obtain the set of best rules. Due to the possible dependencies and duplicities among the rules it often doesn't mean the best rule set. This drawback can be solved by taking the entire rule set as an individual (so called Pittsburgh approach). This way the algorithm only has to be run once and the possible dependencies among the rules are solved within the algorithm itself. Naturally the price is increased complexity and runtime of genetic operators and fitness evaluation. In the following text we decided to use Michigan approach (one rule per individual encoding), although most of the conclusions are valid for Pittsburgh approach, too. More detailed analysis of advantages and disadvantages of both approaches in genetic algorithms can be found for example in [8].

So how to encode a rule into a genetic algorithm individual? We start with encoding a single condition. Let us consider a condition on the attribute **A**. If the attribute **A** is discrete (say *gender* or *country*), we can encode the condition into simple binary string as shown on Figure 2 (a). Such an encoding is simple, the standard genetic operators can be used without modification and we can naturally encode logical OR by simply adding more ones to the string. If the attribute is continuous (like *weight* or

1. WHILE(NOT population is initialized)
 - (a) $seed = \text{getRandomInstance}(input_data_set)$
 - (b) $rule = \text{createExactRule}(seed)$
 - (c) $rule = \text{generalize}(rule)$
 - (d) $\text{assignToPopulation}(rule, population)$

First, a data instance (seed) is randomly selected. Then the rule exactly matching the selected data instance is generated. Finally, this rule is adequately generalized (more on how to generalize a rule in the section 3.5) and assigned to the initial population. This process is repeated until the whole initial population is created.

3.3 Fitness function

Fitness function in genetic algorithms should provide an estimate of an individuals quality. In our case, it should reflect the quality of a rule. There are several criteria, according to which the quality of the rule can be measured. The most important of these criteria is of course how good the rule is in predicting the target attribute. This is usually measured using the *confusion matrix* of the rule (Figure 3)

		Prediction	
		+	-
Reality	+	TP	FN
	-	FP	TN

Figure 3: Confusion matrix of a rule

TP stands for *true positive* and it is the number of data instances that satisfy the rules conditions and their target attribute matches the rules prediction, TN means *true negative* and it's the number of data instances that satisfies neither conditions nor prediction and so on. Confusion matrix serves as a basis for various different quality measures used to estimate the rules quality. Namely *true positive rate* ($\frac{TP}{TP+FN}$), *true negative rate* ($\frac{TN}{TN+FP}$), *accuracy* ($\frac{TP+TN}{TP+FP+TN+FN}$) or *precision* ($\frac{TP}{TP+FP}$). None of these single measures performs best on all kinds of data so their various combinations are often used instead.

There are other criteria that can reflect the quality of a rule, too. Shorter and simpler rules are undoubtedly more usable than longer and complicated ones (given their accuracy is comparable) so we might want to prefer shorter and simpler rules via the fitness function. Generally, we can influence the type and form of produced rules almost at will by addition of other criteria and conditions to the fitness function.

3.4 Crossover & mutation

In this section we assume individuals encoded by the high-level encoding for continuous attributes. Individuals encoded by the binary encoding can use basic versions of genetic operators as described in section 2.

Crossover for high-level encoding works almost like in the standard case, too. That means both parents are divided by random points into several sections and their genetic material is exchanged between some of the adjacent sections. Standard two point crossover (three sections) is shown on Figure 1 earlier. The only thing we have to keep in mind is the fact the parents must be aligned so that conditions on the same attribute are underneath each other. An example of simple crossover of two rules is shown on Figure 4

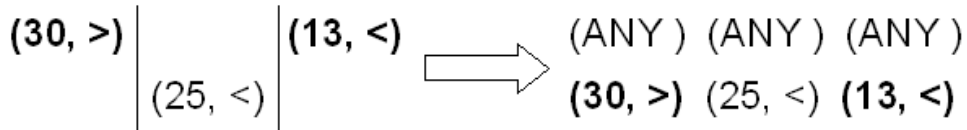


Figure 4: Example of crossover producing empty offspring

From the figure we can see that it is also possible that some of the offspring produced by this kind of crossover are empty. We can resolve this case either by performing the crossover according to different dividing points or by choice of different parents or by simply skipping such offspring.

Mutation lies in a subtle change of a condition. This is usually achieved by addition of a random (preferably small) number to the conditions value. That means for example the condition (*age* < 20) can mutate into (*age* < 22) or (*age* < 15). It is also possible to change the condition by randomly changing its operator, although such a change is more dramatic and possibly more destructive, too (after all, (*age* < 20) and (*age* > 20) are two very different conditions).

3.5 Generalization & specialization

Previous section covered basic versions of genetic operators for rule extraction. Now we present their modified versions used to make a rule either more general, or more specific. In the domain of rule extraction, these modified versions of genetic operators are often used along with the basic versions. The probability of favoring the modified versions over the basic versions is determined for each rule individually according to the rules properties. For example the more specific the rule is, the higher the probability of favoring specialization and vice versa. It is also a good idea to use generalization and specialization operators more often for rules with above-average fitness. The reason for this is that generalization and specialization operators are usually less destructive and therefore tend to preserve and exploit the promising rules more than random basic genetic operators. In the rest of this section we assume high-level encoding for continuous attributes as described in the section 3.1. Situation for individuals and conditions with binary encoding (discrete attributes) is similar.

3.5.1 Crossover

Generalization version of genetic crossover is demonstrated on the Figure 5

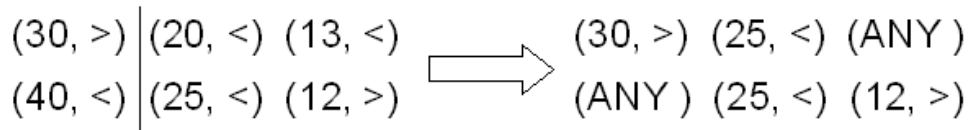


Figure 5: Generalizing crossover

Basic principle remains the same as in the case of standard crossover. Only instead of simply swapping the adjacent parts of both strings, a logical OR is performed. This may cause some conditions to effectively disappear from the rules antecedent (marked by ANY on Figure 5). Specialization crossover works the same way, except for using logical AND instead of OR. In this case, some of the conditions produced by specialization crossover may become unsatisfiable (this problem can be solved simply by omitting the condition from the crossover process).

3.5.2 Mutation

Generalizing mutation is demonstrated on Figure 6

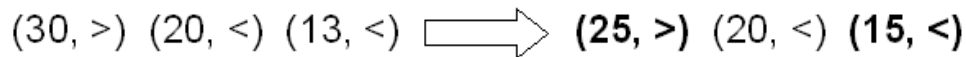


Figure 6: Generalizing mutation

As can be seen from the figure, generalizing mutation changes a value of an affected condition by random number in a way that produces more general condition. The process in the case of specialization stays the same, only the value is affected in the opposite direction to make the condition more specific.

3.5.3 Other operators

Apart from modified versions of common genetic operators described in previous sections there are other possibilities how to perform generalization or specialization of a rule, too. To make a rule more general we can drop an entire condition from the rule (so called *dropping operator*). And on the contrary, to make a rule more specific we can add a new condition into the rule. This new condition can be generated randomly, by some heuristic or by seeding from some data instance covered by the rule.

4 Estimation of distribution algorithms

Estimation of distribution algorithms (EDA) form another important group of evolutionary algorithms. Much like genetic algorithms, EDA are based on iterating through

individual generations. The difference from genetic algorithms lies in the way the transition between subsequent generations is performed. Estimation of population distribution and sampling from this estimate are used in EDA instead of genetic operators (like crossover and mutation). Basic version of EDA is outlined on the following pseudocode.

1. *population* = initializePopulation()
2. *population* = evaluatePopulation(*population*)
3. WHILE (NOT *stopping_criterion_reached*)
 - (a) *selection* = select(*population*)
 - (b) *distribution* = estimate(*selection*)
 - (c) *new_generation* = sample(*distribution*)
 - (d) *population* = evaluatePopulation(*new_generation*)
4. RETURN bestIndividual(*population*)

Initialization and **fitness evaluation** work exactly the same as described in case of genetic algorithms in section 2. Usually the population is initialized randomly and fitness function is problem dependent. **Selection** is based on an individuals fitness again (probability of selecting an individual should be proportional to its fitness). Estimate function performs **estimation** of probabilistic distribution for the current selection. This can be done in many various ways, some of which will be discussed in later sections. After the estimation the current population and selection are thrown away and the new generation is created by random **sampling** from the estimated distribution. This new generation is then evaluated and the whole process continues until the stopping criterion is reached. Options for the **stopping criterion** are the same as described for genetic algorithms in section 2.

From the pseudocode we can see that the basic principle is very similar to genetic algorithms. Like in genetic algorithms the search for the best individual is powered through fitness proportional selection. But in contrast to genetic algorithms independent application of crossover and mutation the EDA approach encapsulates the transition between the two subsequent generations into single probability distribution, from which the new generation is sampled. This simplifies the whole process and opens the door to better examine and explain the EDA methods through mathematical theory. More theoretical work along with more detailed explanation of individual EDA can be found in the monograph [7].

There are several versions of the basic concept presented in the pseudocode above according to the type of the domain (discrete or continuous) and the presence and complexity of dependencies between individual attributes. Following two sections present some of these versions in more detail. We will assume the attributes are either all discrete or all continuous. The "real-life" case in which some attributes are discrete and some are continuous will be briefly discussed in section 5.3.

4.1 Discrete domain

Let us have n data instances, each one with m attributes. Each of these attributes have discrete domain. The estimation phase of the EDA then consists of computing probability $p_k(\mathbf{x}) | S_k = p(x_1, x_2, \dots, x_m)$ where S_k is the current selection. The way we can do this depends on whether there are any dependencies among the attributes or not.

4.1.1 Univariate Marginal Distribution Algorithm (UMDA)

UMDA is probably the simplest algorithm implementing the EDA scheme. It works for data with attributes having discrete domains and it doesn't take into account any possible dependencies among individual attributes. The probability $p_k(\mathbf{x})$ is estimated simply from relative frequencies of values in the current population.

$$p_k(\mathbf{x}) = p(x|S_k) = \prod_{i=1}^m p_k(x_i) = \prod_{i=1}^m \frac{\sum_{j=1}^n \delta_j(X_i = x_i|S_k)}{n} \quad (1)$$

Here, δ_j is 1 if the value of i -th attribute of j -th data instance equals x_i and 0 otherwise. The rest of UMDA follows the standard pattern shown at the beginning of section 4.

4.1.2 Estimation of Bayesian Networks Algorithm (EBNA)

EBNA unlike the simple UMDA takes into account the possibility of dependencies among attributes. In each generation, the algorithm tries to build a Bayesian network to model the possible dependencies between attributes and learn it from the current selection. The probability distribution $p_k(\mathbf{x})$ is then estimated using this Bayesian network.

The construction of the Bayesian network itself forms a bottleneck of this algorithm and therefore we are usually not searching for the best structure possible and settle for a sufficiently good one. There are three variants of EBNA according to the method used for the Bayesian network structure construction:

EBNA_{PC} constructs the Bayesian network by systematically testing the conditional independencies between the attributes by the PC algorithm.

EBNA_{K2+pen} constructs the Bayesian network by searching the space of all possible network structures using the penalized K2 algorithm (see [3])

EBNA_{BIC} constructs the Bayesian network by searching the space of all possible network structures using the BIC criterium to evaluate found structures. $BIC \equiv -2\ln(L_{max}) + k\ln(N)$ where L_{max} is the maximum likelihood achievable by the network, k stands for the number of parameters of the network and N is the sample size.

4.2 Continuous domain

Let us have n data instances, each one with m attributes. Each of these attributes has continuous domain. The estimation phase of the EDA in this case consists of approximating the density function of the distribution of the current selection. The way we can do this depends again on whether there are any dependencies among the attributes or not.

4.2.1 Univariate Marginal Distribution Algorithm for continuous domains (UMDA_c)

UMDA_c is very similar to discrete UMDA in a way it is very simple implementation of the EDA scheme and it does not take into account any potential dependencies among individual attributes. The difference lies in the estimation phase. While UMDA tries to estimate discrete probability $p_k(\mathbf{x})$, UMDA_c estimates continuous density function. For each attribute in each generation UMDA_c performs some statistical tests on several possible density functions and chooses the likeliest one of them. After the density function itself was determined, its parameters are also estimated on the basis of maximum likelihood. The particular method of the parameter estimation depends on the density function used.

4.2.2 Estimation of Gaussian Networks Algorithm (EGNA)

EGNA forms a parallel to EBNA for continuous domains. The basic principle stays the same as in case of EBNA. In each generation, the algorithm tries to build a probabilistic graphical model (in this case Gaussian network) to model the possible dependencies between attributes and learn it from the current selection. The density function of the distribution is then estimated using this Gaussian network.

Like in case of EBNA, there are several versions of EGNA which differ in the method used to build the Gaussian network:

EGNA_{ee} builds the Gaussian network using the edge exclusion tests (basically testing whether a certain edge in the network can be removed) [4].

EGNA_{BGe} constructs the Gaussian network by searching the space of all possible network structures using the BGe metric to evaluate found structures (BGe stands for Bayesian Gaussian equivalence, see [6])

EGNA_{BIC} constructs the Gaussian network by searching the space of all possible network structures using the BIC criterium to evaluate found structures. BIC criterium was outlined earlier in section 4.1.2.

5 EDA for rule extraction

Let us now look at the ways the EDA paradigm can be used for rule extraction. The crucial and basically the only difference from the genetic algorithm approach described in section 3 is in the individual encoding. In the case of genetic algorithms the individual encoding is only constrained by genetic operators (crossover and mutation). So in theory we can use any encoding and we only have to have the appropriate and efficient operators developed for it. In EDA the situation is a bit different. The (relatively variable) genetic operators are replaced by the probability distribution estimation phase which is usually more restrictive in the means of its inputs (ie the population composed of the encoded individuals). Therefore we prefer to encode an individual as a vector of numbers (whole or real depending on whether the domain is discrete or continuous). This kind of encoding is simple to work with and furthermore we do not have to make any special changes to the standard versions of EDA presented in sections 4.1 and 4.2. In the following two sections the specific ways how to encode an individual (a rule) into a number vector are presented. We use the same notation as described in section 1.

5.1 Individual encoding for discrete EDA

The individual in this case means a rule composed of several conditions on attributes. All the attributes have discrete domain (the ones that cannot be discretized). We need the length of a rule to stay fixed for the purposes of EDA. This can be done by simply including exactly one condition for each attribute of the rule and allowing an empty condition (a condition that is satisfied always). In the discrete case, we therefore need to encode three operators $\{=, \neq, ANY\}$

Let us have an attribute *gender* with possible values male, female. We encode the possible values with consecutive integers. The operators are then encoded as shown on Figure 7

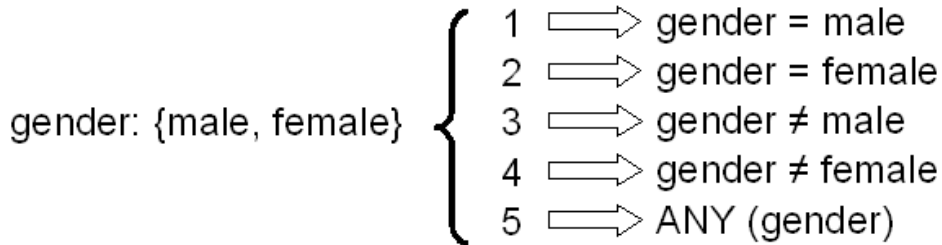


Figure 7: Condition encoding for discrete EDA

We can see the single rules condition is encoded into an integer. The whole rule would then be encoded as a vector of numbers corresponding to individual conditions. For example the rule

IF(*weight* \neq normal) & (*smoker*=yes) & (*gender*=ANY) **THAN** (*cancer_risk*=high)

with attributes $weight \in \{\text{normal, high, extreme}\}$, $smoker \in \{\text{yes, no}\}$ and $gender \in \{\text{male, female, unknown}\}$ will therefore be encoded as (4, 1, 7). Note that the consequent of the rule, in this case ($cancer_risk=\text{high}$) is not part of the rules encoding for the reasons described in the section 3.1 for the case of genetic algorithms.

5.2 Individual encoding for continuous EDA

Encoding of continuous attributes works basically the same as encoding of discrete attributes described in previous section. The main difference is we use real numbers instead of integers in the encoded vector. There are also two more operators (< and >) that make sense for conditions on continuous attributes. In total we therefore need to encode five operators $\{=, \neq, <, >, ANY\}$

Let's have an attribute $weight \in (0, 150]$. The encoding of a condition on this attribute is illustrated on Figure 8

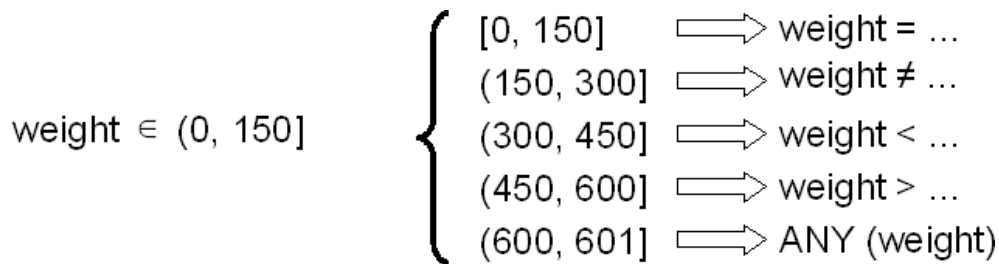


Figure 8: Condition encoding for continuous EDA

The whole rule is once again encoded as a vector composed of codes of individual conditions and the consequent is not part of the encoding.

5.3 Notes on mixed domains

In the previous two sections we showed how the individuals with discrete and continuous domains can be encoded for the purposes of EDA. However, in most real life cases we need to solve problems described by both discrete and continuous attributes. One solution to this problem is to simply treat discrete attributes as continuous. This may be straightforward but since we are throwing away the simplicity (and properties) of discrete attributes, the computational process will often become unnecessarily complicated. Other solution may lay in splitting the input data set into several data sets according to the certain combinations of values of the discrete attributes (we can use some of the standard clustering methods to do that). By such a clustering of the input data set we can effectively get rid of the discrete attributes in individual partial data sets, estimate the distribution of each partial data set separately and finally combine the estimated distributions together in the sampling phase of the algorithm. With this solution, the discrete attributes can be treated like discrete, but on the other hand we are losing some information by clustering and the algorithm will tend to be slower while dealing with the split input data set.

6 Conclusion

In the article we have shown how the two main groups of evolutionary algorithms - genetic algorithms and EDA - can be adjusted and used for the purposes of rule extraction. Surveys and empirical measurements have been performed to compare these evolutionary approaches to rule extraction with the conventional rule extraction algorithms. There is for example empirical comparison [10] comparing the performance of EDA versus CN2 ([1]) and RIPPER ([2]) algorithms. This and similar works showed that the evolutionary algorithms are in some cases capable of outperforming the traditional algorithms. Given that the rule extraction is probably one of the most important (if not the most important) areas of data mining today, it seems the evolutionary approaches to rule extraction are worth of further research. That is especially true for EDA approach. Since the EDA paradigm is still relatively new, the future work has the potential to further improve the EDA-based rule extraction techniques, possibly to the point where they could dominate traditional techniques. Furthermore - the direct comparison of both genetic and EDA approaches to rule extraction is still missing in the literature, too. Such a comparison could shed light on the cases where these two (seemingly similar) evolutionary approaches differ in performance. This knowledge could then help to better directing the research in both areas.

Acknowledgment

The research reported in this paper has been supported by the grant ME 949 of the Ministry of Education, Youth and Sports of the Czech Republic.

Bibliography

- [1] P. Clark, T. Nibblet, "The CN2 induction algorithm", *Machine Learning*, 3(4):261-283, 1989
- [2] W. Cohen, "Fast effective rule induction", *Proceedings of the Twelfth International Conference in Machine Learning*, 115-123, 1995
- [3] G. Cooper, E. Herskovitz, "A Bayesian method for the induction of probabilistic networks from data", *Machine Learning*, 9:330-347, 1992
- [4] A. P. Dempster, "Covariance selection", *Biometrics*, 28:157-175, 1972
- [5] A. A. Freitas, "Data Mining and Knowledge Discovery with Evolutionary Algorithms", Springer, Berlin, 2002
- [6] D. Geiger, D. Heckerman, "Learning Gaussian networks", *Proceedings of Tenth Conference on Uncertainty in Artificial Intelligence*, 235-243, 1994
- [7] P. Larrañaga, J. A. Lozano, "Estimation of distribution algorithms: A new tool for evolutionary computation", Kluwer, Boston, 2002

- [8] C. Pitangui, G. Zaverucha, “Genetic Based Machine Learning: Merging Pittsburgh and Michigan, an Implicit Feature Selection Mechanism and a New Crossover Operator”, Proceedings of the Sixth International Conference on Hybrid Intelligent Systems, 2006
- [9] C. R. Reeves, J. E. Rowe, “Genetic Algorithms: Principles and Perspectives”, Kluwer, Boston, 2003
- [10] B. Sierra, E. A. Jiménez, I. Inza, P. Larrañaga, “Rule Induction by Estimation of Distribution Algorithms”, Estimation of distribution algorithms: A new tool for evolutionary computation, Kluwer, Boston, 2002

Evolve komunikace umělých bytostí

Tomáš Holan

KSVI MFF UK

Malostranské nám. 25, Praha

holan@ksvi.mff.cuni.cz

Abstrakt Text se zabývá evolucí komunikace (umělých) organismů. Popisuje požadavky na komunikaci, návrh formátu podle těchto požadavků a výsledky provedených pokusů s evolucí komunikace.

Klíčová slova: Umělý život, komunikace, evoluce.

1 Úvod

V textech [1] a [2] jsme popisovali simulaci prostředí, ve kterém žijí umělé bytosti. Jako motivaci těchto experimentů jsme uváděli snahu podnítit a sledovat vývoj komunikace takových bytostí. Tento text popisuje takový pokus o vývoj komunikace.

1.1 Umělý svět

Umělý svět, v němž provádíme simulaci, byl popsán v [1]. Jde o čtverečkovou síť, na které roste potrava a po které se pohybují umělé bytosti, pomocí přesunů mezi jednotlivými čtverečky.

Kroky umělého organismu řídí jednoduchý algoritmus rozhodující se podle obsahu sousedních políček. Algoritmus je reprezentovaný tabulkou a je předmětem evoluce.

Organismy se množí pomocí páření, k páření dojde, pokud jsou parametry organismů vyhodnoceny jako kompatibilní a pokud algoritmus rozhodne, že se má provést akce páření.

Ve v tomto textu popisovaném experimentu přibude k podmínkám páření úspěšná komunikace obou jedinců.

2 Návrh formy komunikace

To, co dělají lidé a co dělají zvířata, co se vyvíjelo miliony let a to, co chceme vyvíjet teď během několika minut nebo hodin v počítači, budeme nazývat *komunikace*.

To, co budou provozovat dva konkrétní jedinci v jednom konkrétním čase a místě bude jedna konkrétní instance komunikace; této instanci budeme říkat *rozhovor*.

Komunikace, jazyk, prostředek výměny informací mezi jedinci. . . může mít mnoho různých *forem komunikace*.

Na jednom konci spektra forem komunikace leží pouhé vyhodnocení kompatibility parametrů (vzhledu, pachu, chování) – sem bychom mohli zahrnout experimenty popsané v [1], kde jedinci pouze zjišťovali, zda jejich představy o ceně páření a rodičovských investicích mají společný průnik.

Na druhém konci spektra by potom mohlo ležet něco jako rozhovor dvou lidí v přirozeném jazyku, kdy komunikující jedinci ze slovníku o statisících slov vybírají slova, skládají je do vět, kterými k sobě promlouvají a po určitém počtu vyměněných vět se domluví, nebo nedomluví. Formou komunikace je zde jazyk, chápaný ne jako pouhá množina slov, ale se vši svou sémantikou a s tím, jaký dopad mají vyslovená slova (věty) na obsah vědomí komunikujících jedinců.

Naším cílem bylo pokusit se navrhnout formu komunikace, která by nebyla tak triviální jako v prvním případě a přitom by bylo v silách organismů simulovaných na běžném počítači naučit se jí vyjadřovat.

2.1 Výchozí požadavky na komunikaci

Před samotným návrhem formy komunikace vhodné pro počítačovou evoluci jsme si stanovili požadavky, které by tato forma měla splňovat:

1. Aby jedinci měli nějaký důvod komunikovat a aby byl nějaký důvod pro vývoj/zlepšování komunikace, bude úspěšná¹ komunikace nutnou podmínkou páření.
2. Komunikace jedince bude závislá na parametrech, které budou součástí dědičné informace jedince. Tyto parametry budou předmětem selekce, křížení a mutace, aby se komunikace mohla v průběhu simulace vyvíjet.
3. Rozhovor nebude jen porovnání nějakých hodnot, ani dvojice monologů, ale posloupnost střídavých promluv jedinců, kde na promluvě jednoho jedince budou záležet další promluvy druhého
4. Délka rozhovoru nebude předem daná, bude záležet na kvalitě komunikace, jak rychle se jedinci dokážou domluvit, nebo naopak poznat, že se nedomluví
5. Každý rozhovor skončí úspěchem (partner v rozhovoru je vhodný pro páření) nebo neúspěchem (partner v rozhovoru není vhodný pro páření)
6. Každý rozhovor bude účastníky něco stát, jeho cena by měla růst s délkou, aby byly zvýhodněny takové způsoby komunikace, které vhodnost či nevhodnost partnera dokážou poznat rychleji.
7. Krom toho, jak se bude komunikace vyvíjet z generaci na generaci, by mělo docházet k určitému vývoji i během života jednoho jedince, podobně jako vedle sebe stojí dlouhý, pozvolný vývoj třeba českého jazyka a to, jak se konkrétní dítě učí mluvit nebo psát (fylogeneze a ontogeneze).

Tento požadavek by navíc mohl zajistit, že se nebudou pářit příliš mladí jedinci, což by mohlo přispět dalším strategiím v dříve zkoumané evoluci parametrů

¹Úspěšná bude taková komunikace, kde se oba jedinci shodnou na tom, že se spolu chtějí pářit

páření; umožnilo by to chování, kdy jedinec nejdříve musí poznat několik potenciálních partnerů, podle jejich parametrů získá představu o hodnotách rodičovských investic a dalších parametrů páření — a podle toho potom může nastavit své požadavky.

2.2 Komunikace pomocí tabulek

Výše uvedené požadavky nás vedly k následujícímu návrhu formy komunikace:

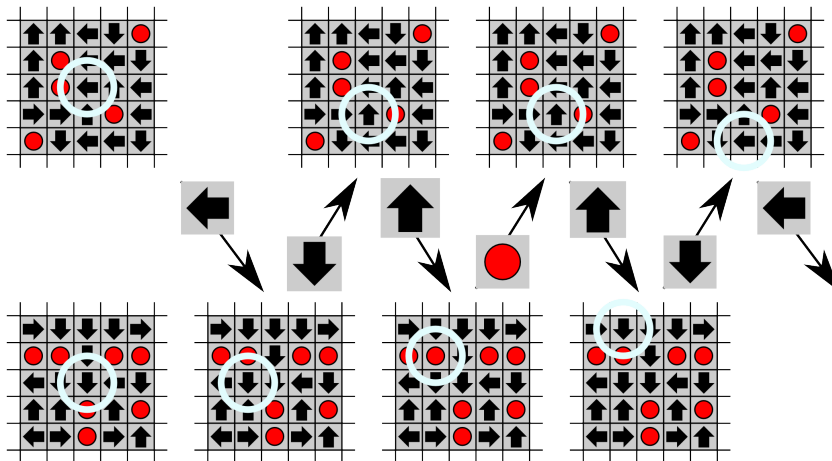
1. Každý jedinec bude mít svoji *tabulku komunikace* a *ukazatel* označující *aktuální políčko*.
2. Tabulka komunikace bude dvourozměrná tabulka obsahující *slova*.
3. Na začátku rozhovoru ukazatel každého jedince bude ukazovat na *výchozí políčko* jeho tabulky.
4. Rozhovor bude vypadat tak, že střídavě každý z účastníků vysloví slovo obsažené v aktuálním políčku, a provede se akce s tímto slovem spojená.
Důvod: Rozhovor tak bude sestávat ze střídavých promluv obou účastníků.
5. Oba účastníci začínají rozhovor s určitým kreditem (100), od kterého se jim odečítá cena za každou promluvu. Pokud některý z účastníků vyčerpá svůj kredit, rozhovor končí neúspěchem.
Důvod: Omezená délka rozhovoru.
6. Kromě tabulky komunikace bude mít každý jedinec ještě stejně velkou *tabulku návštěv*, na které se mu bude zaznamenávat, kolikrát (během svého života) daným políčkem prošel. Cena za každou promluvu bude odvozena od této tabulky návštěv, na začátku vysoká (10), s každou návštěvou políčka se bude snižovat.
Důvod: Tabulka návštěv by měla způsobit, že opakované, známé cesty v tabulce komunikace budou lacinější než nová políčka s vysokou cenou, mělo by to vést ke společnému jazyku uvnitř populace.
Důvod: Tabulka návštěv by zároveň měla snižovat pravděpodobnost úspěšné komunikace jedinců, kteří dosud komunikovali málo.
7. Možná slova jsou:
send_credit promlouvající předá poslouchajícímu část svého kreditu (10),
left, right, up, down posunou ukazatel poslouchajícího v příslušném směru (nejpravější políčka sousedí s nejlevějšími, nejvyšší s nejnižšími).

Důvod: Další vývoj komunikace bude záležet na tom, co bylo právě teď řečeno (účastníci nepovedou dva monology).

*Důvod: **send_credit** je akce, která snižuje schopnost účastníka pokračovat v rozhovoru; pokud ji protějšek nebude (včas) oplácet, dojde dárci kredit a rozhovor skončí neúspěchem.*

8. Komunikace bude úspěšná, pokud každý z účastníků rozhovoru vysloví a provede předem stanovený počet slov a akcí **send_credit**.
9. Kredit předaný operací **send_credit** bude odečítán nejen od kreditu komunikace, ale od energie jedince. a přijatý kredit bude přičítán k energii jedince (takže mohou vznikat příživníci).
*Důvod: Odečítání darovaného kreditu od energie jedince slouží jako zábrana proti tomu, aby vývoj komunikačních tabulek nevedl k jejich zaplnění akcemi **send_credit**.*
10. Každý jedinec bude ve skutečnosti mít dvě tabulky komunikace a k nim dvě tabulky návštěv, jednu pro otcovskou roli, kdy zahajuje komunikaci (a tedy začíná na výchozím políčku) a jednu pro mateřskou roli, kdy začíná jako poslouchající (a tedy, s výjimkou případů, kdy první akcí začínajícího bude **send_credit**, bude začínat rozhovor na políčku, kam byl jeho ukazatel přesunut první akcí rozhovoru).

Obrázek 1 zobrazuje příklad začátku rozhovoru dvou jedinců — horního, který zahajuje komunikaci a dolního. Šipky a značky na nich představují vyslovená slova. Světlý kroužek v každé tabulce označuje aktuální políčko.



Obrázek 1: Princip komunikace pomocí tabulek

3 Příklad úspěšné komunikace

Toto je příklad úspěšného rozhovoru, s počátečním kreditem 100. Podmínkou úspěšné komunikace bylo, aby každý jedinec nejméně třikrát provedl operaci **send_credit**.

K každém řádky je vždy promluva i odpověď, s údaji o pozici v tabulce, kreditu a vysloveném slovu:

5:5 (100)	down	5:6 (100)	right
6:5 (99)	send_credit	5:6 (109)	right
7:5 (88)	right	6:6 (108)	left
6:5 (87)	send_credit	6:6 (117)	left
5:5 (76)	down	6:7 (116)	down
5:6 (75)	up	6:6 (115)	left
4:6 (74)	down	6:7 (114)	down
4:7 (73)	right	7:7 (113)	send_credit
4:7 (82)	right	8:7 (102)	down
4:8 (81)	down	8:8 (101)	down
4:9 (80)	down	8:9 (93)	up
4:8 (74)	down	8:0 (84)	right
5:8 (73)	left	7:0 (75)	down
5:9 (72)	send_credit	7:0 (77)	down
5:0 (61)	up	7:9 (70)	right
6:0 (60)	up	7:8 (61)	up
6:9 (51)	left	6:8 (60)	send_credit
6:9 (58)	left	5:8 (49)	down
6:0 (56)	up	5:7 (40)	left
5:0 (48)	up	5:6 (33)	right
6:0 (47)	up	5:5 (32)	right
7:0 (40)	down	5:6 (31)	right
8:0 (39)	right	6:6 (30)	left
7:0 (30)	down	6:7 (29)	down
7:1 (29)	send_credit	6:7 (38)	down
7:2 (18)	right	7:7 (37)	send_credit

4 Testy

Na vyhodnocení experimentu jsme si předem navrhli sadu testů, zde popíšeme výsledky několika z nich.

1. T01: Jestli bytosti s komunikací vůbec přežijí
2. T02: Srovnat vliv komunikace na evoluci i na jedince
3. T03: Příklad výsledné tabulky akcí a tabulky návštěv
4. T04: Rozlišovací schopnost komunikace

4.1 T01: Jestli bytosti s komunikací vůbec přežijí

Ano, přežijí. Viz další test.

4.2 T02: Srovnat vliv komunikace na evoluci i na jedince

Po spuštění simulace jedinci mají náhodný algoritmus (a pokud se evolují i parametry páření, tak i náhodné parametry páření). To znamená, že neumí hledat potravu ani

partnera na páření a v prvních krocích většina populace vyhyne. Záleží na velikosti mapy, dostupnosti potravy a náhodě, jestli jich přežije dost na to, aby se dokázali rozmnožit a znovu osídlit prostředí.

Když se do modelu přidá komunikace jako nutná podmínka páření, znamená to další omezení možnosti páření a tedy ztížení přežití.

4.2.1 Popis testu

V deseti průbězích simulace bez komunikace a v deseti průbězích simulace s komunikací zaznamenávat počty jedinců až do doby, kdy se začnou zase množit a velikost populace se ustálí, a následně porovnat (dále uvedené) ukazatele.

Délka simulace do ustálení byla stanovena na 5000 kroků, uvádíme položené otázky spolu se získanými odpověďmi pro tyto případy:

- BK: bez komunikace
- K1: s komunikací, podmínka úspěchu je jedna provedená dotace na každé straně
- K3: s komunikací, podmínka úspěchu jsou tři provedené dotace na každé straně.

4.2.2 Výsledky

- **kolikrát se populaci podařilo přežít:**
BK: všechny, K1: všechny, K3: populace 2 a 5 = dvě z deseti = 2/10
- **minimální počet jedinců v populaci, kdy – mediány:**
BK: 68/64.krok, K1: 60/90.krok, K3: 42/198.krok
- **maximální počet jedinců v populaci po přežití:**
BK, K1: skoro stejné, K3: stále roste
- **počet kroků, kdy po počátečním úbytku populace nabude výchozí velikosti resp. jiného určeného počtu, pokud výchozího počtu nenabude:**
Měřeno, kdy velikost populace překročí 100 jedinců (mediány počtu kroků)
BK: 145, K1: 241, K3: 660, ALE v K3 některé populace vyhynuly (8 z 10).
- **medián věku, průměr a medián energie jedince, po 5000 krocích:**
 - BK: 24 173 155
 - K1: 31 202 183
 - K3: 48 269 254— tj. s požadavkem na komunikaci ROSTE věk a roste energie jedince

4.3 T03: Příklad výsledné tabulky akcí a tabulky návštěv

Tento příklad ukazuje komunikační tabulky a tabulky návštěv dvou jedinců po úspěšné komunikaci a záznam z jejich komunikace.

Otec:

^ v < @ v v ^ < v <	0	0	0	0	0	0	0	0	0	0
> v > v v @ v @ ^ @	0	0	0	0	0	0	0	0	0	0
< ^ v ^ > ^ v > ^ <	0	0	0	0	0	1	0	0	0	0
^ < > v v > > > < v	0	0	0	0	0	1	1	1	1	2
^ @ ^ @ ^ @ < > v v	0	0	0	0	0	4	4	3	0	1
> ^ @ < ^ ^ > ^ ^ <	0	0	0	1	1	36	31	3	0	0
> v v < < v > ^ < ^	0	0	0	1	1	2	1	1	0	0
< v ^ ^ > ^ < ^ @ v	0	0	0	0	1	1	0	0	0	0
^ v ^ @ > < > ^ > <	0	0	0	0	0	0	0	0	0	0
> @ ^ < ^ @ v v < <	0	0	0	0	0	0	0	0	0	0

Matka:

^ ^ ^ < > @ v @ @ ^	0	0	0	0	0	2	2	0	0	0
> v ^ > < v > < @	0	0	0	0	0	0	1	0	0	0
@ < > v < ^ ^ v v ^	0	0	5	3	2	3	2	0	0	0
v > > @ < v v > v ^	0	2	10	8	4	1	6	0	0	0
> ^ < < > v > > v v	2	3	11	13	30	22	12	3	4	3
< ^ v > v < > > < ^	1	3	9	9	31	164	4	0	2	0
v > < > ^ > > v < <	0	0	5	2	25	183	11	3	3	0
> ^ ^ v v < v ^ v <	0	0	1	1	2	4	2	2	2	1
v > ^ < > @ > @ < @	0	0	0	0	4	0	0	0	0	0
@ @ < @ < < < ^ > ^	0	0	0	0	4	2	0	0	0	0

Záznam rozhovoru:

5:5 (100) __	up	5:4 (100) __	down
5:6 (99) __	down	5:5 (99) __	left
4:6 (98) __	left	4:5 (98) __	down
4:7 (92) __	right	5:5 (97) __	left
3:7 (85) __	up	5:4 (96) __	down
3:8 (76) __	credit_send	5:4 (105) __	down
3:9 (57) __	left	4:4 (104) __	right
4:9 (48) __	up	4:3 (103) __	left
3:9 (39) __	left	3:3 (102) __	credit_send

4.4 T04: Rozlišovací schopnost komunikace I.

4.4.1 Popis testu

Nechat běžet 10 evolucí, třeba 10.000 kroků, aby byl čas na vývoj určitého společného způsobu komunikace, z každé evoluce náhodně vybrat 10 jedinců (jejich tabulku komunikace i tabulku návštěv) a potom spočítat výsledek rozhovoru každého s každým, v obou rolích. Tím pro každou dvojici evolucí E_i a E_j získáme 10x10 rozhovorů, kde jedinec z E_i zahajuje komunikaci. Do tabulky T na pole o souřadnicích (i, j) potom uložíme procento úspěšnosti komunikace (tabulka nebude symetrická, kvůli různým rolím).

Pokud komunikace dokáže rozlišit jedince ze stejné a z cizí populace, měla by být na diagonále velká čísla a jinde malá.

4.4.2 Výsledky

Vývoj komunikace i vyhodnocení její úspěšnosti závisí na mnoha faktorech: na nastavení podmínky úspěchu komunikace, na počátečním kreditu jedinců, na ceně operace `send_credit`, ale i na délce evoluce, míře mutace a způsobu křížení.

Opakovali jsme tento test pro různá nastavení, ale při žádném z nich jsme nepozorovali výrazný rozdíl mezi čísly na diagonále a mimo ni.

Zde uvádíme příklad jedné z výsledných tabulek T :

25	26	28	23	20	23	28	15	27	28
38	34	28	33	26	24	36	20	32	25
37	33	30	28	28	24	39	35	28	28
41	43	34	39	16	25	31	20	28	32
36	44	44	40	29	30	37	30	44	31
36	40	35	21	31	36	24	29	24	28
39	36	35	22	21	30	37	20	35	34
41	35	33	23	32	28	39	29	35	31
39	33	35	33	17	33	34	28	34	22
32	37	39	38	42	29	35	31	41	28

5 Závěr

Představili jsme návrh způsobu komunikace pro umělé bytosti a několik testů. Ukázalo se, že se simulované bytosti mohou naučit tímto způsobem komunikovat, ale nepotvrdilo se, že by naučený způsob komunikace dokázal odlišit jedince z jiné populace.

Literatura

- [1] Holan, T.: Evoluce parametrů páření. In Vojtáš, Peter (Ed.), ITAT 2008 Informačné technológie - Aplikácie a Teória. Košice: UPJŠ, 2008. pp.49-52.
- [2] Holan, T.: Rodičovské investice umělých bytostí. In Obdržálek, David a Štanclová, Jana a Plátek, Martin: MIS 2008, Sborník semináře. Praha: Matfyzpress pp.11-18.

On the Role of Statistical Methods in Machine Translation Between Related Languages

Petr Homola Jernej Vičič Vladislav Kuboň

Abstrakt. Statistical machine translation is mainstream in today's MT research. Nevertheless there are predominantly rule-based MT systems in development, especially for translation between more or less related languages. This paper summarizes recent research in MT among Slavic languages and discusses the role of statistical modules in the system's architecture. The authors argue that statistical methods should be postponed as far as possible in the module chain since once used, they introduce noise into intermediate results which rule-based methods are unable to cope with. On the other hand it is inevitable to use some auxiliary statistical modules to achieve state-of-the-art translation quality, such as a ranker based on a language model of the target language. Statistical methods are mostly used to cope with ambiguous intermediate results and leaving out such methods can lead to an explosion of intermediate data that cannot be coped with.

1 Introduction

The rule-based shallow-transfer approach to MT has a long tradition and has been successfully used for automatic translation between closely related languages. Shallow-transfer systems usually use a morphological disambiguator before the transfer phase which typically works deterministically. This is obviously a huge restriction, especially for lexical transfer, since in most language pairs, many words have more translations depending on the syntactic and/or semantic context. The parser and structural transfer also produce ambiguous output relatively often. Even if a shallow-transfer MT system would be designed for a narrow domain which significantly simplifies the lexicon and reduces lexical ambiguity in translated texts, a crucial problem remains the morphological disambiguation which is mostly performed by a statistical tagger. Even if we had enough morphologically annotated data to train the tagger, the state-of-the-art taggers have a high error rate. Since the morphological disambiguation is the first module in the core of the system, errors are introduced into the processed data at the very beginning of the translation process and it becomes impossible for the subsequent modules to work properly.

We have implemented an experimental MT framework that uses rule-based partial parser and shallow transfer. The tagger was omitted and finally, we have added a ranker based on a trigram language model as the last module in the translation pipeline. This paper focuses on our experiment with translation from Czech into Slovenian and subsequently from Slovenian into Slovak which shows that a sophisti-

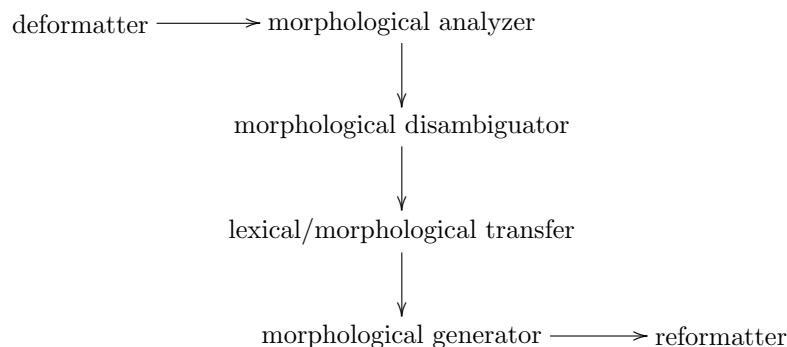


Figure 1: The shallow-transfer MT architecture as proposed by Hajič et al. (2000)

cated combination of two MT systems can be used to obtain a new translation pair with reasonable quality (cf. also (Babych et al., 2007)).

The paper is organized as follows: Section 2 contains a brief description of related research. In Section 3, we describe a modification of the commonly used shallow-transfer approach that leads to higher translation quality. In Section 4, we present the method of combining two MT systems together at a deep linguistic level and its evaluation. Finally, we draw some conclusions in Section 5.

2 Recent research

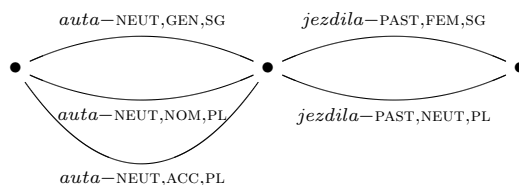
2.1 Shallow-transfer MT

Translation between closely related languages has been explored in detail by Dyvik (1995) for Scandinavian languages. The shallow-transfer approach to rule-based MT has been first proposed by Hajič et al. (2000) for translation from Czech into Slovak. As there are practically no syntactic nor semantic differences between the two languages, their system uses a direct lemma-to-lemma lexical transfer with a one-to-one dictionary. Later, the system was extended to the language pair Czech-Polish (Dębowski et al., 2002) and finally, a partial parser has been added to the system’s architecture for the language pair Czech-Lithuanian (Hajič et al., 2003).

Czech is a language with rich inflection, i.e., a word usually has many different endings that express various morphological categories. The morphological analyzer assigns a set of lemmas and tags to each word. As it was necessary to have only one tag for each word in the transfer phase, a statistical tagger was used with an accuracy of approx. 94% (Hajič and Kuboň, 2003).

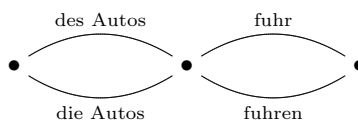
and a word-to-word translation into German would give a correct translation. However, both words are morphologically ambiguous and if we omit the tagger, each input word form would split in several morphologically distinct lemma-tag pairs. For example, some Czech adjectival word forms can have up to 27 distinct morphological meanings. The following structure would be the input of the subsequent modules:

(3)



Without a parser or another module which would resolve the ambiguity, the system would output the following German representation after the morphological synthesis:

(4)



We see that two edges have been merged into one due to morphological syncretism but there are still four possible outputs if one would consider all paths through the multigraph from the initial node to the end node.

We decided to add a module to the system that would find the ‘best’ path through the multigraph. We suggest to use a language model for the target language. In our experiments, a trigram model based on word forms and trained on about 20 million words from the Wikipedia has been used.

In the resulting German representation (in the above example), the correct path through the multigraph would be found correctly. Nevertheless, there is another problem — for longer sentences, this approach leads to a combinatorial explosion. Fortunately, the solution is comparatively simple: we have added a non-deterministic partial parser based on LFG (Bresnan, 2002) and our experiments show that even if we parse only noun and prepositional phrases, the morphological ambiguity gets reduced significantly even for languages with rich inflection, such as Czech. Syntactic analysis is needed anyway to mark local dependencies that will be used in the structural transfer. The improved architecture is given in Figure 2.

4 MT Using a ‘Bridge Language’

We did two experiments with coupled MT systems translating from Czech to Slovak through Slovenian as the intermediary language. The first system simply pipes the output of the Czech-to-Slovenian MT system into the Slovenian-to-Slovak one. The other experiment couples both MT systems at a higher level, omitting morphological

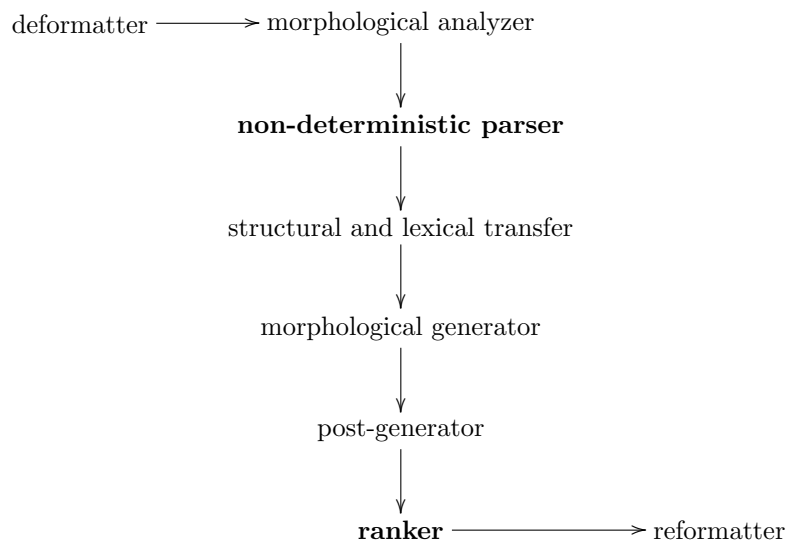


Figure 2: Improved shallow-transfer approach (additional modules are in bold)

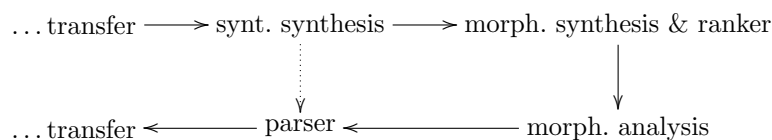


Figure 3: Combining two MT systems (with a ‘shortcut’)

synthesis and statistical ranker in the first language pair. As our experiments have shown, the latter approach produces significantly better translation.

The high-level pipeline is schematized in Figure 3. The dotted arrow denotes the ‘shortcut’ which has been taken in the system architecture to omit morphological synthesis and ranker in the first language pair.

	BLEU	NIST
simple pipe	0.1690	3.5916
high-level pipe	0.2303	4.1737

Table 1: Evaluation of the coupled MT systems

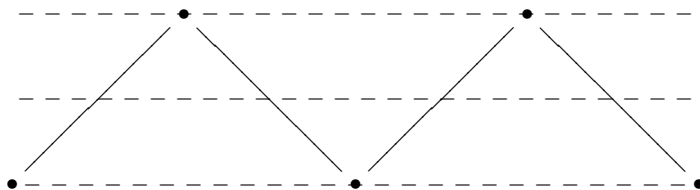


Figure 4: Vauquois triangles for the simple pipe of two MT systems

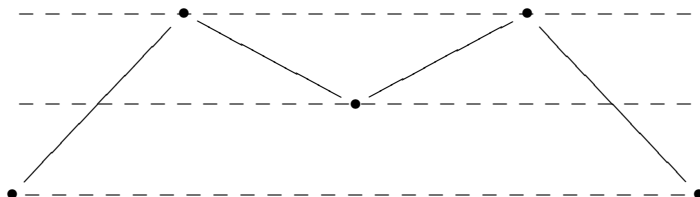


Figure 5: Vauquois triangles for the sophisticated combination of two MT systems

The two approaches could be schematically expressed by the translation (Vauquois) triangle as given in Figures 4 and 5. We see that in the high-level pipeline, the process does not return to the lowest level of linguistic representation (linearized sequence of word forms) but remains at a middle stage, in our case — informally expressed — between morphology and syntax.

To evaluate the difference in translation quality, we have used the MT evaluation metrics BLEU (Papineni et al., 2001) and NIST (Doddington, 2002) although they are based on words which is a crucial problem for languages with rich inflection (a comparatively small difference in an ending of a word is penalized as if the translation was completely wrong; see (Callison-Burch et al., 2006) for a detailed discussion of the deficits of BLEU). Nevertheless if we compare the scores given by both metrics, we see that they correlate in expressing which method gives better results.

The results are summarized in Table 1.

5 Discussion

We have presented an MT method which exploits a natural languages as ‘bridge language’. The evaluation of our experiments with MT from Czech to Slovak through Slovenian clearly shows that we get better results if we couple the two MT systems at a higher level. The main point is that the statistical ranker is not used in the first MT system, postponing the selection of one translation hypothesis to a later stage. At the first sight, this strategy may seem to cause huge ambiguity in the translation

process. However, if we do not use morphological synthesis in the first MT system, we do not need morphological analysis in the second system either, thus what we can avoid is the morphological ambiguity of the input in the second MT system (which is extremely important for languages with rich inflection, such as Slovenian). In other words, the parser in the second MT system deals with ambiguity of a different type, namely structural and semantic, which resulted from the first system and could not have been resolved prior to the ranking.

The comparison of both systems (on the same input text) has also brought an interesting observation: the MT system with the more sophisticated coupling works faster due to the fact that morphological ambiguity of the intermediary representation (which is the input of the MT for the second language pair) is widely reduced.

Acknowledgments

The presented research has been supported by the grant No. 1ET100300517 of the GAAV ČR.

References

- Bogdan Babych, Anthony Hartley, and Serge Sharoff. Translating from under-resourced languages: comparing direct transfer against pivot translation. In *Proceedings of MT Summit XI*, pages 29–35, Copenhagen, Denmark, 2007.
- Eckhard Bick and Lars Nygaard. Using Danish as a CG Interlingua: A Wide-Coverage Norwegian-English Machine Translation System. In *Proceedings of NODALIDA*, Tartu, Estonia, 2007.
- Joan Bresnan. *Lexical-functional syntax*. Blackwell Textbooks in Linguistics, New York, 2002.
- Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluating the Role of BLEU in Machine Translation Research. In *Proceedings of the EACL'06*, Trento, Italy, 2006.
- Lukasz Dębowski, Jan Hajič, and Vladislav Kuboň. Testing the limits — adding a new language to an MT system. *Prague Bulletin of Mathematical Linguistics*, 78, 2002.
- George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the ARPA Workshop on Human Language Technology*, 2002.
- Helge Dyvik. Exploiting Structural Similarities in Machine Translation. *Computers and Humanities*, 28:225–245, 1995.
- Jan Hajič, Petr Homola, and Vladislav Kuboň. A simple multilingual machine translation system. In *Proceedings of the MT Summit IX*, New Orleans, 2003.

Jan Hajič and Vladislav Kuboň. Tagging as a Key to Successful MT. In *Proceedings of the Malý informatický seminář*, Josefův Důl, 2003.

Jan Hajič, Jan Hric, and Vladislav Kuboň. Machine translation of very close languages. In *Proceedings of the 6th Applied Natural Language Processing Conference*, pages 7–12, Seattle, Washington, USA, 2000.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, 2001.

Metodika kvantitativního hodnocení detekčních kódů

Lucie Kárná

AŽD Praha, s.r.o
Žirovnická 2, Praha 10
karna.lucie@azd.cz

Abstrakt. Jednou z důležitých technik používaných v bezpečnostně relevantních aplikacích jsou detekční kódy, určené především k ochraně informací při přenosu v komunikačních systémech nebo při jejich uložení na paměťovém médiu. Pro jejich návrh a zejména pro kvantitativní hodnocení jejich detekčních vlastností byla v rámci disertační práce autorky vytvořena přehledná a korektní metodika, zaměřená na potřeby a možnosti praxe.

Klíčová slova: hodnocení bezpečnostních kódů, detekční kódy, návrh bezpečnostních kódů.

1 Úvod

Detekční kódy jsou jednou z důležitých technik, sloužících k ochraně dat před poškozením při přenosu v komunikačních systémech nebo při jejich uložení na paměťovém médiu.

Pro běžné aplikace obvykle postačuje, je-li kód schopen odhalit či opravit dostatečný počet chyb. V bezpečnostně relevantních systémech (řízení dopravy nebo jiných bezpečnostně kritických procesů) však bývá požadován exaktní kvantitativní průkaz bezpečnosti systému. Jedním z parametrů, které tuto bezpečnost ovlivňují, jsou pravděpodobnosti selhání detekčních kódů, které systém používá. Selháním kódu se rozumí nedetekovaná chyba — situace, kdy kód neodhalí chybu v datech a tato chybná data jsou pak dalšími prvky systému považována za správná.

V literatuře z oblasti matematické teorie kódování se vyskytuje řada vět a pravidel, vhodných ke zkoumání vlastností kódů. Praktickým problémem je však výpočetní náročnost některých metod, kterou se teoreticky zaměřená literatura obvykle nezabývá, a která je v případě dlouhých kódových slov podstatnou otázkou.

Na druhé straně v literatuře z oblasti sdělovací techniky se obvykle jako měřítko detekčních schopností kódů používá pouze jejich minimální vzdálenost, případně pravděpodobnost nedetekované chyby pro určitou konkrétní hodnotu chybovosti kanálu. Rovněž se zde extenzivně využívají numerické postupy, věrohodnost jejichž výsledků je omezená.

Proto byla v rámci disertační práce autorky nově vypracována komplexní metodika návrhu a kvantitativního hodnocení detekčních kódů. Metodika se týká lineárních

blokových kódů, především cyklických kódů a kódů od nich odvozených. Jedná se o nejčastěji využívané typy detekčních blokových kódů.

2 Teorie kódování

V následujících odstavcích stručně shrnu základní pojmy z teorie kódování.

2.1 Lineární kód

Bud' dáno q -prvkové konečné těleso $\mathbf{T} = \mathbf{GF}(q)$. Na tomto tělese definujeme q -nární lineární $(n, k)_q$ -kód jako lineární podprostor prostoru \mathbf{T}^n s dimenzí k . (Pro $q = 2$ se dolní index q u označení kódu zpravidla vynechává.) Prvky kódu $\mathbf{K} \subseteq \mathbf{T}^n$ nazýváme *kódová slova*, ostatní vektory z \mathbf{T}^n jsou *nekódová slova*. Slovo, jehož všechny složky jsou nulové, se nazývá *nulové slovo*. Každý lineární kód obsahuje nulové slovo.

Číslo n je délka kódového slova, neboli *délka kódu*. Číslo k udává *počet informačních znaků* kódu, rozdíl $n - k$ je *počet kontrolních znaků* kódu.

Duální kód \mathbf{K}^\perp k danému lineárnímu $(n, k)_q$ -kódu \mathbf{K} se skládá právě ze všech slov délky n , která jsou ortogonální ke každému kódovému slovu kódu \mathbf{K} . Duální kód k lineárnímu $(n, k)_q$ -kódu \mathbf{K} je lineární $(n, n - k)_q$ -kód; duálním kódem ke kódu \mathbf{K}^\perp je opět kód \mathbf{K} .

2.2 Cyklické a CRC kódy

Velmi významnou skupinou lineárních kódů jsou cyklické kódy a jejich zobecnění: CRC kódy a kvazicyklické kódy. Patří mezi ně mnoho nejpoužívanějších typů kódů: sudá parita, BCH kódy, Reedovy-Solomonovy kódy.

Cyklické kódy jsou lineární kódy, které jsou uzavřené na cyklický posun: lineární kód se nazývá *cyklický*, jestliže pro každé kódové slovo $v_0v_1 \dots v_{n-1}$ je také slovo $v_{n-1}v_0v_1 \dots v_{n-2}$ kódovým slovem.

Kódová slova se zapisují ve tvaru formálních polynomů stupně menšího než n , což jsou prvky okruhu polynomů $\mathbf{T}[x^n - 1] = \mathbf{T}^{(n)}$. Součin v prostoru $\mathbf{T}^{(n)}$ je definován jako zbytek po dělení součinu polynomů polynomem $x^n - 1$. Cyklický posun pak v okruhu $\mathbf{T}^{(n)}$ odpovídá násobení polynomem x .

Každý netriviální cyklický $(n, k)_q$ -kód \mathbf{K} obsahuje jediný normovaný polynom $\mathbf{g}(x)$ stupně $n - k$. Polynom $\mathbf{g}(x)$ se nazývá *generující polynom* kódu \mathbf{K} ; kód \mathbf{K} se skládá právě ze všech násobků polynomu $\mathbf{g}(x)$ v $\mathbf{T}^{(n)}$.

Zobecněním cyklických kódů jsou CRC kódy a kvazicyklické kódy. *CRC kód* je kód vzniklý zkrácením cyklického kódu. Kód je *kvazicyklický*, pokud pro nějaké celé číslo m platí, že cyklickým posunutím kódového slova o m pozic vznikne opět kódové slovo.

2.3 Detekce chyb

Při použití detekčních nebo samoopravných kódů je podstatná jejich schopnost odhalovat, případně opravovat chyby, které vzniknou při přenosu informace nebo při jejím uložení na paměťovém médiu.

Předpokládáme, že je vysláno kódové slovo u a přijato nějaké slovo v z množiny \mathbb{T}^n . Pokud je přijato nekódové slovo, byla *objevena chyba*. Pokud je přijaté slovo kódové, pak buď k chybě nedošlo (slovo nebylo během přenosu změněno), nebo chyba nebyla objevena (místo vyslaného slova bylo přijato jiné kódové slovo). Chyba je *t-násobná*, jestliže počet chybných znaků ve slově je nejvýše t . Kód *objevuje t-násobné chyby*, jestliže při vyslání kódového slova a vzniku t -násobné chyby je přijaté slovo nekódové.

Při některých aplikacích se chyby zpravidla nevyskytují náhodně, ale ve shlucích; to znamená, že se často vyskytnou více chybných znaků blízko sebe. *Shlukem chyb délky b* se nazývá chybové slovo, jehož všechny nenulové složky tvoří část, ležící mezi b po sobě následujícími znaky. Lineární kód *objevuje shlukové chyby délky b* , jestliže žádné nenulové kódové slovo není shlukem chyb délky b .

K detekci shlukových chyb jsou vhodné cyklické kódy a kódy od nich odvozené (CRC kódy, kvazicyklické kódy). Cyklický $(n, k)_q$ -kód objevuje všechny shluky délky $n - k$, ale neobjeví všechny shluky chyb délky $n - k + 1$.

2.4 Minimální vzdálenost kódu

Základním parametrem detekčních schopností kódu vzhledem k náhodným chybám je minimální vzdálenost kódu.

Definujeme *Hammingovu vzdálenost* $d(u, v)$ slov u a v jako počet znaků, ve kterých se slova u a v od sebe liší. *Hammingova váha* $w_H(u)$ slova u je pak jeho Hammingova vzdálenost od nulového slova.

Minimální vzdálenost d kódu nazveme nejmenší Hammingovu vzdálenost dvou různých kódových slov. Lineární kód minimální vzdálenosti d objevuje t -násobné chyby pro všechna $t < d$, ale neobjeví všechny d -násobné chyby. Může ovšem odhalovat určitou (často velmi velkou) podmnožinu těchto chyb. Proto se i kódy se shodnou délkou, dimenzí a minimální vzdáleností mohou velmi výrazně lišit v pravděpodobnosti, že nastane nedetekovaná chyba.

2.5 Pravděpodobnost nedetekované chyby

Kromě triviálního případu, kdy je množina kódových slov jednoprvková, nelze nikdy zcela vyloučit, že při přenosu dojde k chybě, která nebude odhalena.

Pravděpodobnost nedetekované (neodhalené) chyby P_{ud} je pravděpodobnost, že bylo vysláno kódové slovo a přijato jiné kódové slovo. Pro lineární kód je pravděpodobnost nedetekované chyby rovna pravděpodobnosti, že při vyslání nulového slova je přijato nenulové kódové slovo.

Necht' je \mathbf{K} lineární $(n, k)_q$ -kód. Definujeme *váhový vektor* kódu \mathbf{K} jako celočíselný vektor $A = (A_0, A_1, \dots, A_n)$, kde A_i je počet slov kódu \mathbf{K} , která mají Hammingovu váhu i . Protože počet všech slov váhy i je roven $\binom{n}{i}(q-1)^i$, je pravděpodobnost, že přijaté slovo Hammingovy váhy i je kódovým slovem, rovna $A_i / \binom{n}{i}(q-1)^i$. Pravděpodobnost nedetekované chyby P_{ud} pak vypočteme podle vzorce

$$P_{ud} = \sum_{i=1}^n P_i \frac{A_i}{\binom{n}{i}(q-1)^i}, \quad (1)$$

kde P_i je pravděpodobnost, že ve slově je chybně právě i znaků. Tato pravděpodobnost P_i závisí na vlastnostech použitého sdělovacího kanálu.

2.6 Sdělovací kanály

Jako pravděpodobnostní model přenosu informace slouží *sdělovací kanál*. Nejčastěji je používán *binární symetrický kanál*, ve zkratce BSC. Je to jednoduchý model založený na bitovém přenosu — vstupní i výstupní abeceda je dvouprvková množina $\{0, 1\}$, pravděpodobnost chyby v jednom znaku je nezávislá na ostatních znacích (kanál bez paměti) a pravděpodobnost chybného přenosu p_e je stejná pro oba případy (záměna nuly za jedničku i jedničky za nulu — symetrický kanál).

Pro binární lineární (n, k) -kód v binárním symetrickém kanálu je pravděpodobnost, že ve slově délky n vznikne chyba právě v i znacích, rovna

$$P_i = \binom{n}{i} p_e^i (1 - p_e)^{n-i}. \quad (2)$$

Pro pravděpodobnost nedetekované chyby binárního lineárního (n, k) -kódu s Hammingovou vzdáleností d a váhovým vektorem $A = (A_0, A_1, \dots, A_n)$ v BSC dostaneme dosazením (2) do (1) následující vzorec:

$$P_{ud}(p_e) = \sum_{i=d}^n A_i p_e^i (1 - p_e)^{n-i}. \quad (3)$$

Zobecněním BSC je q -*nární symetrický kanál* (označovaný zkratkou QSC), jehož vstupní i výstupní abeceda je těleso $\mathbf{GF}(q)$. Pravděpodobnosti chybného přenosu jednotlivých po sobě následujících znaků jsou vzájemně nezávislé a pravděpodobnost zkreslení jednoho znaku na jiný je stejná pro všechny dvojice vzájemně různých znaků z $\mathbf{GF}(q)$.

Označíme-li ε pravděpodobnost chybného přenosu jednoho znaku, je pravděpodobnost, že ve slově délky n vznikne chyba právě v i znacích, rovna

$$P_i = \binom{n}{i} \varepsilon^i (1 - \varepsilon)^{n-i} \quad (4)$$

a pro pravděpodobnost nedetekované chyby lineárního $(n, k)_q$ -kódu v QSC vznikne dosazením (4) do (1) následující vzorec:

$$P_{ud}(\varepsilon) = \sum_{i=d}^n A_i \left(\frac{\varepsilon}{q-1} \right)^i (1 - \varepsilon)^{n-i}. \quad (5)$$

Nebinární kódy nad tělesem $\mathbf{GF}(2^m)$ je přirozené studovat v modelu QSC s $q = 2^m$. Je ale také možné interpretovat je binárně a zkoumat jejich vlastnosti — například pravděpodobnost nedetekované chyby — v BSC. Ukazuje se, že kód, který je 'dobrý', resp. 'správný' (definice viz 2.7) v 2^m -árním symetrickém kanálu, nemusí mít stejnou vlastnost i v kanálu binárním (viz například Kárná [8]). Například Reedovy-Solomonovy kódy jsou v QSC 'správné', ale jejich binární interpretace v BSC

většinou nejsou 'dobré' a maximální hodnota pravděpodobnosti nedetekované chyby překračuje hodnotu 2^{n-k} často i o několik řádů.

Pravděpodobnostní modely přenosových kanálů jsou spíše abstraktním měřítkem struktury lineárního kódu, než matematickým modelem, který by popisoval nějaký reálný přenosový systém. Některé reálně používané kanály se mohou více blížit BSC, jiné QSC a bez apriorní znalosti použitého přenosového kanálu není žádný zřejmý důvod preferovat kterýkoliv z těchto modelů před druhým.

2.7 Dobrý a správný kód

Pro $\varepsilon = (q-1)/q$ je pravděpodobnost nedetekované chyby každého lineárního $(n, k)_q$ -kódu rovna $(q^k - 1)/q^n$; pro ε z okolí bodu $(q-1)/q$ platí odhad $P_{ud}(\varepsilon) < q^{k-n}$. Protože naším cílem je najít horní odhad pravděpodobnosti nedetekované chyby na intervalu $[0, (q-1)/q]$, mají pro nás zvláštní význam ty kódy, pro které platí odhad $P_{ud}(\varepsilon) < q^{k-n}$ na co největším okolí bodu $(q-1)/q$. Proto zavádíme následující definice:

- Lineární $(n, k)_q$ -kód je 'dobrý' (*good*), jestliže platí $P_{ud}(\varepsilon) < q^{k-n}$ pro $\varepsilon \in [0, (q-1)/q]$.
- Lineární $(n, k)_q$ -kód je 'správný' (*proper*), jestliže je funkce $P_{ud}(\varepsilon)$ monotónně rostoucí na intervalu $[0, (q-1)/q]$.

'Správný' kód je vždy 'dobrý', nikoliv však obráceně. Výhodou 'správného' kódu je, že na kanálech s menší chybovostí má menší pravděpodobnost selhání, tj. na lepších kanálech dává lepší výsledky než na horších.

3 Metodika návrhu a kvantitativního hodnocení detekčních kódů

V následujícím textu shrnuji základní body vypracované metodiky.

3.1 Specifikace požadavků

Při použití detekčních kódů v bezpečnostně relevantních aplikacích je požadováno prokazatelné splnění požadavků na jejich spolehlivost. Přesná specifikace těchto požadavků je proto prvním krokem při návrhu detekčních kódů nebo jejich systému.

Analýza požadavků se provádí i v případě hodnocení již navrženého nebo používaného kódu, aby bylo možné ověřit, že kód těmto požadavkům vyhovuje. Mezi základní specifikované požadavky patří:

- Podmínky, pro které je kód určen (architektura systému, struktura přenosového protokolu, typ a kapacita přenosového kanálu atd.) a účel, který má kód plnit (zabezpečení komunikace, kontrola integrity souborů a podobně).
- Je-li kód určen pro zajištění bezpečnostně relevantních funkcí.

- Délka kódu (kódového slova). Cyklické kódy jsou zpravidla používány ve své zkrácené formě (jako CRC kódy) a kód pak není definován pouze svým generujícím polynomem, ale též délkou kódového slova. V praxi se však doposud zanedbává skutečnost, že zkrácení kódu má obvykle negativní vliv na jeho detekční schopnosti. Výpočty přitom ukazují, že mnohé běžně užívané kódy mají pro zprávy malé délky špatnou schopnost detekce, zatímco pro dlouhé zprávy dosahují dobrých detekčních vlastností. U dalších kódů je pak situace obrácená, mají dobré výsledky pro krátké zprávy a špatné pro dlouhé zprávy. Proto je nezbytné již v počátku určit délku kódového slova, resp. rozmezí délek zpráv, pro které má být kód určen.
- Počet redundantních znaků. Je to jeden ze základních parametrů kódu; kódy s malým počtem redundantních znaků nemohou mít dobré detekční schopnosti. Mnohdy bývá z technických důvodů požadováno, aby byl počet redundantních bitů u binárního kódu dělitelný osmi.
- Minimální vzdálenost kódu, resp. násobnost chyb, které má být kód schopen detekovat, případně opravit.
- Požadovaná schopnost detekovat vybrané chybové vzorce. Protože se předpokládá, že určité vybrané typy chyb se vyskytují častěji, požaduje se, aby kód tyto chyby detekoval. Mezi tyto typické chyby patří především všechny bity v logické jedničce, všechny bity v logické nule a negace zprávy.
- Požadovaná schopnost detekce shlukových chyb.
- Maximální přípustná pravděpodobnost nedetekované chyby. Tato hodnota je pro bezpečnostně relevantní aplikace odvozena výpočtem z požadovaných bezpečnostních charakteristik systému. Pokud navrhovaný kód nebude binární, je potřeba určit, zda bude použit model QSC, nebo se kód interpretuje jako binární a bude se provádět analýza v BSC. To částečně závisí i na případných normativních požadavcích — například pro datové komunikace v drážních zařízeních vyžaduje v současné době evropská norma [5] výpočet v BSC; v blízké době se však očekává její změna. Analýzy prováděné v modelu QSC se ukazují být rychlejší a snazší než v modelu BSC.
- Zda je požadován 'dobrý', případně 'správný' kód. Podobně jako u předchozí položky je nutné volit mezi modely BSC a QSC.

Dále je vhodné vypracovat slovník termínů, pojmů a zkratk, který budou mít k dispozici všichni členové vývojového týmu, aby se co nejvíce omezila možnost nedorozumění, způsobenému rozdílnou terminologií.

3.2 Návrh kódu

Pokud je úkolem návrh kódu nebo systému kódů, postupuje se podle následujících zásad:

- Vhodné jsou lineární kódy, buď binární nebo nad tělesem $\mathbf{GF}(2^p)$.

- Redundance kódu má být přiměřená požadavkům. Pro zajištění větší úrovně detekce chyb se často intuitivně volí kódy s velkým počtem redundantních znaků, aniž by se blíže zkoumaly jejich vlastnosti. Kód s větším počtem redundantních znaků ale nemusí mít automaticky lepší detekční vlastnosti než dobře zvolený kód s menší redundancí. Velký počet redundantních znaků přináší značné technické problémy při výpočtu pravděpodobnosti nedetekované chyby a studiu jejího průběhu.
- Přednost dáváme kódům cyklickým, kvazicyklickým nebo CRC kódům. Důvodem jsou jednak jejich dobré detekční schopnosti, jednak snadná a efektivní implementace kódování a dekodování. Kromě toho i kvantitativní výpočet detekčních schopností je pro tyto typy kódů jednodušší.
- Generující polynom CRC kódu volíme nesoudělný s generujícími polynomy standardizovaných nebo jiných běžně používaných CRC kódů a s generujícími polynomy ostatních kódů v systému, aby byla alespoň částečně zajištěna jejich funkční nezávislost.
- Volíme kódy s větší vahou generujícího polynomu. Mají lepší detekční schopnosti, protože minimální vzdálenost kódu nemůže být větší, než je počet nenulových koeficientů generujícího polynomu.

Má-li se provést analýza a kvantitativní hodnocení již navrženého nebo používaného kódu, identifikuje se v této fázi jeho typ (lineární, CRC, Hammingův, BCH a pod.).

3.3 Výpočet minimální vzdálenosti kódu

Minimální vzdálenost kódu je rozhodující pro určení, kolikanásobně chyby je kód schopen odhalit, případně opravit.

3.3.1 Minimální vzdálenost daná konstrukcí kódu

V některých případech je již konstrukcí kódu dána buď jeho minimální vzdálenost, nebo její dolní odhad. Jedná se například o Hammingovy kódy, BCH kódy, MDS kódy a kódy vzniklé zkrácením nebo binární interpretací kódu, jehož minimální vzdálenost je známa.

3.3.2 Přímý výpočet minimální vzdálenosti kódu

Pro výpočet minimální vzdálenosti lineárního kódu je možné buď

- spočítat Hammingovu váhu všech kódových slov a najít její minimum, nebo
- postupně generovat všechna slova z \mathbf{T}^n , která mají Hammingovu váhu 1, 2, 3, ... a zjišťovat, zda jsou slovy kódovými.

3.3.3 Určení minimální vzdálenosti z váhového vektoru kódu

Pokud je hodnocený kód bezpečnostně relevantní, bude pro kvantitativní průkaz jeho bezpečnosti nutné vypočítat jeho váhový vektor. Z něj je pak zřejmá i minimální vzdálenost kódu.

3.4 Výpočet váhového vektoru kódu

Váhový vektor kódu dává přesnou představu o rozložení vah kódových slov. Především však slouží k výpočtu pravděpodobnosti neodhalené chyby kódu v binárním (resp. q -nárním) symetrickém kanálu pomocí vzorce (3), resp. (5).

Přirozeným postupem, jak jej spočítat, je vygenerování všech nenulových kódových slov a vypočítání jejich vah. Protože ale lineární $(n, k)_q$ -kód má q^k kódových slov, je pro větší hodnotu k takový výpočet obtížně uskutečnitelný. Proto se využívá nepřímého výpočtu, jehož náročnost je úměrná q^{n-k} (redundance kódu $n-k$ je obvykle mnohem menší, než počet informačních znaků k).

3.4.1 Předem známý váhový vektor

Pro některé kódy je váhový vektor dán jejich konstrukcí. Jsou to například sudá parita, opakovací kód, Hammingovy kódy, některé BCH kódy a všechny MDS kódy.

3.4.2 Přímý výpočet váhového vektoru

Pro kódy s velmi malým počtem informačních znaků k je vhodný přímý výpočet váhového vektoru — vygeneruje se všech q^k slov kódu a spočítají se jejich váhy.

3.4.3 Využití identity MacWilliamsové

Pro $k > n/2$ je vhodnější nepřímý postup. Definujeme *váhový polynom kódu* jako formální polynom

$$\mathbf{pw}(x, n, \mathbf{K}) = \sum_{i=0}^n A_i x^i. \quad (6)$$

Váhový polynom kódu \mathbf{K} je svázán s váhovým polynomem duálního kódu \mathbf{K}^\perp identitou MacWilliamsové, která má tvar

$$q^k \mathbf{pw}(x, n, \mathbf{K}^\perp) = (1 + (q-1)x)^n \mathbf{pw}\left(\frac{1-x}{1+(q-1)x}, n, \mathbf{K}\right). \quad (7)$$

Duální kód obsahuje pro velké k podstatně méně kódových slov, než kód původní, a proto je výpočet jeho váhového vektoru rychlejší.

Váhový vektor duálního kódu je možné vypočítat přímo vygenerováním všech q^{n-k} jeho slov. Pro CRC kód může být místo toho použita metoda dvou LFSR.

3.4.4 Metoda dvou LFSR

Metoda dvou LFSR je určena pro rychlý výpočet váhového vektoru kódu duálního k binárnímu CRC kódu s generujícím polynomem $\mathbf{g}(x)$. Využívá skutečnosti, že kódová slova takového kódu mohou být bit po bitu vypočítána pomocí Fibonnacciho LFSR (lineární registr se zpětnou vazbou) s generujícím polynomem $\mathbf{g}(x)$ (viz např. Manganiello [10]). Paralelní činností druhého (identického) LFSR, který zahájí svoji činnost teprve po vygenerování celého prvního kódového slova, se zajistí, že váha každého dalšího kódového slova bude vypočtena v jediném kroku.

Všechna nenulová kódová slova jsou z jednoho počátečního stavu vygenerována pouze tehdy, pokud je generující polynom $\mathbf{g}(x)$ primitivní. Pokud $\mathbf{g}(x)$ není primitivní, jsou nenulové stavy LFSR rozděleny do více tříd ekvivalence — cyklických cosetů. Identifikace těchto tříd ekvivalence, resp. jejich reprezentantů, sestává z následujících kroků:

- faktorizace generujícího polynomu na součin ireducibilních polynomů,
- nalezení tříd ekvivalence pro každý z těchto ireducibilních faktorů,
- získání uspořádaných J -tic polynomů zkombinováním nalezených tříd ekvivalence všemi možnými způsoby, a
- získání reprezentantů tříd ekvivalence pro generující polynom $\mathbf{g}(x)$ pomocí jistého izomorfismu.

Blíže například články Castagnoli *et al.* [1]; Manganiello [10]; Kárná [9].

Výhodou metody dvou LFSR je především lepší využití vyrovnávací paměti (cache), což se pro velké délky kódových slov projeví výrazným zrychlením výpočtu oproti postupu pomocí generující matice. Tato metoda také umožňuje provést výpočet na hradlových polích (FPGA) namísto běžného PC a je dobře paralelizovatelná.

3.5 Dobrý, správný kód

Při posuzování, jedná-li se o 'dobrý', resp. 'správný' kód, se snažíme upřednostnit metody, pro které jsou postačující celočíselné výpočty (viz 3.5.1 nebo 3.5.2). Ty mohou být — přinejmenším teoreticky — provedeny zcela přesně a proto jsou výsledky jimi dosažené maximálně věrohodné. Teprve když žádný z těchto postupů nevede k rozhodnutí, uchylujeme se k numerickým metodám. Spolehlivost jejich výsledků je totiž již z principu omezená.

3.5.1 Určení vlastností bez výpočtu váhového vektoru

Ve velmi příznivých případech je možné určit, zda lineární $(n, k)_q$ -kód \mathbf{K} je (není) 'dobrý', resp. 'správný', bez výpočtu jeho váhového vektoru, jen na základě znalosti jeho minimální vzdálenosti d , případně minimální vzdálenosti d^\perp jeho duálu či počtu kódových slov minimální váhy $A_d^{\mathbf{K}}$.

Nejčastěji je možné využít následující pravidla (viz Nikolova [11]; Dodunekova, Weng [4]):

- Jestliže je minimální vzdálenost kódu d větší, než $n/2$, je kód \mathbf{K} 'správný'.
- Je-li $A_d^{\mathbf{K}} \left(\frac{d}{n}\right)^d \left(1 - \frac{d}{n}\right)^{n-d} > q^{k-n}$, pak kód \mathbf{K} není 'dobrý'.
- Je-li $d^\perp \geq \frac{n(q-1)+1}{q}$, je kód \mathbf{K} 'správný'.
- Pokud je $\frac{n(q-1)+2}{q+1} \leq d^\perp \leq \frac{n(q-1)+1}{q}$, a navíc $\frac{n(q-1)-d^\perp q+1}{n(q-1)-d^\perp q+1+\frac{d^\perp-1}{q-1}} \leq \frac{d}{n}$, pak je kód \mathbf{K} 'správný'.

'Správné' jsou též některé 'optimální' třídy kódů a to perfektní kódy a MDS kódy (viz Dodunekova, Dodunekov [2]; Kasami, Lin [6]). (*Perfektní kódy* mají pro danou délku a minimální vzdálenost nejvyšší možný počet slov; *MDS kódy* mají pro danou délku a dimenzi největší možnou minimální vzdálenost.)

3.5.2 Využití binomických momentů

Nelze-li využít předchozích postupů, je nutné nejprve vypočítat váhový vektor duálního kódu a zjistit minimální vzdálenost původního kódu i jeho duálu.

Předpokládejme lineární $(n, k)_q$ -kód \mathbf{K} s minimální vzdáleností d . Kód \mathbf{K}^\perp , který je k němu duální, nechť má váhový vektor $B = (B_0, B_1, \dots, B_n)$ a minimální vzdálenost d^\perp . Pokud je $d + d^\perp > n$, jedná se o MDS kód, který je 'správný'.

Platí-li $d + d^\perp \leq n$, vypočteme *rozšířené binomické momenty* duálního kódu \mathbf{K}^\perp podle těchto vzorců:

- $B_l^* = 0$ pro $l < d^\perp$,
- $B_l^* = q^{l+k} - 1$ pro $l = n - d + 1, \dots, n$,
- $B_l^* = \sum_{i=1}^l \frac{l(l-1)\dots(l-i+1)}{n(n-1)\dots(n-i+1)} B_i$ pro $l = d^\perp, \dots, n - d$.

Pak můžeme využít následující kritéria (viz Dodunekova [3]; Dodunekova, Dodunekov [2]):

- Je-li $B_{n-l}^* \geq B_{n-l+1}^* - q^{n-k-l}(q-1)$ pro všechna $l = d + 1, \dots, n - d^\perp + 1$, je kód \mathbf{K} 'správný'.
- Je-li $q^{-n+l} B_{n-l}^* \leq q^{-k} - q^{n-k+l}$ pro všechna $l = d, \dots, n - d^\perp$, pak je kód \mathbf{K} 'dobrý'.

Podobný postup je možné použít i v případě, že jsme vypočítali váhový vektor původního kódu namísto váhového vektoru kódu duálního.

3.5.3 Numerické řešení

Není-li možné určit, zda je kód 'dobrý', případně 'správný', výše uvedenými postupy, nezbyvá než najít maxima funkce P_{ud} numericky. K tomu účelu se v praxi jeví jako nejvhodnější Newtonova metoda; ani ta však není zcela ideální, mimo jiné proto, že funkce P_{ud} má v nule vícenásobný kořen. Proto je vhodné, podaří-li se pomocí vhodných kritérií zmenšit interval, na kterém se budou maxima funkce hledat.

Pro lineární $(n, k)_q$ -kód \mathbf{K} s minimální vzdáleností d , jehož duální kód \mathbf{K}^\perp má minimální vzdálenost d^\perp , platí následující tvrzení (viz Dodunekova, Weng [4]; Nikolova [11]):

- Funkce $P_{ud}(\varepsilon)$ je rostoucí na intervalu $[0, \frac{d}{n}]$.
- Jestliže $\frac{n(q-1)+2}{q+1} \leq d^\perp \leq \frac{n(q-1)+1}{q}$, pak je funkce $P_{ud}(\varepsilon)$ rostoucí na intervalu $\left[\frac{n(q-1)-d^\perp q+1}{n(q-1)-d^\perp q+1+\frac{d^\perp-1}{q-1}}, \frac{q-1}{q} \right]$.

Protože funkce P_{ud} není zcela vhodná pro použití Newtonovy metody, nehledá se přímo kořen její derivace, ale funkce vzniklé z derivace vhodnými transformacemi (substituce, vydělení kladným výrazem a pod.), které zachovávají znaménko funkce, její první i druhé derivace. Přitom je za určitých okolností nutné Newtonovu metodu ještě kombinovat s metodou půlení intervalu. Technika výpočtu je poměrně komplikovaná; stručný nástin postupu lze nalézt v práci [7] autorů Klapky a Harlenderové.

4 Závěr

Uvedená metodika byla ověřována na velkém množství kódů různých typů. Byl pomocí ní proveden kvantitativní průkaz bezpečnosti řady kódů, které jsou nyní využívány v různých železničních zabezpečovacích zařízeních, produkovaných firmou AŽD Praha, s.r.o., a kódů, navržených k použití v rámci evropského projektu SafeDMI. Rovněž byly analyzovány detekční vlastnosti některých často používaných standardizovaných kódů. Metodika se ukázala být funkční a v praxi dobře použitelná.

Největší problém při popisu kvantitativních bezpečnostních parametrů kódu činí určení maximální hodnoty pravděpodobnosti nedetekované chyby kódu v binárním symetrickém kanálu, které je výpočetně velmi náročné. Předložená metodika zahrnuje postupy umožňující zrychlení potřebných výpočtů, a to jak pomocí zlepšené organizace výpočtu, tak díky využití obecných matematických zákonitostí.

Literatura

- [1] Guy Castagnoli, Stefan Bräuer, and Martin Herrmann. Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits. *IEEE Transactions on Communications*, 41(6):883–892, 06 1993.
- [2] R. Dodunekova and S. M. Dodunekov. Sufficient Conditions for Good and Proper Error Detecting Codes. *IEEE Transactions on Information Theory*, 43(6):2023–2026, 11 1997.
- [3] Rossitza Dodunekova. On the Binomial Moments of Linear Codes and Undetected Error Probability. *Preprint*, 2002:49:14 p., 2002.
- [4] Rossitza Dodunekova and Li Weng. Sufficient Conditions for Interval Properness of Linear Error Detecting Codes. *Math. Balkanica (NS)*, 3–4(21):245–258, 2007.
- [5] EN 50159-1 Railway Applications – Communication, Signalling and Processing Systems Part 1: Safety Related Communication in Closed Transmission Systems, 2002.
- [6] Tadao Kasami and Shu Lin. On the Probability of Undetected Error for the Maximum Distance Separable Codes. *IEEE Transactions on Communications*, COM-32(9):998–1006, 9 1984.
- [7] Štěpán Klapka and Magdaléna Harlenderová. How Good Detection Codes in Binary Symmetrical Channel Are. *To appear*.

- [8] Lucie Kárná. Chování nebinárního kódu v q -nárním symetrickém kanálu. *Sborník semináře MIS 2009, Josefův Důl*.
- [9] Lucie Kárná. Metodika kvantitativního hodnocení detekčních kódů. *Disertační práce*, 2009.
- [10] Felice Manganiello. Computation of the Weight Distribution of CRC Codes. *Applicable Algebra in Engineering, Communication and Computing*, 19(4):349–363, 2008.
- [11] E. Nikolova. A Sufficient Condition for Properness of a Linear Error-Detecting Code and its Dual. *Mathematics and Mathematical Education. Proc. 34th Spring Conf. of the Union of Bulgarian Mathematicians*, pages 136–139, 4 2005.

Vizualizácia modelu v systéme Bobox

Jana Katreniaková

Jiří Dokulil

KI FMFI UK, Bratislava

KSI MFF UK, Praha

katreniakova@dcs.fmph.uniba.sk

dokulil@ksi.mff.cuni.cz

Abstrakt. Cieľom projektu Bobox je vytvoriť prostredie pre paralelné výpočty riadené tokom dát. Základnou myšlienkou je prepojenie jednoduchých výpočtových prvkov (krabičiek) pomocou jednotného systému prepojovacích prvkov (via) a tak vytvoriť nelineárny prúd pre spracovanie dát, ktorý môže bežať paralelne alebo aj distribuovane. V článku sa venujeme vizualizácii modelu Boboxu, ktorý sa dá reprezentovať orientovaným grafom. Na jeho kreslenie použijeme algoritmus s využitím viditeľnostnej reprezentácie grafu. Modifikácia tohoto algoritmu nám umožní zobrazovať aj krabičky a via, ktoré majú rôzne rozmery.

Kľúčové slová: Bobox, kreslenie grafov, metóda s využitím viditeľnosti

1 Úvod

V dnešnej dobe majú už takmer všetky procesory viac ako jedno výpočtové jadro, nehovoriac o tom, že server bežne obsahuje niekoľko takých procesorov. Paralelné výpočty sa preto stávajú dôležitým smerom vývoja softwaru.

Cieľom projektu Bobox [3, 4] je poskytnúť platformu pre niektoré typy takých výpočtov, konkrétne pre databázové výpočty. Dôraz sa kladie na prípady, kedy systém vyhodnocuje menšie množstvo zložitých dotazov a preto by paralelizácia na úrovni celých dotazov bola nepostačujúca. Základnou myšlienkou je prepojiť väčšie množstvo relatívne jednoduchých výpočtových komponentov (tzv. *krabičiek*) do nelineárnej pipeline pomocou prepojovacích prvkov – *via*. Každá krabička má niekoľko vstupných a niekoľko výstupných *via*, čím vznikne orientovaný graf, v ktorom sa na každej ceste pravidelne striedajú krabičky a *via*.

Používateľom zadaný dotaz je najskôr preložený do štruktúry krabičiek a *via* tzv. *modelu*. Z tohoto vzoru sa neskôr vytvorí tzv. *inštancia modelu* – skutočné krabičky a *via*, ktoré vykonávajú reálny výpočet.

Keďže model aj inštanacie sú pre každý dotaz vytvárané prekladačom, je potrebné mať nástroj na preskúmanie vytvorených modelov. V súčasnosti inštanacie presne zodpovedajú modelu. Keďže predpokladáme, že inštanacie budú môcť počas chodu narastať pridávaním krabičiek a *via*, má zmysel aj vizualizácia inštancií. Graf zodpovedajúci inštancii má však podobné vlastnosti ako samotný model a preto pri jeho vizualizácii môžeme použiť tú istú metódu.

V súčasnosti sú grafy zodpovedajúce modelom v podstate acyklické, prípadne len

s malým počtom hrán, ktoré acyklickosť porušujú. Preto sme sa rozhodli vizualizovať tieto modely pomocou metódy s využitím viditeľnosti, ktorý sme upravili tak, aby vrcholy grafu nereprezentoval ako body ale ako reálne obdĺžniky s dopredu známymi a nezanedbateľnými rozmermi.

V článku najskôr popíšeme systém Bobox a jeho model, ktorý chceme vizualizovať. Samotnú vizualizáciu sme založili na kreslení grafov. Preto v ďalšej časti zhrnieme aspoň základy z tejto oblasti, ktoré sú potrebné pre pochopenie algoritmu vizualizácie. Samotná vizualizácia v časti 4 je založená na metóde s využitím viditeľnosti, resp. na jej modifikácii pre obdĺžnikové vrcholy. Na záver zhrnieme výsledky článku a načrtneme niektoré otvorené problémy.

2 Bobox

Hlavnou úlohou Boboxu je sprostredkovávanie komunikácie medzi krabičkami a paralelné spúšťanie kódu krabičiek. Komunikácia medzi krabičkami v inštancii modelu prebieha iba pomocou posielania dát v dávkach označovaných ako *obálky*. Krabička môže poslať obálku s dátami niektorej via, ktorá je pripojená na jej výstup. Via potom prepošle obálku všetkým krabičkám, ktoré na ňu majú napojené svoje vstupy. Z logického hľadiska sú krabičky a via prepojené orientovanými *hranami*, v samotnej implementácii však tieto hrany nemajú reálny ekvivalent a zodpovedajú im časti krabičiek a via.

Okrem obálok s reálnymi dátami existuje špeciálny typ obálky tzv. *otrávená pilulka*, ktorá označuje koniec dát. V prípade, že krabička pošle na niektorý svoj výstup otrávenú pilulku, znamená to, že na tento výstup už nepošle žiadne ďalšie dáta. Táto skutočnosť sa využíva na zahájenie a ukončenie výpočtu. Každý model (a teda aj inštancia) musí obsahovať dve špeciálne krabičky: inicializačnú a terminačnú. Pre každý orientovaný sled (cesta pripúšťajúca opakovanie hrán a vrcholov) v modeli musí platiť, že začína inicializačnou krabičkou a končí terminačnou krabičkou. Na začiatku výpočtu pošle inicializačná krabička na svoj výstup otrávenú pilulku – táto krabička žiadne dáta negeneruje a preto môže rovno na začiatku vyhlásiť, že ďalšie dáta už posielat nebude. To slúži ako signál pre ďalšie krabičky, aby zahájily výpočet. Terminačná krabička ukončí výpočet keď má na všetkých svojich vstupoch otrávenú pilulku. Vtedy môže byť inštancia modelu odstránená.

Počas výpočtu sa významné udalosti (vznik obálky, spustenie krabičky, ...) zaznamenávajú a na ich prehliadanie sa používa špeciálna aplikácia, ktorá ukazuje kedy a v ktorej časti inštancie modelu sa udalosť udiala spolu s časovou osou výpočtu. Keďže prepojenie krabičiek a via je v inštancii modelu obvykle veľmi zložité, je potrebné aby bola táto inštancia modelu zobrazená graficky a prehľadne.

3 Vizualizácia grafov

Pri vizualizácii modelu Boboxu potrebujeme zobrazit krabičky a via, ktoré budeme v ďalšom súhrne nazývať vrcholmi. Tieto sú spojené pomocou hrán. Keďže budeme využívať skutočnosť, že model chápeme ako graf, zhrnieme naprv niekoľko základných pojmov z oblasti grafov a ich kreslenia.

3.1 Grafy

Graf $G = (V, E)$ je matematická štruktúra, ktorá sa skladá z konečnej množiny vrcholov V a konečnej množiny hrán E ; každá hrana je neusporiadaná dvojica vrcholov. Graf, kde E je definovaná ako usporiadaná dvojica rôznych vrcholov nazývame *orientovaný graf* (krátko *digraph*). *Ohodnotený graf* je taký graf, ktorého každá hrana je ohodnotená nejakou číselnou hodnotou – váhou.

Ak $(u, v) \in E$, potom vrcholy u a v grafu G sú *susedné*; u a v sa tiež nazývajú *incidentné* (*susedné*) k hrane (u, v) a naopak; u sa nazýva *susedom* vrcholu v .

Uvažujme hranu (u, v) v orientovanom grafe G . Táto sa nazýva *vstupnou* hranou vrchola v a *výstupnou* hranou vrchola u . *Stupeň* vrchola je počet jeho incidentných hrán. Podobne *vstupný* (*výstupný*) *stupeň* vrchola je počet jeho vstupných (*výstupných*) hrán. *Zdrojom* grafu nazývame vrchol s nulovým vstupným stupňom a *ústím* grafu je vrchol, ktorý má nulový výstupný stupeň. Je zrejmé, že zdrojov a ústí môže mať graf viac alebo aj žiadne.

Topologické číslovanie grafu G je ohodnotenie vrcholov G tak, že číslovanie spĺňa nasledujúcu podmienku: $(u, v) \in E(G) \Rightarrow \text{number}(u) < \text{number}(v)$. *Vážené topologické číslovanie* je také topologické číslovanie na G , že pre každú hranu (u, v) grafu G platí $\text{number}(v) \geq \text{number}(u) + \text{weight}(u, v)$.

3.2 Kreslenie grafov

Nakreslenie grafu je jeho reprezentácia v rovine. Formálne vzaté je kreslenie grafu funkcia, ktorá mapuje vrcholy na rôzne body v rovine a hrany na krivky, ktoré spájajú vrcholy incidentné s touto hranou.

Pre "pekné" nakreslenie grafu existuje viacero kritérií – plocha grafu, počet krížení hrán, počet ohybov hrán, symetria atď., ktoré však podľa [7] nie sú založené na experimentálnych dátach. Preto pre rôzne aplikácie môžu byť zaujímavé rôzne kritériá.

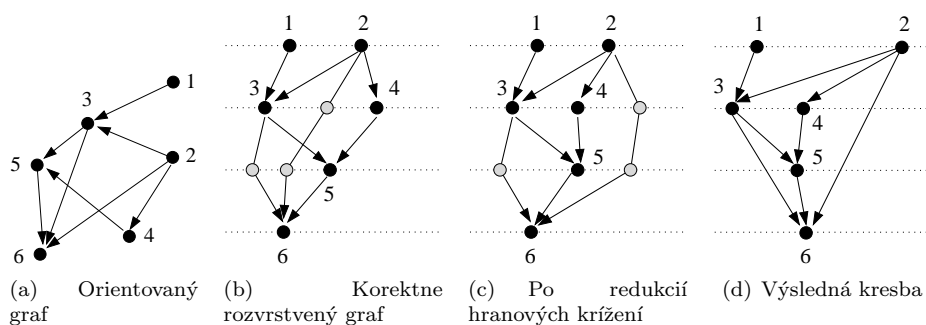
3.3 Metódy kreslenia grafov

Metódy vizualizácie závisia od vlastností vykresľovaných grafov. Pre rôzne triedy sa preto využívajú dost' odlišné prístupy. Pre kreslenie orientovaných acyklických grafov sa využívajú najmä dve kategórie algoritmov.

Algoritmy založené na rozvrstvení vrcholov (tzv. layering-based) sa uskutočňujú v štyroch základných krokoch (obr. 1):

Hierarchizácia: Rozdelenie vrcholov do vrstiev. Vrcholy na jednej vrstve budú mať vo výslednej kresbe rovnakú y -ovú súradnicu. Dôležitou podmienkou je, aby hrany boli orientované rovnako, obvykle smerom k nižším úrovniam. Na hierarchizáciu sa používa očíslovanie vrcholov resp. topologické triedenie.

Normalizácia: Cieľom tohoto kroku je znormalizovať rozvrstvený graf tak, aby hrany viedli iba medzi vrcholmi na susedných úrovniach. Takýto normálny tvar (korektne rozvrstvený graf) dosiahneme jednoducho pridaním falošných vrcholov na úrovne medzi dvoma susednými vrcholmi.



Obrázok 1: Hierarchický prístup kreslenia grafov

Usporiadanie vrcholov: Vrcholy na jednotlivých úrovniach usporiadame tak, aby vzniklo čo najmenej križení hrán. Obvykle sa na usporiadanie vrcholov využívajú heuristiky.

Rozvrhnutie súradníc: Priradí súradnice vrcholom. Pritom zruší falošné vrcholy tým, že ich nahradí ohybmi na hranách. Pri priradovaní súradníc sa snažíme minimalizovať počet takto vzniknutých ohybov a zároveň dosiahnuť čo najväčšie uhly medzi hranami, aby boli hrany dobre viditeľné a rozlíšiteľné.

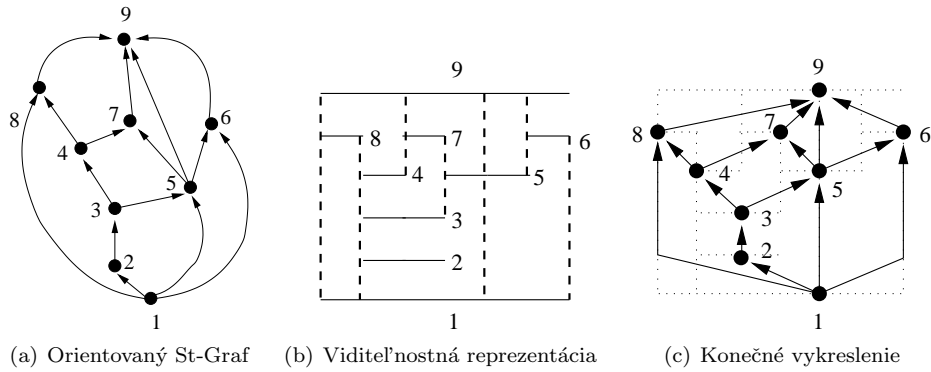
Druhou skupinou sú **mriežkovo-orientované algoritmy** (grid-based). Tieto majú ako svoj vstup planárny st-graf, t.j. planárny acyklický graf s jedným zdrojom a jedným ústím. Vrcholy na rozdiel od vrstvovo-orientovaných algoritmov umiestňujú do mriežky a hrany sú orientované smerom nahor (resp. nadol) – s rastúcou (resp. klesajúcou) y-ovou súradnicou.

Príkladom takéhoto prístupu je napr. algoritmus s využitím viditeľnosti ([2, 1]), ktorý si najprv vytvorí viditeľnostnú reprezentáciu grafu (vid' obr. 2(b)), kde je každý vrchol reprezentovaný ako horizontálna úsečka a každá hrana ako vertikálna úsečka. Jediný prienik vrcholových a hranových úsečiek sú vrchný a spodný bod hranovej úsečky, ktoré zároveň patria vrcholom incidentným s touto hranou. Z viditeľnostnej reprezentácie dostaneme umiestnenie vrcholov tak, že každú vrcholovú úsečku nahradíme jedným bodom (obr. 2(c)). Hrany spájajú incidentné vrcholy s tým, že obsahujú nejakú časť svojej hranovej úsečky.

Nevýhodou takéhoto algoritmu je práve jeho vstup – planárny acyklický graf. Preto je potrebné si neplanárny graf predprípraviť do tohoto tvaru – použiť planarizáciu.

4 Vizualizácia Boboxu

Na vizualizáciu modelu Boboxu potrebujeme nakresliť orientovaný graf, ktorého vrcholy však nie sú body, ale obdĺžniky s rôznymi veľkosťami. Dosiaľ nie je známe, ako presne budú modely vyzerat' (grafové vlastnosti modelov nie sú ešte preskúmané) avšak dá sa predpokladať, že graf bude často acyklický, alebo aspoň nebude mať veľa hrán, ktoré acyklickosť porušujú. Viditeľnostná reprezentácia má síce vo všeobecnosti



Obrázok 2: Metóda s využitím viditeľnosti

oproti algoritmom s rozvrstvením vrcholov horší počet krížení hrán, avšak dosahuje lepšiu plochu. Medzi ďalšie výhody patrí rýchlosť algoritmu.

V ďalšom popíšeme podrobnejšie algoritmus s využitím viditeľnosti a jeho modifikáciu pre kreslenie grafov, ktorých vrcholy sú rôzne veľké obdĺžniky.

4.1 Algoritmy s využitím viditeľnosti

Ako už bolo spomínané, pri algoritme s využitím viditeľnosti je vstupom planárny st-graf. Jedná sa o graf s jedným zdrojom s a jedným ústím t , ktorý je navyše možné nakresliť do roviny s tým, že vrcholy s a t sa nachádzajú na hranici vonkajšej oblasti.

Nech G je planárny st-graf a F je množina jeho oblastí (množina F obsahuje dva prvky pre externú oblasť s^* , ktorý je incidentný s ľavou hranicou G a t^* , ktorý je incidentný s pravou hranicou grafu G). Definujeme $left(e)$ (resp. $right(e)$) ako oblasť, ktorá sa nachádza napravo (resp. naľavo) od hrany e (obr. 3(a)). Oblasti oddeľujúce vstupné a výstupné hrany vrchola sa označujú $left(v)$ resp. $right(v)$ (obr. 3(b)).

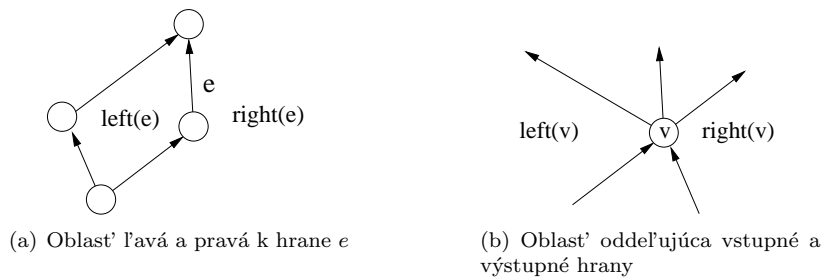
Graf duálny ku grafu G je graf $G^* = (F, E^*)$ (ďalej v texte ho nazývame skráteno *duál* G), taký, že $E^* = \{(f, g) \mid \exists e \in E(G) \setminus (s, t) : f = left(e) \text{ a } g = right(e)\}$ Je zrejmé, že pre každý planárny st-graf je jeho duál tiež planárny st-graf.

Všeobecný algoritmus na nakreslenie grafu (vytvorenie viditeľnostnej reprezentácie) vyzerať nasledovne:

```

VISIBILITY REPRESENTATION(G)
1   $G^* \leftarrow$  Dual graph of  $G$ 
2   $X \leftarrow$  OPTIMAL TOPOLOGICAL NUMBERING OF  $G^*$ 
3   $Y \leftarrow$  OPTIMAL TOPOLOGICAL NUMBERING OF  $G$ 
4
5  for each  $v$  in  $V$ 
6    do  $\Gamma(v) \leftarrow$  LINE [ $X(left(v)), Y(v)$ ] to [ $X(right(v)-1), Y(v)$ ]]
7  for each  $e = (u, v)$  in  $E$ 
8    do  $\Gamma(u, v) \leftarrow$  LINE [ $X(left(e)), Y(u)$ ] to [ $X(left(e)), Y(v)$ ]

```



Obrázok 3: Vlastnosti planárnych st-grafov

V ďalšom kroku sa potom z tejto reprezentácie vytvorí nakreslenie grafu tak, že sa za umiestnenie vrcholu grafu vyberie ľubovoľný bod jeho vrcholovej úsečky. Formálnejšie popíšeme nakreslenie grafu z viditeľnostnej reprezentácie v nasledovnom algoritme:

```

POLYLINE( $G=(V,E)$ )
1  for each  $v$  in  $V$ 
2    do  $P(v) \leftarrow$  arbitrary  $(x_v, y_v)$  from  $\Gamma(v)$ 
3  for each  $e = (u, v)$  in  $E$ 
4    do if  $(y(v)-y(u)=1)$ 
5      then LINE  $P(u)$  to  $P(v)$ 
6      else POLYLINE  $P(u)$  to  $P(v)$ 
7          through  $(x(\Gamma(u, v)), y(u) + 1)$  and  $(x(\Gamma(u, v)), y(v) - 1)$ 

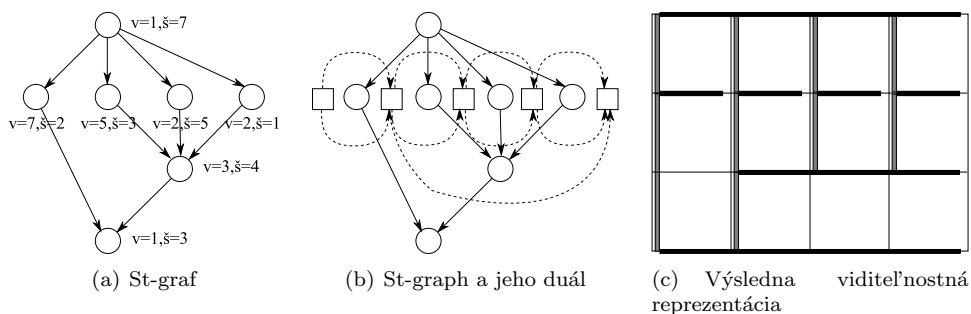
```

4.2 Úprava algoritmu

Pri využívaní algoritmov na kreslenie grafov narážame na problém, ako je definované nakreslenie grafu. Pripomeniem, že formálne vzaté je kreslenie grafu funkcia, ktorá mapuje vrcholy na rôzne body v rovine a hrany na krivky, ktoré spájajú vrcholy incidentné s touto hranou. Lenže pri vizualizácii boboxu nepotrebujeme kresliť body ale krabičky a via, ktoré majú istý rozmer. Najmä pri krabičkách môže byť rozmer väčší a rozhodne nie je zanedbateľný. Preto potrebujeme algoritmus upraviť s ohľadom na rozmery vrcholov.

Najjednoduchším riešením je úprava iba posledného kroku algoritmu (nahrádzanie úsečiek z viditeľnostnej reprezentácie), kedy namiesto bodu umiestňujeme na vrcholovú úsečku celý obdĺžnik (resp. iný útvar) a tým posúvame a rozťahujeme celú mriežku. Avšak jeden veľký (dlhý alebo široký) vrchol nám vie pokaziť celú úroveň na mriežke aj napriek tomu, že môže byť na krátkej ceste a tým padom nemusí ničomu vadit'. Preto sme sa rozhodli modifikovať už počítanie súradníc na mriežke tak, aby vrcholy mali zaručený dostatok priestoru na umiestnenie.

Máme teda daný planárny st-graf G spolu s rozmermi jeho vrcholov (obr. 4(a)). Vytvoríme jeho duál G^* (obr. 4(b)). Pôvodný algoritmus počítal na určenie súradníc



Obrázok 4: Pôvodny algoritmus

topologické číslovanie s rovnakou (jednotkovou) váhou na všetkých hranách (výsledná viditeľnosťná reprezentácia je na obr. 4(c)). V modifikovanom algoritme na tento účel využijeme vážené topologické číslovanie vrchlov.

Na y-ovú súradnicu vrcholov vplyva číslovanie v pôvodnom st-grafe G . Preto hrany grafu G ohodnotíme tak, aby ich váha zodpovedala výške, ktorú potrebuje vrchol na svoje umiestnenie (vid' obr. 5(a)). Formálne $weight(u, v) = height(u)$. Potom pri počítaní y-ových súradníc máme zaručené, že vrchol v je umiestnený tak, aby sa vrchol u na výšku zmestil nad neho. Okrem toho v prípade, že nejaký vrchol je vysoký, tak ovplyvňuje iba vrcholy, ktoré musí a vrcholy na nezávislých cestách (cestách, ktoré nemajú spoločné hrany) sa môžu dostať do tejto výšky bez obmedzení.

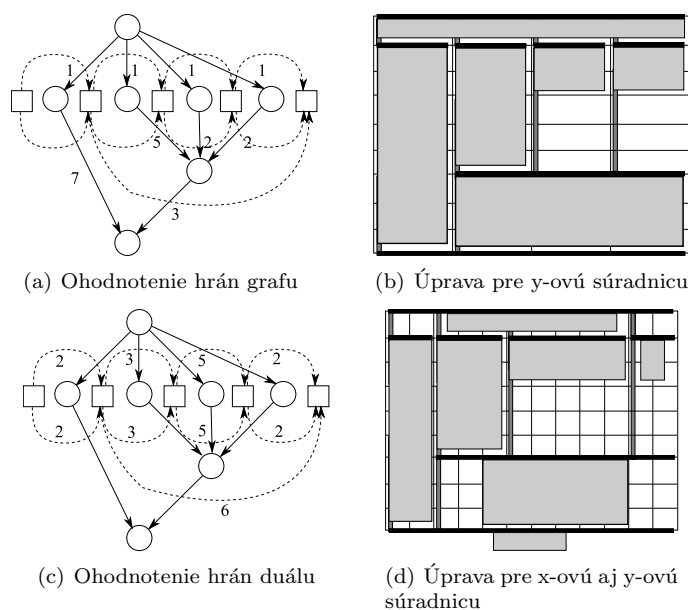
S x-ovou súradnicou je to kúsok náročnejšie. Pripomeňme, že hrany v duálnom grafe sú $E^* = \{(f, g) \mid \exists e \in E(G) \setminus (s, t) : f = left(e) \text{ and } g = right(e)\}$. Triviálnym riešením vyzerá byť ohodnotenie hrany tak, že hranu (f, g) , kde $e = (u, v)$ ohodnotíme maximom spomedzi širok vrcholov u a v . Avšak takto definované ohodnotenie je síce korektné, avšak zbytočne veľké. V prípade, že má vrchol niekoľko susedov – napr. potomkov (aj malej šírky) tak stačí jeho šírku rozdeliť medzi hrany incidentné s ním a týmito vrcholmi. Preto si najskôr pre každý vrchol v vypočítame dve hodnoty $weight_{in}(v) = \lceil width(v)/deg_{in}(v) \rceil$ a $weight_{out}(v) = \lceil width(v)/deg_{out}(v) \rceil$. Potom hranu (f, g) , kde $e = (u, v)$ ohodnotíme maximom spomedzi $weight_{out}(u)$ a $weight_{in}(v)$. Týmto dosiahneme pre každý vrchol dostatok šírky preň (vid' obr. 5(c)). V budúcnosti sa budeme snažiť toto ohodnotenie ešte optimalizovať.

Výsledný algoritmus teda vyzerá nasledovne:

MODIFIED VISIBILITY REPRESENTATION(G)

```

1 for each  $v$  in  $V$ 
2   do  $weight_{out}(v) \leftarrow \lceil width(v)/deg_{out}(v) \rceil$ 
3   do  $weight_{in}(v) \leftarrow \lceil width(v)/deg_{in}(v) \rceil$ 
4
5 for each  $e = (u, v)$  in  $E$ 
6   do  $weight(e) \leftarrow height(u)$ 
7
```



Obrázok 5: Úprava algoritmu pre rôzne veľkosti vrcholov

```

8  $G^* \leftarrow$  Dual graph of  $G$ 
9
10 for each  $(f, g)$  in  $E^*$ 
11   do let  $e = (u, v) \in E(G) : f = \text{left}(e)$  and  $g = \text{right}(e)$ 
12      $\text{weight}((f, g)) \leftarrow \max\{\text{weight}_{out}(u), \text{weight}_{in}(v)\}$ 
13
14  $X \leftarrow$  OPTIMAL WEIGHTED TOPOLOGICAL NUMBERING OF  $G^*$ 
15  $Y \leftarrow$  OPTIMAL WEIGHTED TOPOLOGICAL NUMBERING OF  $G$ 
16
17 for each  $v$  in  $V$ 
18    $\Gamma(v) \leftarrow$  LINE  $[X(\text{left}(v)), Y(v)]$  to  $[X(\text{right}(v)-1), Y(v)]$ 
19 for each  $e = (u, v)$  in  $E$ 
20    $\Gamma(u, v) \leftarrow$  LINE  $[X(\text{left}(e)), Y(u)]$  to  $[X(\text{left}(e)), Y(v)]$ 

```

Na finálne nakreslenie grafu potrebujeme ešte umiestniť vrcholy a hrany. Vrcholy (obdĺžniky) umiestňujeme podobne ako body v pôvodnom algoritme s tým, že na vrcholovej úsečke musí ležať celá horná hrana obdĺžnika zodpovedajúceho vrcholu.

Pre nakreslenie hrán máme niekoľko možností. Prvou je kreslenie podobne ako v pôvodnom algoritme. Druhá možnosť je kreslenie hrán pomocou postupného presmerovávania hrán okolo už nakreslených vrcholov [6].

4.3 Vstupné podmienky

Graf zodpovedajúci modelu Boboxu vo všeobecnosti nemusí spĺňať vstupné podmienky pre prístup s využitím viditeľnosti. Algoritmus požaduje planárny acyklický graf a preto je možné, že bude potrebné si neplanárny graf predprípraviť do tohoto tvaru – použiť planarizáciu. Prvým krokom planarizácie je nájsť čo najväčší planárny podgraf (odoberaním hrán). Tento problém je NP-t'ažký, ale existujú uspokojivé heuristiky na jeho riešenie (napr. [8]). Potom je možné buď kresliť graf bez týchto hrán a po nakreslení ich pridať alebo nahradiť každé kríženie fiktívnym vrcholom [9].

Pre úplnosť je potrebné dodať, že aj v prípade, že graf nebude acyklický je možné ho acyklickým urobiť (napríklad ubratím resp. otočením niektorých hrán) [5].

5 Záver

Pri spracovávaní dotazov v mnohých prípadoch nepostačuje paralelizácia na úrovni dotazov. V projekte Bobox je dotaz rozdelený na jednoduché výpočtové komponenty, ktoré sa navzájom prepoja. Vzor tejto štruktúry ako aj skutočný beh výpočtu je potrebné prehľadne zobrazit'. Na vizualizáciu modelu ako aj inštalácie modelu môžeme použiť metódu kreslenia orientovaných acyklických grafov s využitím viditeľnosti. V článku sme túto metódu kreslenia grafov modifikovali pre vrcholy, ktoré majú byť reprezentované obdĺžnikmi.

V budúcnosti je potrebné podrobnejšie preskúmať vlastnosti grafov modelu ako aj inštalácie modelu. Práve s ohľadom na tieto grafové vlastnosti môžeme ďalej upravovať a vylepšovať algoritmus pre vizualizáciu. Jedným z vylepšení, ktoré priamo vychádzajú z výsledkov článku je optimalizácia ohodnocovania hrán duálneho grafu, čím vieme dosiahnuť menšiu šírku výsledného nakreslenia.

Literatúra

- [1] G. Di Batista, R. Tamassia, and I. G. Tollis. Constrained Visibility Representations of Graphs. *Inform. Process. Letter*, 41:1–7, 1992.
- [2] G. Di Battista and R. Tamassia. Algorithms for Plane Representations of Acyclic Digraphs. *Theoretical Computer Science*, 61:175–198, 1988.
- [3] David Bednárek, Jiří Dokulil, Jakub Yaghob, and Filip Zavoral. The bobox project - a parallel native repository for semi-structured data and the semantic web. In Filip Zavoral and Peter Vojtáš, editors, *ITAT 2009 - IX. Informačné technológie - aplikácie a teória*, pages 44–59, 2009.
- [4] David Bednárek, Jiří Dokulil, Jakub Yaghob, and Filip Zavoral. Using methods of parallel semi-structured data processing for semanticweb. In Nima Dokoochaki, Filip Zavoral, and Josef Noll, editors, *3rd International Conference on Advances in Semantic Processing, SEMAPRO*, pages 44–49. IEEE Computer Society Press, 2009.

- [5] B. A. Berger and P. W. Shor. Tight bounds for the acyclic subgraph problem. *Journal of Algorithms*, 25:1–18, 1997.
- [6] Jiri Dokulil and Jana Katreniakova. Edge routing with fixed node positions. In *IV '08: Proceedings of the 2008 12th International Conference Information Visualisation*, pages 626–631, Washington, DC, USA, 2008. IEEE Computer Society.
- [7] W. Huang and P. Eades. How people read graphs. In *Asia-Pacific Symposium on Information Visualisation, APVIS 2005, Sydney, Australia, January 27-29, 2005*, pages 51–58. Australian Computer Society, 2005.
- [8] M. Jünger and P. Mutzel. Maximum Planar Subgraphs and Nice Embeddings: Practical Layout Tools. *Algorithmica*, 1(1):1–25, 1996. special issue on Graph drawing, edited by G. Di Batista.
- [9] P. N. Klein, S. Rao, M. Rauch, and S. Subramanian. Faster shortest-path algorithms for planar graphs. In *Proc. ACM Symp. on Theory of Computing*, 1994.

Obecné výpočty na grafických kartách – použitelnost, vlastnosti, praktické zkušenosti

Martin Kruliš, Jakub Yaghob

KSI MFF UK

Malostranské nám. 25, Praha

{krulis,yaghob}@ksi.mff.cuni.cz

Abstrakt. Nedávný pokrok ve vývoji grafických čipů umožnil provádět na běžných grafických kartách obecné výpočty. V kombinaci s univerzální platformou OpenCL, jejíž první implementace pro GPU byly uvedeny v druhé polovině roku 2009, slibují tyto technologie značné zrychlení paralelizovatelných úloh. Tento článek se zabývá použitelností těchto technologií na různé praktické problémy. Zároveň nabídne výkonnostní srovnání prototypových implementací na soudobém hardware.

Klíčová slova: GPGPU, grafické karty, paralelizace

1 Úvod

Jedním z hlavních cílů informatiky jak na poli teoretickém tak praktickém je snaha optimalizovat řešení nejrůznějších problémů za účelem jejich rychlejšího zpracování. Efektivnější algoritmy zvládnou řešit problémy rychleji, případně vyřešit větší problémy ve stejném čase. Paralelizace úloh byla vždy jedním ze slibných směrů optimalizace řešení. V minulosti byl tento přístup výsadou specializovaných výpočetních stanic a sálových počítačů. Díky nedávné revoluci na poli čipů grafických karet je dnes možné z běžně dostupných komponent postavit domácí počítač, který bude mít výpočetní výkon srovnatelný s o několik let staršími superpočítači.

Tento článek shrnuje výsledky zkoumání použitelnosti grafických karet pro obecné výpočty nejrůznějších typů. U všech příkladů uvádíme také použitý algoritmus a popisujeme nutné úpravy pro jeho nasazení na GPU. Každý příklad také demonstruje určitý problém, který se může při těchto úpravách projevit, zejména pak otázka latence a propustnosti paměti.

1.1 Hardware a metodika měření

Všechny příklady byly testovány na grafické kartě ATI Radeon HD 5870 s čipem RV870 a 1GB interní paměti DDR5 taktované na 4800MHz s 256 bitovou sběrnici. Jádro pracuje na frekvenci 850MHz a obsahuje 1600 5-cestných shader jednotek (což

odpovídá 320 výpočetním jádrům), které běží na stejné frekvenci [1]. Testovací sestavu pohání procesor Intel Core i7 860 obsahující 4 fyzická jádra s technologií Hyperthreading (tzn. 8 logických jader) taktovaná na 2.8GHz. Sestava obsahovala 4GB operační paměti DDR3 na frekvenci 1333MHz a 64-bitový operační systém Windows 7.

Každý test byl opakován desetkrát a výsledné naměřené časy jsou aritmetickým průměrem těchto měření. Jednotlivá měření každého testu se od sebe nelišila o více než 5% od výsledného průměru.

1.2 Osnova

Sekce 2 stručně shrnuje nejdůležitější aspekty architektury GPU a její abstrakce, kterou poskytuje framework OpenCL. Sekce 3 představuje nejjednodušší možný problém – jednoduché SIMD výpočty nad jednorozměrným vektorem čísel. Sekce 4 rozebírá velmi známý problém násobení velkých matic, na kterém demonstruje problémy s přístupem do globální paměti GPU. Sekce 5 navazuje Floyd-Warshallovým algoritmem, který je do jisté míry podobný problému násobení matic. Sekce 6 se věnuje otázce NP problémů a jejich řešení pomocí backtrackingu. Sekce 7 shrnuje poznatky z celého článku.

2 Architektura GPU a OpenCL framework

V této sekci se budeme věnovat nejdůležitějším vlastnostem architektury GPU [1, 2] a frameworku OpenCL [3] pro paralelní výpočty. Neklademe přitom důraz na úplnost, spíše se sazážíme zdůraznit aspekty, na které je třeba brát zvláštní ohled při paralelizaci algoritmů.

2.1 Architektura GPU

Architektury GPU a CPU se značně liší v celé řadě ohledů, zejména

- v počtu výpočetních jader,
- ve výpočetním výkonu (zejména v plovoucí desetinné čárce) a
- v přístupu k řešení problému latence paměti.

GPU má mnohem vyšší počet jader než soudobá CPU, avšak tato jádra mají celou řadu omezení. Jádra jsou seskupena do SMP jednotek¹, které aplikují SIMD² výpočetní model. Všechna jádra jedné SMP jednotky mají tedy stejný program a vykonávají vždy tutéž instrukci (pouze nad jinými daty). Například ATI Radeon použitý pro testy obsahuje 320 jader, která jsou seskupena do 20 SMP jednotek po 16 jádrech.

Na jednu SMP jednotku bývá typicky naplánováno více vláken, než kolik má jednotka procesorů, přičemž počet těchto vláken je násobkem počtu procesorů. Všechna tato vlákna (jak běžící tak naplánovaná) běží ve střídavém SIMD režimu – tedy musí

¹Streaming MultiProcessor Units

²Single Instruction Multiple Data

mít stejný program, který ale nevykonávají zcela souběžně, nýbrž se po skupinách střídají na dostupných jádrech. V případě, že právě běžící sada vláken je nucena z nějakého důvodu čekat (např. na načtení dat z globální paměti), plánovač tuto sadu odstaví a mezi tím nechá počítat jinou sadu naplánovaných vláken, aby byly procesory co nejvíce vytíženy. Touto technikou se GPU snaží minimalizovat efekt latence paměti.

Grafická karta obsahuje několik druhů pamětí:

- *Registry* – velmi rychlá paměť, kterou má k dispozici každé jádro. Nejnovější GPU nabízí až 1024 registrů po 32 bitech.
- *Sdílená paměť* – vyrovnávací paměť SMP jednotky, do které mohou přistupovat všechna vlákna. Pro grafické výpočty funguje zpravidla podobně jako L1 cache na CPU. Tato paměť je stejně rychlá jako registry, ale přístup do ní má jistá omezení (viz dále). Současná GPU mají řádově desítky kB paměti na SMP jednotku (typicky 32 nebo 64 kB).
- *Globální paměť* – je společná pro celý čip (všechny SMP jednotky). Přístup do ní je výrazně pomalejší neboť všechny SMP sdílí paměťovou sběrnici a data z ní nejsou většinou ukládána do žádné (nebo jen velmi malé) vyrovnávací paměti. Současné karty mají stovky MB až jednotky GB této paměti.

Kromě těchto pamětí je potřeba ještě zdůraznit, že při kombinování výpočtů na CPU a GPU jsou data, která chceme použít, typicky umístěna v paměti hostujícího PC (v operační paměti CPU). Proto je potřeba nejprve data přenést do globální paměti grafické karty, než je vůbec možné začít samotný výpočet.

2.2 Spouštění vláken

Framework OpenCL nabízí řadu mechanismů, pomocí kterých je schopen detekovat paralelní hardware a spustit na něm požadovaný kód. Kód je v programu reprezentován zpravidla ve zdrojovém tvaru a před spuštěním je zkompilován přímo pro cílovou architekturu. Funkce, které slouží jako vstupní body vláken, se nazývají *kernels*. Kernel může být spuštěn buď klasickým způsobem (jako jedna instance vlákna), nebo paralelně na více dat.

Při paralelním spuštění jednoho kernelu jsou instance vláken organizovány do jedno až tří rozměrného pole. Svou pozici v poli může kernel zjistit pomocí vestavěných funkcí a tato pozice také jednoznačně určuje, jakou část práce má instance vlákna vykonat. Kromě toho se instance vláken sdružují do skupin. Velikost skupiny musí být v každém rozměru soudělná s velikostí pole vláken. Vlákna v jedné skupině jsou fyzicky mapována na jednu SMP jednotku se všemi výhodami (např. sdílená paměť) a omezeními (např. SIMD režim), která z toho vyplývají.

Všechny operace prováděné na zařízení jsou spravovány tzv. frontou požadavků. Do této fronty se vkládají požadavky na spuštění kernelů, přesuny dat z/do grafické karty a také synchronizační značky a bariéry. Technické detaily si dovolueme z důvodu úspory místa vynechat.

2.3 Správa paměti

Nyní se podíváme na paměť ještě jednou, tentokrát z pohledu architektury frameworku, a zmíníme také některá podstatná omezení. Z hlediska OpenCL dělíme paměť na:

- *Privátní* – paměť vlastní jednotlivým instancím kernelů. Tato paměť v podstatě odpovídá registrům jednotlivých jader.
- *Lokální* – paměť sdílená mezi vlákny v jedné skupině. Tato paměť odpovídá sdílené paměti SMP jednotek.
- *Globální* – paměť sdílená všemi instancemi kernelu, což odpovídá globální paměti GPU.
- *Konstantní* – speciální případ globální paměti určené pouze pro čtení. Díky tomu, že je dopředu deklarováno, že se jedná o konstantní paměť, může být přístup k datům v této paměti optimalizován.

Běžící vlákna si nemohou žádnou paměť alokovat. Veškeré alokace musí být deklarovány před spuštěním kernelu a jednotlivé instance mohou manipulovat pouze s pamětí, na kterou dostanou ukazatele v argumentech kernelu. Z toho vyplývá poměrně zásadní omezení, že jednotlivé instance musí vracet výsledky konstantní velikosti nebo je potřeba velikost výsledku nejprve dopředu spočítat.

Při práci s pamětí GPU musíme mít na paměti ještě dvě důležitá omezení [1, 2]. První omezení se týká načítání dat z globální paměti. Globální paměť je přístupná skrze relativně širokou sběrnici, avšak latence požadavků je poměrně výrazná. Z tohoto důvodu se optimalizují přenosy, při kterých sousední vlákna přenáší sousední bloky paměti (tzv. coalesced load). Pokud alespoň 16 sousedních vláken z jedné skupiny začne ve stejném okamžiku číst nebo zapisovat 16 sousedních bloků paměti (o velikosti 32 bitů) a celý tento blok (64B) je správně zarovnaný, hardware zpracuje tento přesun v jediném požadavku.

Druhé omezení se týká přístupu do lokální (sdílené) paměti. Lokální paměť je rozdělena do tzv. *bank* (na nejnovější architekturách od je těchto bank zpravidla 16). Dvě sousední 32-bit. buňky paměti se nachází v následujících dvou bankách (modulo 16). Pokud dvě jádra v jedné instrukci přistoupí do stejné banky, je tento přístup serializován, čímž dojde ke zpomalení výpočtu. Výjimkou je, pokud všechna vlákna čtou stejnou hodnotu z lokální paměti. Takový přístup je detekován v hardware a data jsou k jádrům přenesena pomocí broadcastu.

3 Jednoduché SIMD výpočty

Jak jsme zmínili v předchozí sekci, architektura grafických procesorů je od základu postavena na SIMD³ paralelizaci. První příklad proto otestuje výkonnost grafické karty při jednoduchých vektorových výpočtech, které se nejlépe zpracovávají právě na SIMD výpočetním modelu.

³Single Instruction Multiple Data

V příkladu vyzkoušíme dvě vektorové operace. Obě mají dvě vstupní pole x a y a výstupní pole z . Výpočet probíhá nezávisle nad všemi prvky pole, tedy pro všechna i z rozsahu pole provedeme $z[i] = op(x[i], y[i])$. První operace pouze vynásobí oba prvky. Druhá operace je o trochu složitější a při výpočtu uplatní také druhou odmocninu a goniometrické funkce:

$$op_2(x[i], y[i]) = \frac{y[i]\sqrt{x[i]}}{x[i]} + x[i] \cos(y[i])$$

Tato operace nemá žádný praktický základ. Byla pouze uměle vytvořena k otestování více druhů operací a složitějších matematických funkcí.

3.1 Naměřené výsledky

Pro účely následujících testů byly použity vektory čísel x a y o velikosti $16M$ (t.j. 2^{24}) 32-bitových reálných čísel (floatů). Kernely obou operací byly spuštěny paralelně pro každý prvek pole, přičemž velikost skupiny byla nastavena na 256 (což je maximum na použitém GPU). Tato hodnota byla vybrána na základě empirických pozorování.

V tabulce 1 jsou shrnuty výsledky měření. Jednotlivé sloupce obsahují srovnání sériového algoritmu, paralelní implementace používající knihovnu Threading Building Blocks [6], implementace OpenCL běžící na CPU a na GPU. Sloupec GPU obsahuje časy celého výpočtu včetně přesunu dat na grafickou kartu a zpět, zatímco GPU* obsahuje pouze časy samotného výpočtu.

	sériový	TBB	CPU	GPU	GPU*
op_1	38	22	65	189	15
op_2	505	130	165	196	21

Tabulka 1: Naměřené časy SIMD výpočtů v milisekundách

Z naměřených výsledků plyne, že GPU zvládne výpočty provést poměrně rychle, avšak úzkým hrdlem celé operace je přenos dat z hlavní paměti do paměti grafické karty a zpět. U všech výpočtů prováděných na GPU musíme tedy nutně započítat dobu potřebnou na přenos dat, případně plánovat více operací za sebou, jinak se triviální vektorové operace nevyplatí paralelizovat na GPU, neboť procesor je zvládne provést ve srovnatelném čase.

4 Násobení matic

Druhým příkladem je často používaný matematický problém – násobení matic. Přestože tato operace není příliš častá v oblasti zpracování dat, pomůže nám demonstrovat další aspekty přenosů dat mezi paměťmi. Tentokrát se zaměříme na přenosy dat mezi globální pamětí grafické karty a lokální (a privátní) pamětí jednotlivých jader. Pro jednoduchost se omezíme na násobení dvou čtvercových matic, jejichž strany mají délku mocniny 2. Rovněž pro jednoduchost předpokládáme, že druhá matice je již transponovaná, abychom lépe využili cache při výpočtu na CPU.

Přestože existují lepší algoritmy [4], v našem příkladu použijeme klasický algoritmus pracující v čase $\mathcal{O}(N^3)$, kde N je délka strany matice.

4.1 Naivní implementace

Kernel naivní implementace obsahuje pouze nejvíce vnořený cyklus. Zbývající dvě úrovně cyklů jsou nahrazeny vytvořením příslušného počtu vláken. Každý prvek výsledné matice je tedy počítán paralelně a v případě, že bychom měli k dispozici dostatek jader (řádově $\mathcal{O}(N^2)$), mohli bychom dosáhnout asymptotické složitosti $\mathcal{O}(N)$.

```
__kernel void mul_matrix (__global const float *m1,
                          __global const float *m2,
                          __global float *mRes)
{
    int n = get_global_size(0);
    int r = get_global_id(0);
    int c = get_global_id(1);
    float sum = 0;
    for (int i = 0; i < n; ++i)
        sum += m1[r*n + i] * m2[c*n + i];
    mRes[r*n + c] = sum;
}
```

Přestože zvolený algoritmus je velmi dobře paralelizovatelný a na první pohled slibuje výrazné zrychlení, experimentální výsledky naznačují opak. Časy uvedené v tabulce 2 byly naměřeny při násobení matic 32-bitových reálných čísel pro N rovno 1024 a 2048. Vlákna uspořádána do dvou rozměrného pole, které přesně kopíruje tvar matice, a spojena do skupin velikosti 16×16 .

Matice	sériový	TBB	CPU	GPU
1024×1024	3630	542	319	392
2048×2048	29370	4060	2311	3400

Tabulka 2: Časy násobení matic v milisekundách

Výpočet na GPU, na kterém spolupracovalo 320 jader, byl zřetelně pomalejší než výpočet na čtyřjádrovém CPU s hyperthreadingem. V tomto případě ale neleží problém v přenosu dat z operační paměti do paměti grafické karty, neboť tento přenos zabere pouze 20ms v případě $N = 1024$, resp. 70ms v případě $N = 2048$.

Otázku neuspokojivého výkonu našeho řešení zodpoví profilovací nástroj. V následující tabulce jsou uvedeny nejdůležitější hodnoty naměřené při násobení dvou matic velikosti 1024×1024 .

Fetch (kolikrát četlo každé vlákno z globální paměti)	2048×
ALUFetchRatio (poměr operací ALU vůči čtení)	2.51%
ALUFetchBusy (podíl operace čtení na celkovém času)	94.37%
FetchUnitStalled (kolik času čekaly fetch jednotky na data)	83.15%

Z naměřených údajů je jasné, že většinu času celého výpočtu se pouze přenášela data z hlavní paměti k jádrům, přičemž se téměř výhradně na data čekalo. Při optimalizaci se tedy zaměříme hlavně na přenosy dat v rámci grafické karty.

4.2 Cache-aware implementace

Vylepšená implementace využívá faktu, že vlákna jsou spouštěna v SIMD režimu po skupinách. Každá tato skupina má k dispozici lokální paměť⁴, kterou může využít jako cache pro data z globální paměti. Při vhodné úpravě algoritmu můžeme docílit výrazného zrychlení, neboť

- přístup do lokální paměti je stejně rychlý jako přístup do privátní paměti vlákna,
- vlákna mohou spolupracovat při načítání dat (tzv. coalesced load) a
- většina dat je sdílena mezi vlákny, takže postačí jedno načtení těchto dat do lokální paměti místo aby si každé vlákno tato data stahovalo zvlášť.

Vlákna jsou uspořádána do skupin velikosti 16×16 . Při zpracování matice provedou vlákna v každém průchodu hlavním cyklem dva kroky. V prvním kroku kooperativně načtou čtvercové výřezy velikosti 16×16 prvků z násobených matic. Tyto výřezy obsahují prvky řádků resp. sloupců násobených matic, takže vždy jeden řádek resp. sloupec je sdílen 16 vlákny ve skupině. V druhém kroku si každé vlákno přičte následujících 16 součinů prvků k částečnému součtu. Za prvním i druhým krokem následuje synchronizační bariéra, která zajistí, že všechna vlákna vykonávají stejný krok. Další detaily jsou patrné z následujícího kódu kernelu.

```
__kernel void mul_matrix_opt(__global const float *m1,
                             __global const float *m2,
                             __global float *mRes,
                             __local float *tmp1,
                             __local float *tmp2)
{
    int size = get_global_size(0);
    int lsize_x = get_local_size(0);
    int lsize_y = get_local_size(1);
    int block_size = lsize_x * lsize_y;
    int gid_x = get_global_id(0);
    int gid_y = get_global_id(1);
    int lid_x = get_local_id(0);
    int lid_y = get_local_id(1);
```

⁴V případě použité grafické karty AMD Radeon HD5870 je to 32kB.

```

int offset = lid_y*lsize_x + lid_x;

float sum = 0;
for (int i = 0; i < size; i += lsize_x) {
    // Load data to local memory
    tmp1[offset] = m1[gid_y*size + i + lid_x];
    for (int j = 0; j < lsize_x / lsize_y; ++j)
        tmp2[offset + j*block_size] =
            m2[(gid_x + lsize_y*j)*size + i + lid_x];

    barrier(CLK_LOCAL_MEM_FENCE);

    // Add data from block to the sum
    for (int k = 0; k < lsize_x; ++k)
        sum += tmp1[lid_y*lsize_x + k] * tmp2[lid_x*lsize_x + k];

    barrier(CLK_LOCAL_MEM_FENCE);
}
mRes[gid_y*size + gid_x] = sum;
}

```

V tabulce 3 jsou opět naměřené výsledky, včetně optimalizované verze pro GPU, která je označena GPU⁺. Toto řešení již vykazuje výrazné zrychlení – 45× resp. 52× proti sériové verzi a přibližně 4× proti OpenCL verzi spuštěné na všech jádrech CPU.

Matice	sériový	TBB	CPU	GPU	GPU ⁺
1024 × 1024	3630	542	319	392	81
2048 × 2048	29370	4060	2311	3400	564

Tabulka 3: Časy násobení matic v milisekundách

5 Floyd-Warshallův algoritmus

Podobnou charakteristiku z hlediska časové složitosti a přístupu do paměti jako mělo násobení matic vykazuje také známý algoritmus na hledání nejkratších cest v grafu. Násobení matic však mohlo paralelizovat vnější dva cykly, díky čemuž se vnitřní cyklus vykonával přímo v kernelu. Floyd-Warshallův algoritmus proti tomu dokáže paralelismu využít pouze u vnitřních dvou cyklů, protože po každém průchodu vnějším cyklem musí dojít k synchronizaci, abychom udrželi integritu dat. OpenCL bohužel neobsahuje globální bariéru pro všechny instance jednoho kernelu a ani její implementace by nebyla příliš efektivní. Proto přesuneme vnější cyklus mimo grafickou kartu a v každém jeho kroku spustíme paralelně kernel řešící vnitřní dva cykly.

Tabulka 4 shrnuje naměřené výsledky, přičemž verze pro GPU již používá obdobnou optimalizaci, jako algoritmus násobení matic. Z výsledků je patrné, že cyklická invokace kernelů je výrazně méně efektivní než provedení cyklu uvnitř kernelu. I přesto

Vrcholů	sériový	TBB	CPU	GPU
1024	1262	1006	5200	289
2048	10600	7775	35800	3285

Tabulka 4: Časy Floyd-Warshallova algoritmu v ms

však podává naše řešení více než uspokojivé výsledky a použití GPU se i v tomto případě více než vyplatí.

6 NP problémy a backtracking

Dalším ukázkovým problémem je backtracking, který zde zastupuje exaktní způsob řešení NP problémů. NP problémy není⁵ v současné době možné řešit v polynomiálním čase. Masivní paralelismus je proto jednou z cest, jak alespoň o trochu posunout hranice velikostí problémů, které je možné v rozumné době spočítat.

Jako reprezentanta jsme vybrali NP-úplnou úlohu známou pod názvem Součet podmnožiny, jejíž zadání je následující. Je dána množina \mathcal{M} celých čísel a celé číslo s . Ptáme se, zdali existuje vybraná podmnožina $\mathcal{M}' \subset \mathcal{M}$ taková, že součet všech jejích prvků je právě s .

Princip řešení je velice snadný. Vybereme všechny existující podmnožiny \mathcal{M} (kterých je $2^{|\mathcal{M}|}$) a u každé z nich ověříme, zda nemá součet s . Testy budeme provádět na množině velikosti $|\mathcal{M}| = 30$. Každou podmnožinu identifikujeme 30-bitovým číslem, kde bity odpovídají jednotlivým prvkům, přičemž 1 znamená, že je daný prvek v podmnožině přítomen a 0, že v ní přítomen není.

Každý kernel pak dostane prefix délky 24 bitů, které použije jako pevný základ a pro zbývajících 8 bitů vyzkouší všechny možnosti. Tabulka 5 shrnuje naměřené výsledky.

$ \mathcal{M} $	sériový	TBB	CPU	GPU
30	6625	1865	3820	595

Tabulka 5: Časy hledání podmnožiny s daným součtem v ms

Při testování jsme si dovolili malé zjednodušení – všechna čísla v množině byla sudá, zatím co hledaný součet s byl lichý, abychom donutili algoritmus projít všechny podmnožiny. Tento přístup ponechává ve vzduchu otázku jak zastavit běžící výpočet, když jedno z vláken nalezne řešení. Vzhledem k množství vláken a nemožnosti efektivní komunikace mezi nimi se jako nejlepší způsob jeví spuštění kernelů po vhodné velkých částech tak, aby se příliš nezvýšila režie, ale zároveň aby se jednotlivé části nepočítali příliš dlouho. Po skončení výpočtu každé části se provede kontrola, zda byl již výsledek nalezen, a pokud ano, výpočet skončí.

⁵Autor se zde přiklání k všeobecně rozšířené domněnce, že $P \neq NP$.

7 Závěr

V tomto článku jsme představili relativně novou architekturu na poli paralelních výpočtů a ověřili její použitelnost na řadě různorodých problémů. Výsledky naznačují, že se jedná o slibnou technologii, avšak její použitelnost je do značné míry omežována nutností v některých případech poměrně rozsáhlých úprav algoritmů a programovacích technik. Zatím co na CPU je programátor často zachráněn přítomností velké cache, na GPU je potřeba přístupy do paměti pečlivě plánovat a optimalizovat.

V budoucí práci bychom se rádi zaměřili na využití GPU při zpracování databázových dotazů. Zejména nás zajímají možnosti nasazení na relační a sémantická data (RDF [5], linked-data). V této souvislosti bychom také rádi prozkoumali možnosti zpracování grafových algoritmů, které by mohli být využity při zpracování indexů síťových dat.

Literatura

- [1] ATI Stream Computing – OpenCL Programming Guide. http://developer.amd.com/gpu/ATIStreamSDK/assets/ATI_Stream_SDK_OpenCL_Programming_Guide.pdf.
- [2] NVIDIA OpenCL Programming Guide. http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_OpenCL_ProgrammingGuide.pdf.
- [3] OpenCL Framework Manual. <http://www.khronos.org/ocl/>.
- [4] R.P. Brent and STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE. *Algorithms for matrix multiplication*. Citeseer, 1970.
- [5] Frank Manola and Eric Miller. RDF Primer, W3C Recommendation, February 2004. <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.
- [6] J. Reinders. *Intel threading building blocks*. O'Reilly, 2007.

(Eds.) David Obdržálek, Martin Plátek

MIS 2010

Vydal
MATFYZPRESS
vydavatelství
Matematicko-fyzikální fakulty
Univerzity Karlovy v Praze
Sokolovská 83
186 75 Praha 8
jako svou 342. publikaci

Z připravených předloh
vytisklo ReproStředisko UK MFF
Sokolovská 83, 186 75 Praha 8

Vydání první
Praha 2010

ISBN 978-80-7378-148-4