Advances in Computer Science Research

Volume 12

Oficyna Wydawnicza Politechniki Białostockiej

Białystok 2015

EDITOR-IN-CHIEF / REDAKTOR NACZELNY:
Marek Krętowski (m.kretowski@pb.edu.pl, 85 746 90 95)

SCIENTIFIC EDITORS / SEKRETARZE NAUKOWI:
Magdalena Topczewska (m.topczewska@pb.edu.pl, 85 746 90 86)
Marek Parfieniuk (m.parfieniuk@pb.edu.pl, 85 746 91 08)

STATISTICAL EDITOR / REDAKTOR STATYSTYCZNY:
Dorota Mozyrska (d.mozyrska@pb.edu.pl, 85 746 90 82)

TECHNICAL EDITOR / SEKRETARZ TECHNICZNY:
Tomasz Łukaszuk (t.lukaszuk@pb.edu.pl, 85 746 90 86)

The articles published in *Advances in Computer Science Research* have been given a favourable opinion by reviewers designated by Editor-In-Chief and Scientific Board

ISSN 2300-715X

This journal is the continuation of *Zeszyty Naukowe Politechniki Białostockiej. Informatyka* issued in the years 2002-2012, ISSN 1644-0331

THE ADRESS FOR THE CORRESPONDENCE / ADRES DO KORESPONDENCJI:

„*Advances in Computer Science Research*"
Faculty of Computer Science / Wydział Informatyki
Bialystok University of Technology / Politechnika Białostocka
ul. Wiejska 45a, 15-351 Białystok
tel. 85 746 90 50, fax 85 746 97 22
e-mail: znpb@irys.wi.pb.edu.pl
http://irys.wi.pb.edu.pl/acsr

Printing / Druk: Oficyna Wydawnicza Politechniki Białostockiej
Issue / Nakład: 100 copies / egzemplarzy

CONTENTS

TOWARDS GESTURE RECOGNITION IN THREE-DIMENSIONAL SPACE

Łukasz Gadomer

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: In this work, author describes the continuation of his researches about gesture recognition. The previous varaint of the solution was using plain data and was dependent of the performance velocity. In the described researches author made it speed and position invariant by resolving problem of too long or too short gestures – in a previous solution the user had to decide about gesture duration time before performing, now it is not necessary. He also proposed another data representations, using features computed of recorded data. Previous representation, which assumed storing relative positions between samples, was replaced by transforming each gesture to the axis origin and normalizing. He also tried to connect these two representations – plain data and features – into a single one. All of these new data representations were tested using the SVM classifier, which was judged to be the best for the given problem in the previous work. Each of them was tested using one of four popular SVM kernel functions: linear, polynomial, sigmoid and radial basis function (RBF). All achieved results are presented and compared.

Keywords: data classification, gesture recognition, data features, three dimensional space, speed invariant, position invariant, kinect

1. Introduction

The gesture recognition problem is an issue which many authors are interested in. They treat and understand gestures in a different way and propose different solutions to resolve this problem. Some of these solutions are presented in the current paragraph.

The first approach assumes treating gesture as a movement of a single point (or a set of points) which represents a part of the body (usually a hand). This corresponds with definition which says that gestures are "movements of the arms and hands which are closely synchronized with the flow of speech" [1]. The author's solution, which is described in this publication, is based on this way of treating gestures. Sample

Advances in Computer Science Research, vol. 12, pp. 5-20, 2015.

works, where this approach is used, are [2], [3] and [4]. In all these solutions gestures were collected using accelerometer MEMS (Microelectromechanical System). It is a device which was placed in the user's hand and it was tracking hand's movement in three dimensional space. In [2] authors tried to recognize seven simple gestures. They measured motion in three dimensions, but in fact gestures were two–dimensional. They extracted features from collected data and then performed recognition. Authors of [3] performed their research on 18 different gestures — database was consisting of 3780 instances. They decided to resolve classification problem using Dynamic Time Warping (DTW). Third solution based on accelerometer is described in [4]. Authors were facing gesture recognition problem on 3200 same length (3.40 seconds) gesture instances, grouped into 8 gestures. They also used Dynamic Time Warping algorithm, but hardware accelerated. During the research they tested recognition accuracy in user–dependent and user–independent cases.

Another example of a similar gesture tracking approach is described in [5]. To collect gesture data, authors used finger–worn device called Magic Ring. They described a method of adaptive template adjustment to personalized gesture recognition. In [6] authors designed a solution based on 1$ algorithm which uses Compressed Sensing (CS) and Sparse Representation (SR) methods. Their solution is based on algorithm proposed in [7]. The algorithm is called 1$ to emphasize its low cost and simplicity. The implementation takes about a hundred lines of code. Authors claim it works well even with a single instance of each gesture as a training set. Authors of [8] proposed a solution based on a digital camera, plugged into a computer. The camera images were processed in real time and gesture data extracted from them. The Modified Levenshtein Distance were used as a classification method. Authors performed their research on the dataset consisting of ten digits.

In a solution proposed in this publication author uses Microsoft Kinect device. The same device was used in [9] and [10]. Authors of [9] used gesture recognition as a way of communication with a robot. To test their solution, they performed a research using Dynamic Time Warping algorithm. In [10] gesture recognition process was divided into three stages: modelling, analysis and recognition. The research was performed by authors using Hidden Markov Model.

Sometimes gesture is expressed by a movement of a set of points, each represents different part of the body. The example of recognition such kind of gesture was described in [11]. Authors also used the Kinect device to track user's movements. The points representing different parts of the user's body were used to recognize karate chops.

A different way of understanding gestures assumes treating them as a palm's shape. An example of treating gesture that way is shown in [12]. Author was interpreting gesture as a characteristic arrangement of a palm. Recognition was performed by palm images analysis, which were provided by a digital web camera. A similar approach was presented in [13]. Authors of this publication were studying a possibility of using gestures to interact with a computer game. Similarly to [12], device used to track performed gestures was a digital camera. Gesture recognition was possible thanks to detecting an outline of a palm. In order to check the practical usage of created solution authors tested performing them in a computer game created by themselves. The last example of understanding gestures in a similar way is [14]. In this article authors compared two gesture recognition algorithms: Finger-Earth Moving Distance (FEMD) and Shape Context. They proposed to use their solution in a Sudoku game. To test gesture data they used Microsoft Kinect device.

The last example of understanding gesture concept is sign language. According to [15], each sign can be divided into four parameters: hand shape, position, motion and orientation. Many authors proposed their solutions to track and recognize sign language. One of them is described in [16]. In this publication authors described their way of solving huge search space problem in a large sign vocabulary. Their research was made on 5113 signs and gave results about 95%. Another example is presented in [17]. Authors treat sign language as hand positions and movements. They proposed a sign language recognition method based on a multi–stream HMM technique. Research was performed on 21960 sign language word data. Classification accuracy achieved 70.6% for the same weight of hand position and movement and 75.6% for 0.2 : 0.8 weight proportion, which allowed them to conclude hand movement is more important in sign language.

Gesture recognition is a kind of data classification problem. The point of this issue is creating a solution which is able to track gesture performed by its user and recognize which gesture from the previously learned dataset it was. Author of this publication proposed such solution in [18]. It was able to recognize performed gestures in real–time using one of four selected classifiers. The real–time recognition was position invariant, which means user could stand in a different place of the controller's field of view and in a different distance from it. Recognition was not speed invariant — each set of recognizing samples had the same length. The solution also allowed to test prepared datasets in offline mode. This is the way each classifier was tested and compared with others.

The practical usage of real–time gesture recognition was shown using CAVE3D environment. It is the device which simulates three dimensional space by displaying prepared images on three cube–framed screens. User is placed into this space (between screen walls) and can communicate with it using gestures. Each gesture has assigned action which allows him to, for example, create objects or move.

The continuation of the work described in [18] is presented in this publication. Author concentrated on the strict aspect of gesture recognition and proposed two different data representations in comparison to the one presented in the previous work. Both of them were compared with themselves (and with the combination of them) using the best classifier fitting to the given problem (according to the previous work). He also improved the way of collecting gestures, which made recognition speed invariant.

2. Gesture tracking equipment

Author prepared his own dataset to learn the classifier and perform the research. He obtained gesture data by tracking them using Microsoft Kinect [19]. It is a registering device that has build-in (inter alia) color sensor, depth camera and infrared emitter. All these features allow for recording user's movements in three dimensional space. Color sensor is responsible for width and height recording, depth camera and infrared emitter provide support for the third dimension: the device captures gray scale images; in these images the intensity parameter defines depth, so objects that are located near the camera are represented in a different intensity than objects located far from the camera.

The used Kinect SDK [20] gave possibility of tracking user's skeleton. It is a data structure consisting of characteristic detected user's body points, which coordinates changes in real time (about 30 times in a second) according to current user's body position. Author used this feature to record user's right hand's position changes in time. It was helpful because author preferred to focus on gesture recognition instead of image analysis.

3. Gesture data

3.1 Speed invariance

In [18] authors proposed gesture representation which allowed them to make tracking position invariant. All gestures from the one dataset have defined the common number of samples. According to Kinect's capturing frequency the following samples positions were compared to the previous one. The difference between coordinates of two nieghboring samples was computed separately for each axis and stored in a single vector. It means that for each gesture which length was n samples, there was created $3n+1$ vector.[1] Capturing relative positions of samples instead of direct one allowed to make recognition independent of the place in field of controller's view where the gesture was performed. All vectors from the dataset were also normalized in the recognition process (in the dataset file they were stored unnormalized way), which allowed for comparison of high dimensional points and made recognition independent of the distance between the user and the controller.

This way of capturing gestures was not speed irrelevant. If the recorded gesture was too short (had too small number of samples), program was informing user about that and user had to record another gesture. If the gesture was too long, excessed samples were simply cut from the beginning of the gesture. It means user had to care about the approximate length of gesture he performed. This problem was resolved by changing the way of interpreting recorded gesture. Two ways of dealing with the improper length of recorded gesture were implemented:

– If gesture is too short, program puts additional sample in the middle of the longest distance between two samples in a recorded set. This action is being repeated until the proper number of samples is achieved. It is shown on figure 1, which is ilustrated by the single instance of 'O' gesture.

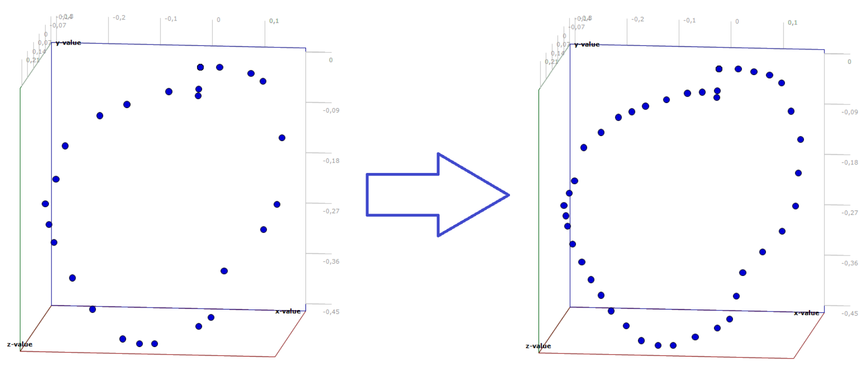


Fig. 1. The way of dealing with too short gestures.

– If gesture is too long, program removes single sample closest to another one in comparison to other neighbour couples in the set. The distance between the

[1] In fact for the gesture with length n there were captured $n+1$ samples to compute differences between coordinates of n couples of neighboring samples.

remaining neighbour of the deleted sample and that sample is added to the new remaining sample's neighbour. This action is repeating until achieving proper number of samples. It is shown on figure 2, which is ilustrated by the single instance of '(' gesture.

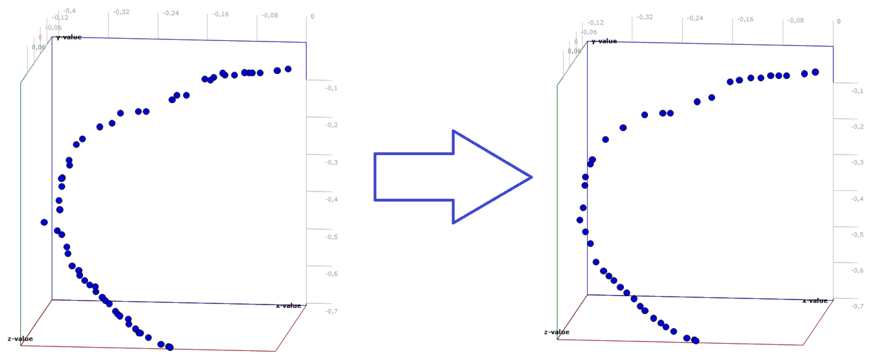


Fig. 2. The way of dealing with too long gestures.

Two proposed ways of dealing with too short or too long gestures makes recognition speed invariant. It means user can perform gesture with any speed — program rebuilds each recording to the same size.

3.2 Data representation

For the purposes of the research described in the following paragraphs recorded dataset were transformed. Performed gestures were written to file consisting of vectors created in the way described before. The idea of new data representation were to make recognition not only speed invariant, but also independent of the different styles of performing the same gesture by different people (for example: different velocity of performing creating gesture parts). What is more, there was an important issue to reduce the number of classifier inputs: for n samples, there were $3n$ inputs, which is a large number. To achieve this goals author decided to extract features from the dataset.

Vectors consisting of relative location to the previous samples were not good source of data to feature extraction. Their values have tendency to oscillate about zero, which was an intentional action for the previous assumptions, but inadequate to the new ideas. To allow for successful feature computation, relative vectors were changed to absolute, but always with their start at the axis origin. This returns recorded gestures to their source representation, but with additional improvement making every gesture start at the same point.

From the dataset prepared that way there were extracted features presented in Table 1. Most of these features were computed to the each of three axis independently, some of them also together for the all axes. Ratio features were computed between cartesian of axes. All in all there were produced 49 features.

Table 1. Features extracted from the dataset.

Feature	Comments
Average	Average value of the each axis and of the all axes.
Standard deviation	Standard deviation of the each axis and of the all axes' values.
Variance	Variance of the each axis and of the all axes' values.
Axis to axis ratio	Ratio between the amplitude of the extreme points of two axes. Computed for the each couple of axes.
Axis to axis correlation	Correlation between the two axes. Computed for the each couple of axes.
Axis to axis covariance	Covariance between the two axes. Computed for the each couple of axes.
Skewness	Skewness of the each axis and of the all axes.
Kurtosis	Kurtosis of the each axis and of the all axes.
Signal magnitude area	Signal magnitude area of the each axis and of the all axes.
Root mean square	Root mean square of the each axis and of the all axes.
Mean deviation	Mean deviation of the each axis and of the all axes.
Interquartile range	Interquartile range of the each axis.
Energy	Energy of the each axis and of the all axes.

Gesture recognition was tested using these features, transformed to absolute values and the same start point dataset and fusion of these two data types, which is further described in the following chapters.

4. Research description

Considering the novel way of recording data the new dataset was prepared. It has the same number and kind of gestures as the dataset tested in [18], but it consists of the new gesture instances. Because of that reason the results between these two researches should not be compared. There are 960 gesture instances in the dataset, divided into 12 different gestures, 80 instances each. These gestures are presented in table 2, which shows also where the gesture performing starts.

There was tested three representations of created dataset, which are mentioned in chapter 3.2. First of them assumed extracting absolute positions of the hand in gesture performance time from the source datafile, transforming it to the beginning at the axis origin and min–max normalizing with the [-1, 1] scope. Each gesture instance had

Table 2. Gestures which belong to dataset.

Gesture shape	Gesture name	Starting point
(	Opening bracket	Top
)	Closing bracket	Top
<	Less than	Top
>	More than	Top
∧	Caret	Left
\	Backslash	Top
/	Slash	Top
|	Vertical line	Top
—	Horizontal line	Right
∼	Tilde	Left
O	Circle	Top
8	Eight	Top

117 attributes (distances between 40 samples in 3 axes). The second representation was consisting of features extracted from the previously described one. There were 49 features, which was the number of attributes in this case. The third representation is a fusion of the first and the second one. In this case each gesture instance was described by hand positions and features. It means each gesture instance was described by 166 attributes (summary of two previous representations).

In the previous publication [18] the following classifiers have been tested: SVM [21], Neural Network and Radial Basis Network. SVM classifier gave the best results – it appeard to fit the best to the given problem and it was used to perform classification in this publication. Four kernel functions were tested and compared: linear, polynomial, sigmoidal and radial basis function (RBF). Using each of these functions there were performed the classification of three gesture representations. For each functions and each representation there was a necessity of classifier's parameters optimization. It was performed using 5-fold crossvalidation on the single division of the dataset. It was divided into five equal parts, each of this part was containing the same number of each gesture instances as the other ones. SVM algorithm was learning using four of these parts and testing using remaining one. It was repeated five times, every time another part was the testing one. The result was average of five achieved accuracies. Such operation was repeated many times on different classifier's parameters. The parameters were combined using grid search: for each parameter, there was assumed a range of testing and a step. There was tested all combinations of parameters in assumed ranges. Such optimization was performed for all kernel functions and for three data representations. What is more, after first optimization, there was performed a second one with narrowed ranges to fit dataset the best possible way.

All parameter ranges and test results are presented in tables 5, 6, 7, 8, 9, 10, 11, 12, 13.

After obtaining the best parameters of the classifier's kernel functions for each data representation, the research was performed using them. Like in parameters optimization process, there was used 5-fold crossvalidation, described above. The difference was about the dataset division. In the parameter optimization, there was only one division. In the research, there were used 100 different divisions. Using each of these divisions there was done 5-fold crossvalidation. Then all of one hundred results were averaged, which gave the final accuracy.

5. Results and discussion

The results of the research described in chapter 4. are presented in Table 3.

Table 3. Research results.

Kernel function	Data [%]	Features [%]	Data with features [%]
Linear	95.39	93.36	94.56
Polynomial	95.48	93.37	94.56
Sigmoid	95.35	8.56	8.38
RBF	95.58	88.11	38.71

As we can see, the best results were achieved for plain data representation. Using each tested kernel it was possible to achieve comparable accuracy which was higher than 95%. The best result for this gesture representation gave radial basis function kernel, which was 95.58%. For features representation the results was more differential. Linear and polynomial kernels gave almost the same result (about 93.65%), which was about 2% worse results than for plain data. It is understandable because feature representation has above twice columns less than plain data representation. RBF kernel achieved noticeably worse result: 88.11%. Sigmoid kernel gave the worst accuracy: 8.56% which is almost the same as random. Performing described tests author was unable to adjust this kernel to given problem. The most surprising is the result of data with features representation. It has most columns which means it brings the most information and allows to expect to give the best results of all representations. However, the results were different. For linear and polynomial kernel, the results were comparable (94,55%) and placed between plain data and features representation. For sigmoid and RBF kernels, they were even worse than for features representations. Sigmoid kernel gave results close to random, RBF achieved 38.71%, which is also bad result.

The sample confusion matrix is presented in Table 4. We can observe that the most common reason of missclassification was the similarity of gestures. For example, the gesture ’<’ in some cases was recognized as (and vice versa. The way of performing these gestures is quite similar so it could be a bit confusing for the classifier. The same issue was about gestures ’>’ and ’)’. Gesture ’|’ was missclassified as ’/’ and ’\’ for the same reason. This explains the main reason of mistakes.

Table 4. Sample confusion matrix.

Gest	O	/	\	∧	<	>	(	)	8	∼	|	—
O	99.75	0.00	0.00	0.00	0.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00
/	0.00	99.48	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.52	0.00
\	0.00	0.00	98.68	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.32	0.00
∧	0.00	0.00	0.13	98.62	0.00	0.00	0.00	0.00	1.25	0.00	0.00	0.00
<	0.00	0.00	0.00	0.00	90.45	0.00	9.55	0.00	0.00	0.00	0.00	0.00
>	0.00	0.52	0.00	0.00	0.00	92.87	0.00	6.21	0.00	0.00	0.00	0.92
(	0.00	0.00	0.00	0.00	7.93	0.00	89.55	0.00	0.00	0.00	2.52	0.00
)	0.00	0.00	0.00	0.00	0.00	5.47	0.00	92.58	0.00	0.00	1.95	0.00
8	1.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	98.69	0.00	0.00	0.00
∼	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00
|	0.00	2.68	5.87	0.00	0.00	0.00	1.02	0.64	0.00	0.00	89.80	0.00
—	0.00	2.43	0.00	0.00	0.00	1.51	0.00	0.00	0.00	0.00	0.81	95.25

6. Conclusion

In this paper author described his progress at work on gesture recognition issue. He improved solution developed by himself by making gesture recording and recognizing speed invariant. He also proposed three gesture representations and tested if it is possible to recognize gestures using each of them. To test the recognition accuracy, author used classifier which appeared to be the best for the given problem (according to his previous researches): SVM. He tested four kernel functions: linear, polynomial, sigmoid and radial basis function (RBF). All results were presented and compared.

It turned out that the best results can be achieved using plain data representation. In combination with new gesture tracking method it seems to be the best recognition way for the given problem. However, this gesture representation can bring the problem of too many features, which can make recognition process not efficient enough. The solution can be second proposed representation. It has been tested that for more than twice features less it can give only a bit worse results. When full gesture dataset in plain data recognition way gives satisfying accuracy, it can be beneficial to check feature representation, which allows to reduce recognition time. The third representation was expected to give best accuracy as having the largest number of features, which surprisingly was not proved by performed research. It can be tested for the more complex datasets, which makes the thread for the further researches.

Table 5. Parameters ranges used for the first test in a grid search to find the best ones — dataset containing of plain data.

First test ranges — data					
Kernel function	C	Gamma	Degree	Coef0	Best test result [%]
Linear	1–1000, step 1	X	X	X	95.625
Polynomial	1–50, step 1	0–100, step 1	1–10, step 1	X	96.146
Sigmoid	1–50, step 1	0–100, step 1	X	0–1, step 0.1	95.208
RBF	1–100, step 0-100, step 1	X	X	X	96.146

Table 6. Parameters ranges used for the second test in a grid search to find the best ones — dataset containing of plain data.

Second test ranges — data					
Kernel function	C	Gamma	Degree	Coef0	Best test result [%]
Linear	0.001–1, step 0.001	X	X	X	95.625
Polynomial	46–50, step 0.001	0	3–4, step 1	X	96.146
Sigmoid	1–50, step 1	0–0.04, step 0.002	X	0–0.1, step 0.02	95.208
RBF	81–92, step 0.001	0	X	X	96.146

Table 7. Best parameters achieved in a grid search — dataset containing of plain data.

Best parameters — data				
Kernel function	C	Gamma	Degree	Coef0
Linear	0.71	X	X	X
Polynomial	48	0	3	X
Sigmoid	37	0	X	0
RBF	85	0	X	X

Table 8. Parameters ranges used for the first test in a grid search to find the best ones — dataset containing of features.

First test ranges — features					
Kernel function	C	Gamma	Degree	Coef0	Best test result [%]
Linear	1–1000, step 1	X	X	X	93.333
Polynomial	1–50, step 1	0–100, step 1	1–5, step 1	X	94.167
Sigmoid	1–500, step 1	0–100, step 1	X	0–1, step 0.1	8.646
RBF	1–100, step 0-100, step 1	X	X	X	83.230

Table 9. Parameters ranges used for the second test in a grid search to find the best ones — dataset containing of festures.

Second test ranges — features					
Kernel function	C	Gamma	Degree	Coef0	Best test result [%]
Linear	0.001–1, step 0.001	X	X	X	94.167
Polynomial	10–16, step 0.001	0	1	X	94.167
Sigmoid	10	0	X	5–6, step 0.01	8.646
RBF	0.1–5, step 0.1	0–1, step 0.001	X	X	88.542

Table 10. Best parameters achieved in a grid search — dataset containing of features.

Best parameters — features				
Kernel function	C	Gamma	Degree	Coef0
Linear	0.243	X	X	X
Polynomial	11.92	0	1	X
Sigmoid	10	0	X	5.27
RBF	4.7	0.003	X	X

Table 11. Parameters ranges used for the first test in a grid search to find the best ones — dataset containing of plain data and features.

First test ranges — data and features					
Kernel function	C	Gamma	Degree	Coef0	Best test result [%]
Linear	1–1000, step 1	X	X	X	94.480
Polynomial	1–50, step 1	0–100, step 1	1–10, step 1	X	95.000
Sigmoid	1–50, step 1	0–100, step 1	X	0–1, step 0.1	8.542
RBF	1–100, step 0-100, step 1	X	X	X	40.313

Table 12. Parameters ranges used for the second test in a grid search to find the best ones — dataset containing of plain data and features.

Second test ranges — data and features					
Kernel function	C	Gamma	Degree	Coef0	Best test result [%]
Linear	0.01–10, step 0.01	X	X	X	95.000
Polynomial	10–20, step 0.01	0–0.1, step 0.001	1	X	95.000
Sigmoid	0–50, step 1	0–0.04, step 0.02	X	0–0.1, step 0.02	8.542
RBF	1–100, step 1	38.5–39.5, step 0.1	X	X	40.521

Table 13. Best parameters achieved in a grid search — dataset containing of plain data and features.

Best parameters — data and features				
Kernel function	C	Gamma	Degree	Coef0
Linear	0.08	X	X	X
Polynomial	14	0	1	X
Sigmoid	8	5	X	0
RBF	27	38.6	X	X

References

[1] D. McNeill. *Gesture and Thought*. University of Chichago Press, Chicago, USA, 2007.

[2] R. Xu, S. Zhou, and W. J. Li. Mems accelerometer based nonspecific-user hand gesture recognition. *Sensors Journal, IEEE*, 12(5):1166–1173, May 2012.

[3] A. Akl, C. Feng, and S. Valaee. A novel accelerometer-based gesture recognition system. *Signal Processing, IEEE Transactions on*, 59(12):6197–6205, Dec 2011.

[4] S. M. A Hussain, and A. B. M. H. Rashid. User independent hand gesture recognition by accelerated dtw. In *Informatics, Electronics Vision (ICIEV), 2012 International Conference on*, pages 1033–1037, May 2012.

[5] Y. Zhou, D. Saito, and L. Jing. Adaptive template adjustment for personalized gesture recognition based on a finger-worn device. In *Awareness Science and Technology and Ubi-Media Computing (iCAST-UMEDIA), 2013 International Joint Conference on*, pages 610–614, Nov 2013.

[6] A. Boyali, and M. Kavakli. A robust gesture recognition algorithm based on sparse representation, random projections and compressed sensing. In *Industrial Electronics and Applications (ICIEA), 2012 7th IEEE Conference on*, pages 243–249, July 2012.

[7] J. O. Wobbrock, A. D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *ACM Symposium on User Interface Software and Technology (UIST '07). Newport, Rhode Island*, pages 159–168, July 2007.

[8] C. Nyirarugira, H.-R. Choi, J. Kim, M. Hayes, and T. Kim. Modified levenshtein distance for real-time gesture recognition. In *Image and Signal Processing (CISP), 2013 6th International Congress on*, volume 2, pages 974–979, Dec 2013.

[9] S. Bodiroza, G. Doisy, and V. V. Hafner. Position-invariant, real-time gesture recognition based on dynamic time warping. In *Human-Robot Interaction (HRI), 2013 8th ACM/IEEE International Conference on*, pages 87–88, March 2013.

[10] Y. Wang, C. Yang, X. Wu, S. Xu, and H. Li. Kinect based dynamic hand gesture recognition algorithm research. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2012 4th International Conference on*, volume 1, pages 274–279, Aug 2012.

[11] T. Hachaj, M. R. Ogiela, and M. Piekarczyk. Image processing and communications challenge 5. In *Real-Time Recognition of Selected Karate Techniquies Using GDL Approach*, pages 99–106, 2014.

[12] M. Panwar. Hand gesture recognition based on shape parameters. In *Computing, Communication and Applications (ICCCA), 2012 International Conference on*, pages 1–6, Feb 2012.

[13] S. S. Rautaray, and A. Agrawal. Interaction with virtual game through hand gesture recognition. In *Multimedia, Signal Processing and Communication Technologies (IMPACT), 2011 International Conference on*, pages 244–247, Dec 2011.

[14] Z. Ren, J. Meng, and J. Yuan. Depth camera based hand gesture recognition and its applications in human-computer-interaction. In *Information, Communications and Signal Processing (ICICS) 2011 8th International Conference on*, pages 1–5, Dec 2011.

[15] W. Gao, J. Ma, S. Shan, X. Chen, W. Zheng, H. Zhang, J. Yan and J. Wu. Handtalker: A multimodal dialog system using sign language and 3-d virtual human. In Tieniu Tan, Yuanchun Shi, and Wen Gao, editors, *ICMI*, volume 1948 of *Lecture Notes in Computer Science*, pages 564–571. Springer, 2000.

[16] G. Fang, W. Gao, and D. Zhao. Large vocabulary sign language recognition based on fuzzy decision trees. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 34(3):305–314, May 2004.

[17] M. Maebatake, I. Suzuki, M. Nishida, Y. Horiuchi and S. Kuroiwa. Sign language recognition based on position and movement using multi-stream hmm. In *Universal Communication, 2008. ISUC '08. Second International Symposium on*, pages 478–481, Dec 2008.

[18] Ł. Gadomer, M. Skoczylas. Real time gesture recognition using selected classifiers. *Architecturae et Atribus*, 6(1):14–18, 2014.

[19] Microsoft. Kinect for Windows. `http://www.microsoft.com/en-us/kinectforwindows/`, 30.03.2015.

[20] Microsoft. Kinect for Windows SDK v1.7. `http://www.microsoft.com/en-us/download/details.aspx?id=36996`, 08.05.2014.

[21] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

Łukasz Gadomer

ROZPOZNAWANIE GESTÓW W PRZESTRZENI TRÓJWYMIAROWEJ

Streszczenie W niniejszym artykule autor opisał kontynuację swoich badań dotyczących rozpoznawania gestów. Ulepszył on stworzone przez siebie rozwiązanie w taki sposób, aby nagrywanie i rozpoznawanie gestów było niezależne od szybkości ich wykonywania, a co za tym idzie — ich zróżnicowanej długości. Zaproponował on także inne reprezentacje danych, za pomocą których wyrażany jest stworzony zbiór gestów. Wcześniejsze rozwiązanie, opierające się na przechowywaniu relatywnego położenia dłoni w stosunku do poprzedniej zarejestrowanej próbki (poprzedniego położenia), zastąpione zostało sprowadzeniem gestu do początku układu współrzędnych i zastąpieniem wartości relatywnych absolutnymi, a następnie ich normalizacją. Z tak przygotowanego zbioru gestów obliczone zostały cechy stanowiące drugą zaproponowaną reprezentację danych. Trzecia reprezentacja stanowi połączenie dwóch poprzednich: zawiera jednocześnie bezpośrednie wartości wyrażające ruch dłoni, jak i obliczone na podstawie jego cechy. Wszystkie trzy reprezentacje zostały przetestowane przy pomocy klasyfikatora, który okazał się najlepszy dla zadanego problemu podczas przeprowadzania wcześniejszych badań: SVM. Porównano, jak z zadanym problemem radzą sobie cztery popularne funkcje jądra: liniowa, wielomianowa, sigmoidalna i radialna. Otrzymane wyniki zostały przedstawione, porównane i omówione.

Słowa kluczowe: klasyfikacja danych, rozpoznawanie gestów, cechy, przestrzeń trójwymiarowa, niezależne od prędkości, niezależne od położenia, kinect

ASSESSMENT OF LDAP SERVICES IN HIGH AVAILABILITY ENVIRONMENT

Marcin Marchel[1,2], Cezary Boldak[1]

[1] Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

[2] Transition Technologies, Warsaw, Poland

Abstract: In this work we assess functioning of an LDAP service in the High Availability Environment. A system configuration with two alternative existing open source LDAP implementations (OpenLDAP and Apache Directory Server) was examined. Pacemaker/Corosync tool were employed here to manage two distributed resources: virtual main IP address and two cloned LDAP services. The test system was deployed on the LDAP production servers at Faculty of Computer Science, Bialystok University of Technology. Then several tests were run to measure the time of various operations (initialization, read/write, switchover, failover) as well as to verify the continuity of working and data consistency in presence of diverse faults, including power supply off. Few low level problems (due to used tools) were encountered and solved.

Keywords: LDAP, high availability, data consistency, Pacemaker/Corosync, OpenLDAP, Apache Directory Server

1. Background

LDAP (Lightweight Directory Access Protocol) [3] is an Internet Protocol that provides access to information that is organized in the hierarchical form (tree-structured). It is also commonly used as an external service for user authentication and authorization in numerous information systems. OpenLDAP [9] and Apache Directory Server [10] are well known open source LDAP implementations. Both of them provide built-in replication functionalities in two modalities: *refresh-only* – where consumers (replicas) periodically synchronize their states with the provider (replicas initiate the synchronization) and *refresh-and-persist* – where the consumers (after initial registration with the provider) are synchronized immediately after the provider is modified (the provider initiates the synchronization).

High availability (HA) is a term that refers to the information systems with protections against possible unwanted downtime to your system caused by hardware or software malfunctions (faults) [13,14]. The desired availability level (period of correct functioning without failure – statistically estimated or empirically calculated in a longer term) depends on a domain of the system usage, but for the most critical domains on can strive for single seconds of downtime per year (99.99999 % for ultra-avaliability) [5]. "No-dowtime" is probably only the theoretical level.

The key technique for handling such malfunctions is redundancy [4]. According to this assumptions, the defective part of the infrastructure (e.g. server, service) will be automatically replaced with the other spare one available in the pool, which until the fault occurrence might not serve any significant features. When the replaced resource is statefull, all redundant copies must be kept consistent (data replication). But the redundancy is only a pre-requisite to the high availability – further system organization to recognize and deal with faults is necessary to assure better availability rate [5].

High-availability mechanisms refer to functionalities that help to increase its reliability. High-availability mechanisms are:

– **Failover** is the mechanism that performs his job at the time when a lack of access to the resource is detected. This mechanism performs a switch to another available resource that should be mirrored copy of the resource that has failed.
– **Switchover** mechanism is aimed at getting the same as its automatic *failover* equivalent. The difference between these mechanisms lies in the fact that the *switchover* requires the intervention of a person (such as a system administrator) to perform the switch node or restart the software.
– **Switchback** is a mechanism that involves the restoration of network traffic to a node with a higher priority than that which is currently supporting the resource. This mechanism assumes that higher priority node was unavailable for some period of time for any reason.
– **Heartbeat** is the mechanism that for specified period of time (interval) sends information to defined system objects of its availability. Depending on the information returned by this mechanism defined actions will be performed.

There exist technologies/tools to facilitate implementation of the high availability paradigm. One of them is Pacemaker/Corosync [11,12], offering low level solutions (heartbeat, distributed resource definition and management) to the the high available system developer .

2. Related works

The literature study did not reveal many works on assuring the high availability in hierarchical databases (LDAP). [1] proposes a solution (BFT-LDAP) to overcome Byzantine Faults to keep all the LDAP replicas consistent, but does not take into account the necessity of the high availability of the service, for instance when a physical machine is gone. [2] presents a very detailed study on the OpenLDAP performance (time measures for several LDAP operations) in different hardware and software configurations in order to achieve the optimum response time and throughput. But only one OpenLDAP instance was examined, not replicated nor distributed, and without component faults, what is of the main concern in this work.

Much more sources treat high availability in standard relational databases. [6] describes a highly available configuration of the MySQL database using Heartbeat, rsync and internal MySQL replication functionality, but no verification of the proposed solution was performed. [7] builds a "highly available" PostgreSQL configuration in the cloud using the Threshold Based File Replication technique. The server replication and automatic, load-balanced request routing seem to fulfill the HA requirements, but more interests there is oriented toward the load-balancing concerns (practical experiments) while the system resiliency in presence of faults is not studied at all. [8] also proposes a highly available transactional database system (using Oracle and Tuxedo tools). This work focuses on redundant components (hardware), database replication, parallel server/clustering and transactional replication. Several test scenarios (even complex ones) were prepared and performed. They consisted of: database failure, server process crash, network failure and verified the correct functioning of the system in their presence. The response times for 10,000 messages were measured (however not reported in details), but not times of other operations (e.g. failover).

3. Test system configuration

Our test system is based on redundancy of the run LDAP servers, controlled by the the Pacemaker/Corosync software suite, to provide the high availability of the LDAP service and fault tolerance (verified experimentally in Section 4.) to several malfunctions. The redundant LDAP servers are kept synchronized by the internal mechanisms of the two analyzed LDAP implementations.

The system needs at least two machines (real or virtual), but can be further grown up to a larger number, to take into account more failed nodes. In our experimental configuration we used two nodes:

- **ldap-one** - Main node with *IP1*,
- **ldap-two** - Backup node with *IP2*.

The two IP addresses *IP1* and *IP2* are not exposed to clients – they are used only for internal resource management. Instead, the third address *IPglobal*, bound to a DNS name, is offered to external systems (by means of the bound DNA name) – see below.

The Pacemaker/Corosync environment (installed in the distributed manner on all nodes) defined two managed resources.

1. **Virtual IP Address** (*IPglobal*): This resource was configured with higher priority assigned to the ldap-one node. It means that in case of 2 nodes availability, the *IPglobal* address will be assigned to **ldap-one** node (which is the real machine, while **ldap-two** is the virtual one). This resource was working in Active/Passive mode.
2. **LDAP Resource**: This resource (representing the run and redundant LDAP servers) had to be cloned for every node. It means that it should be active all the time on every node defined in the cluster (working in Active/Active mode). There was some additional work done by us to create a new OCF (Open Cluster Framework) [15] resource agent for Apache Directory Server, because it was not provided out of the box. OCF resource agent for OpenLDAP is available as out of the box solution for *Pacemaker*. Creating new resource agent for *Pacemaker* made us possible to perform examinations for both ApacheDS and OpenLDAP implementations.

Detecting a node failure (loss) and switching to the backup node are managed with the Pacemaker/Corosync suite but recovering to the initial (fully operational) configuration can be more complicated. Some faults (local LDAP process, system restart) are to be dealt with the used tool (by restarting, what is sufficient in many cases [5]), but others (system stop, power off, machine error) need a manual intervention. In this case, after detecting the permanent loss of one node, the system should report this abnormal state to the administrator. Such behavior can be enabled by adding *ocf:pacemaker:ClusterMon* resource. This resource provides notifying about cluster events, which can be received in a few different ways like email/SNMP (Simple Network Management Protocol) notifications or by using external agent (shell script).

Two different LDAP implementations were examined independently: OpenLDAP and Apache Directory Server. Both of them were configured accordingly to their technical documentation to the *refresh-and-persist* replication. This mode is closer to assure the strong data consistency (all modifications of the main node are immediately propagated to the backup ones), while the second mode (*refresh-only*) realizes the eventual data consistency (changes are propagated periodically, so with a latency) [4].

Figure 1 presents functioning of the test system. External users (systems) use the domain name resolved by the DNS to *IPglobal*. This address, as the managed distributed resource, is assigned to the only one node from the pool. It is worth noticing that the entire system works in the Active/Pasive model and the redundant replicas do not serve the client requests. It could be further changed to the Active/Active model, implementing some load balancing technique [14] to increment the system throughput. The connection between the LDAP resources means the replication mechanism.

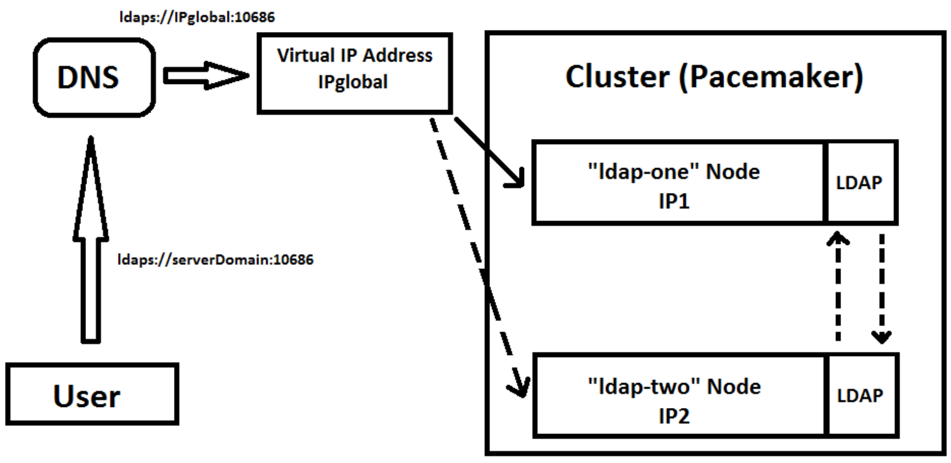


Fig. 1. Architecture of the test system.

4. Experiments

The main goals of the prepared tests were to:

– verify the failover functioning under different malfunctions,
– measure the response time of basic LDAP operations, in normal conditions and in presence of faults,
– verify the data consistency after the failover server switch.

All the tests were performed on the two selected LDAP implementations (OpenLDAP 2.4 and Apache Directory Server 2.0.0-M20) independently, so the additional goal was to compare these two technologies regarding all the three mentioned above aspects.

The test environment was set up on the production servers at the Faculty of Computer Science, Bialystok University of Technology, Poland. A cluster of two nodes was built, one located on a real machine, second on a virtual one. A third machine in the same network was used to run all the test scripts. Tests were performed without using the LDAPS secure protocol. All experiments were run 10 times (except stated differently) and min, max and mean statistics are presented.

Test plan

We planned and ran four types of tests.

1. **Measuring times of read and write operations in normal circumstances.** The main goal was to have the reference for the following experiments (scaling). Additionally the read and written information was located on different tree levels to check if it influenced the response times.
2. **Measuring times of initialization/restoration of the cluster functioning** after a single node loss and in case of the entire cluster restarting.
3. **Verifying the failover functioning and measuring its times**. Only one node (active) failure was simulated. As this mechanism is driven by the heartbeat messages, experiments were conducted for several their intervals.
4. **Verifying the data consistency in the event of active node failure**. LDAP unavailability was caused by different reasons: killing the LDAP service local process, controlled node disconnection from the cluster, controlled machine shutdown, pulling out the power plug.

4.1 Read/write times (no faults present)

Tests presented in this chapter are designed to check how fast it is possible to save and read a specified data set from/to the LDAP server. This examination also assumes that no node failure would not occur during performing it. An important aspect of this examination is the depth of the tree where the information is stored: 10, 30 50, 100.

Read times

This examination assumed preforming 10,000 read operations. Every read operation was done on a different entry from the set of 1,000 entries and on the specified tree depth to minimize usage of the memory cache mechanism. The previously mentioned entries were located at four different depths of the tree structure (10, 30, 50, 100). Every such the depth contained its own set of 1,000 entries. We used default cache settings for both ApacheDS and OpenLDAP. The algorithm shown below presents steps of our shell script to calculate time required to perform 10,000 read operations:

1. **Start** the time counter.
2. In the **loop** of 10,000 iterations:
   (a) generate DN for the current loop iteration,
   (b) execute read operation (*ldapread*) – repeat it until success if failed occasionally.
3. **Stop** the time counter. Calculate time required for performed operations and display result.

Results in Table 1 present times for both LDAP implementations and take into account different depths of the tree structure. Based on the examination it can be concluded that OpenLDAP offers 7% higher read speed than ApacheDS. It is also worth noticing that depth of information in a tree structure does not significantly affect the speed of reading.

Table 1. Read times (in seconds) in function of the tree depth (no faults).

Depth of a tree structure	10		30		50		100	
	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP
Minimum	203.49	188.81	204.99	192.15	209.39	192.33	222.16	197.08
Maximum	211.21	195.14	211.07	202.46	219.42	197.20	230.37	206.87
Average	206.08	193.03	207.55	195.36	212.89	194.73	225.36	200.49

Write times

This examination assumed performing 10,000 write operations. As before, the writes modified data at different depth of the tree structure. This script assumes existence of the defined entry on the specified tree depth, which contains *Description* attribute. Every write operation performs incrementation of that attribute's value. The algorithm given below presents steps of our shell script to calculate the time required to perform 10,000 write operations:

1. **Start** the time counter.

2. In the **loop** of 10,000 iterations:
   (a) generate DN for the write operation using the current loop iteration context,
   (b) perform the write operation (*ldapmodify*) – repeat it until success if failed occasionally.
3. **Stop** the time counter. Calculate time required for performed operations and display result.

Results in Table 2 present times for both LDAP implementations and take into account different depths of the tree structure. Based on the examination it can be concluded that OpenLDAP implementation is able to write information about 38 times faster than ApacheDS. It can be partially acceptable (in the case of ApacheDS) in scenarios where the data is much more frequently read then written. Here again, the tree depth has only minor influence on the performance.

Table 2. Write times (in seconds) in function of the tree depth (no faults).

Depth of a tree structure	10		30		50		100	
	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP
Minimum	7410.86	193.12	7317.77	209.62	7423.12	205.68	7653.84	209.97
Maximum	7498.54	240.16	7351.10	241.95	7478.83	239.48	7714.01	237.50
Average	7455.31	213.42	7331.34	224.85	7450.69	224.08	7685.88	227.71

4.2 Times of initialization/restoration the cluster functionality

The examinations presented in this chapter were executed to verify how fast will the developed system restore its full availability after occurrence of two different fault types: loss of all the nodes and loss of the active node. The algorithm shown below presents steps that were done for both test cases. The script implementing it is executed on the external node.

1. In a **loop** wait until LDAP becomes inaccessible - it is performed by executing *ldapsearch* command and checking the returned status. In meanwhile the manual step is done – pacemaker/corosync services are restarted on the active node.
2. **Start** the time counter.
3. *ldapsearch* command is executed in a second **loop** finished once LDAP becomes accessible again.
4. **Stop** the time counter and print a message with the calculated time period.

After a failure of all the nodes

That examination concerned the system behavior after failure of all two available nodes (forced cluster software issue). The algorithm presented above was executed in the normal cluster configuration, where two nodes had active (by cloning) LDAP resource. Data in Table 3 presents the examination results for two chosen LDAP implementations. Based on them, it can be concluded that OpenLDAP becomes accesible 40% faster than ApacheDS. The latter showed a high variance, with observations going from 22.11 to 52.49.

Table 3. Times (in seconds) needed to restore LDAP functionality (all nodes failed in the same time).

LDAP	ADS	OpenLDAP
Minimum	22.11	23.88
Maximum	52.49	25.38
Average	39.27	24.38

After a failure of the active node

That examination concerned the system behavior after failure of the only one available node that hosted the LDAP resource. The algorithm presented above was executed in the abnormal cluster configuration, where only one node was working (e.g. after a *failover* or *switchover* event). Data in Table 4 presents examination results for two chosen LDAP implementations. Based on the examination it can be concluded that OpenLDAP becomes accesible about 40% faster than ApacheDS.

Table 4. Times (in seconds) needed to restore LDAP functionality (active node failed).

LDAP	ADS	OpenLDAP
Minimum	40.39	23.69
Maximum	42.80	32.48
Average	41.03	25.51

4.3 Examination of the failover functioning

This examination verified one of the main issues of this work: if the automatic switching to the backup node (*failover*) functioned at all. It also involved setting various time intervals for the heartbeat module to examine its influence of this mechanism speed. This test assumed that LDAP process would be manually killed on the active node. Measured time showed how fast LDAP resource became available again. Used algorithm shown below presents steps that were done (the script implementing it was executed on the external computer).

1. In a **loop** wait until LDAP becomes inaccessible, its performed by executing *ldapsearch* command and checking returned status – in meanwhile the manual step is done: LDAP process (or 2 processes in case of ApacheDS) is killed on the active node .
2. **Start** the time counter.
3. *ldapsearch* command is executed in a second **loop** finished once LDAP becomes accessible again, what is verified by use of *ldapsearch* command.
4. **Stop** the time counter an print a message with the calculated time period.

Data placed in Table 5 presents examination results for the chosen LDAP implementations. Based on the examination it can be concluded that:

- *failover* worked all the times,
- heartbeat period affected the *failover* time in nearly linear manner (Figure 2),
- LDAP implementation did not affect the *failover* time.

Table 5. Times (in seconds) needed to restore LDAP functionality in function of the heartbeat period.

Heartbeat configuration	3		10		15		20	
	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP
Minimum	0.59	0.18	2.31	0.25	1.28	0.29	2.03	4.94
Maximum	2.98	3.04	9.12	9.88	13.29	14.89	23.19	28.66
Average	1.49	1.93	5.19	5.58	6.86	7.29	9.47	15.77

4.4 Examination of the data consistency after a failure of the active node

After confirming the *failover* functioning, this examination aimed probably the most important goal for the proposed solution: verification of the data consistency when

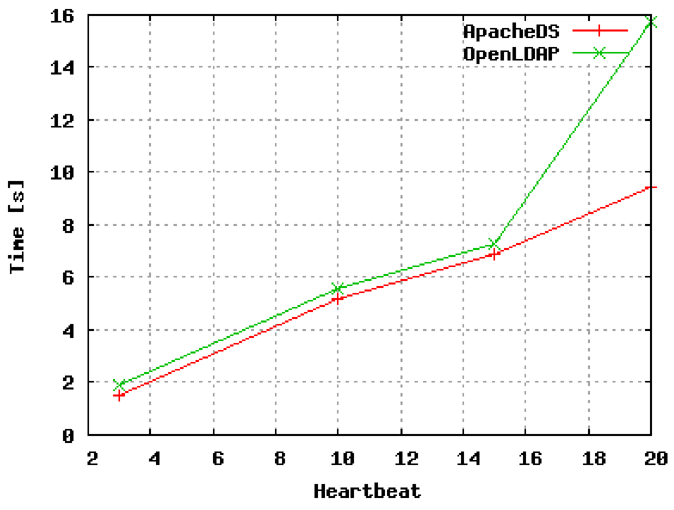


Fig. 2. Failover times in function of the heartbeat period.

a fault (consisted of a failure of the active node) occurred during the data sending process (*refresh-and-persist* replication model was used).

This test was performed in four variants differing in the malfunction nature:

- **Controlled disconnecting of the main node from the cluster**, which was performed by restarting the Pacemaker/Corosync services on the active node.
- **Killing the LDAP process**, which was performed by executing a shell script that was able to automatically find LDAP process in memory and kill that process (two processes in case of ApacheDS) on the active node.
- **Controlled server shutdown**, which was perfomed using *shutdown* command on the active node.
- **Plugging off the power plug**, which was performed by manually plugging off the power plug on the active node.

We used the algorithm presented below and composed of the following steps.

1. **Set** a local counter (*LC*) variable value to 1.
2. **Write** *LC* (*ldapmodify*) to the LDAP tree - it becomes a remote counter (*RC*).
3. In a **loop** (in meanwile one of the faults described above is produced):
   (a) **read** *RC* and **calculate** delta $D = LC - RC$ (it shows how many data modifications were not replicated due to the active node failure – if it occurred in that loop iteration),

(b) **if** $D \neq 0$ **then**: **display** a message about the event and **exit** the shell script,
(c) **increment** *LC*,
(d) **increment** *RC* (by *ldpapmodify*).

The results in Table 6 presents examination results for two chosen LDAP implementations. There was no data loss/inconsistency detected – our system for the both chosen LDAP implementations **passed this test**.

Table 6. Numbers of the data loss cases (data inconsistency).

Fault	Controlled disconnecting node from cluster		Killing LDAP process		Controlled server shutdown		Pulling the power plug	
	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP
Minimum	0	0	0	0	0	0	0	0
Maximum	0	0	0	0	0	0	0	0
Average	0	0	0	0	0	0	0	0

4.5 Encountered problems

We believe that an important aspect of our work is also to inform about problems that were encountered in the studies.

We noticed an unexpected replication mechanism behavior during performing examinations for both selected LDAP implementations. This problem occurred in case of disconnecting power supply for the server and then turning the server on again. The problem was caused by the replication mechanism being broken – it stopped working bidirectionally. The only way to fix this issue was to restart LDAP on the node, which was previously working as the backup one. This problem occurred when the *refresh-and-persist* replication mode was chosen. In case of the *refresh-only*, this problem was not occurring. The workaround for it was to extend initially created OCF resource script for ApacheDS to be able to detect such situation and perform restarting cluster services on the node that caused the problem.

Another type of issue that was noticed during the testing was a problem with commands like *ldapsearch* and *ldapmodify*. This issue was about hanging process in case when destination LDAP server got offline during command execution. In such the case an additional parameter related to the process timeout was not working as it was expected. This issue was noticed only for the ApacheDS implementation. The workaround for it was to use eternal shell script that was used to detect and kill hanging *ldapsearch* or *ldapmodify* process.

5. Conclusions

This work resulted in the experimental assessment of functioning of the cluster system realizing the highly available LDAP service, in normal working and in presence of different faults . These examinations, thanks to using two different LDAP implementations, allowed us to make more general observations about behavior of the LDAP services in High Availability Environment in similar configurations.

Performed operations aimed to experimentally demonstrate the most important features of high availability on example of the developed system. They responded to the question whether the selected LDAP implementations are able to maintain consistency of data between server instances in the case of a node failure. In the case of any of the examined implementations data loss has not occurred at the time of the main node failure. In our opinion, in the case of large production environments where read/write operations from/to LDAP resource can be counted in thousands per second, the result may be different than the one obtained in our studies. Particularly in the case of a decidedly slower write operation when using the ApacheDS implementation.

On the basis of the studies we additionally described four characteristics, which in our opinion defines perfect LDAP implementation oriented for high availability.

1. **The quality of replication mechanism**: this is an important feature which can ensure that data consistency will be kept in case of cluster's node loss.
2. **The speed of writing information**: this is an important feature for the replication mechanism because it can directly cause to not replicate the information in the case of node failure (provider), which updates the information to other nodes (consumers).
3. **The speed of reading information**: this is an important feature for replication mechanism when you need to query the LDAP instance for specific information.
4. **The speed of the service startup**: this is particularly important in the case of configuration, when a node becomes available only at the time of the root node loss. In such case, the time it takes to boot up the LDAP server is the time which delays the start of the entire cluster environment.

On the basis of those studies, we observed that the OpenLDAP implementation is more stable in the case when high availability is required. However, during our experiments, negative influence of the lower ApacheDS performance have not been encountered, but we do not exclude such a possibility in a longer time period usage.

This work can be continued in several directions. More LDAP implementations can be examined to prove the generality of the system. The system availability can be even increased by including (and further experimental verification) more backup nodes. The system performance can be boosted by routing client requests to all the nodes, not only the main one, with the load balancing techniques [14]. It would be also interesting to observe and measure the system behavior and availability in longer term (months or years) to empirically classify our solution to the concrete availability class [5].

Acknowledgment

This work was supported by the Bialystok University of Technology with the grant S/WI/2/2013.

References

[1] Ali Shoker, Jean-Paul Bahsoun: Towards Byzantine Resilient Directories, IEEE 11th International Symposium on Network Computing and Applications, 2012.

[2] Xin Wang, Member, Henning Schulzrinne, Fellow, Dilip Kandlur, and Dinesh Verma: Measurement and Analysis of LDAP Performance, IEEE/ACM Transactions On Networking, Vol. 16, No. 1, pp. 232–243, 2008.

[3] Franco Milicchio, Wolfgang Alexander Gehrke: Distributed Services with OpenAFS. Springer Science & Business Media, 2007.

[4] Andrew S. Tanenbaum, Maarten Van Steen: Distributed Systems. Principles and Paradigms (2nd Edition). Prentice-Hall, 2006.

[5] J. Gray, D.P. Siewiorek: High-availability computer systems. Computer, Vol. 24, No. 9, pp. 39–48, 1991.

[6] V. Chaurasiya, P. Dhyani, S. Munot: Linux Highly Available (HA) Fault-Tolerant Servers. Information Technology, (ICIT 2007). 10th International Conference on, pp. 223–226, 2007.

[7] S. Pippal, S. Singh, R.K. Sachan, D.S. Kushwaha: High availability of databases for cloud. Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on pp. 1716–1722, 2015.

[8] S. Budrean, Y. Li, B. C. Desai: High Availability Solutions for Transactional Database Systems. Proceedings of the Seventh International Database Engineering and Applications Symposium (IDEAS'03) 2003.

[9] OpenLDAP Software 2.4 Administrator's Guide, Site: `http://www.openldap.org/doc/admin24/OpenLDAP-Admin-Guide.pdf`, Access time: 19 October 2015

[10] ApacheDS 2.0 Advanced User Guide, Site: `https://directory.apache.org/apacheds/advanced-user-guide.html`, Access time: 19 October 2015

[11] Andrew Beekhof: Pacemaker 1.1, Configuration Explained, An A-Z guide to Pacemaker's Configuration Options, Site: `http://clusterlabs.org/doc/en-US/Pacemaker/1.1-pcs/pdf/Pacemaker_Explained/Pacemaker-1.1-Pacemaker_Explained-en-US.pdf`, Access time: 19 October 2015
[12] Andrew Beekhof: Pacemaker 1.1, Clusters from Scratch, Step-by-Step Instructions for BuildingYour First High-Availability Cluster, Site: `http://clusterlabs.org/doc/en-US/Pacemaker/1.1-pcs/pdf/Clusters_from_Scratch/Pacemaker-1.1-Clusters_from_Scratch-en-US.pdf`, Access time: 19 October 2015
[13] Theo Schlossnagle: Scalable Internet Architectures Sams Publishing, 2006.
[14] P. Membrey, E. Plugge, D. Hows: Practical Load Balancing, Ride the Performance Tiger Apress, 2012.
[15] Florian Haas: The OCF Resource Agent Developer's Guide, Site: `http://www.linux-ha.org/doc/dev-guides/ra-dev-guide.html`, Access time: 20 October 2015

OCENA USŁUG KATOLOGOWYCH LDAP W ŚRODOWISKU WYSOKIEJ DOSTĘPNOŚCI

Streszczenie W niniejszej pracy oceniono działanie usług katalogowych LDAP w środowisku wysokiej dostępności. Wzięto pod uwagę dwie implementacje LDAP o otwartym kodzie: OpenLDAP i Apache Directory Server. W celu zarz adzania dwoma rozproszonymi zasobami (wirtualny adres IP i dwie sklonowane usługi LDAP) użyto narzędzia Pacemaker/Corosync. Testowa konfiguracja został wdrożona na serwerach produkcyjnych LDAP na Wydziale Informatyki Politechniki Białostockiej. W dalszej kolejności przeprowadzono testy w celu pomiaru czasów różnych operacji (inicjalizacji, odczytu/zapisu, zaplanowanego i awaryjnego przeł aczenia serwerów) jak również w celu zweryfikowania ci agłości pracy i spójności danych w przypadku różnego rodzaju awarii, w tym zaniku zasilania. Konieczne było rozwi azanie kilku technicznych problemów zwi azanych z użytymi narzędziami.

Słowa kluczowe: usługi katalogowe LDAP, wysoka dostępność, spójność danych, Pacemaker/Corosync, OpenLDAP, Apache Directory Server

Artykuł zrealizowano w ramach pracy badawczej nr S/WI/2/2013.

COMPARISON OF SELECTED TRADE CLASSIFICATION ALGORITHMS ON THE WARSAW STOCK EXCHANGE

Joanna Olbryś[1], Michał Mursztyn[2]

[1] Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

[2] Student of Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: Empirical equity market microstructure research often requires knowledge about trade sides. It is important to recognize the side that initiates the transaction and to distinguish between the so called buyer- and seller-initiated trades. Unfortunately, most data sets do not identify the trade direction. Therefore, researchers rely on indirect trade classification rules to infer trade sides. The aim of this paper is to implement and compare four trade classification algorithms on the Warsaw Stock Exchange (WSE). The high-frequency data for the selected WSE-listed companies covers the period from January 3, 2005 to June 30, 2015. To check the robustness of empirical results to the choice of sample, three adjacent sub-periods of equal size: the pre-crisis, crisis, and post-crisis period are analysed. The Global Financial Crisis on the WSE is formally established as the period June 2007-February 2009.

Keywords: trade classification algorithms, intraday data, Polish stock market

1. Introduction

The issue of trade direction is important in the market microstructure literature. For example, in order to measure dimensions of liquidity, intra-day liquidity, and imbalance on the order-driven market we have to recognize the side initiating the transaction and to distinguish between the so called buyer- and seller-initiated trades. The classification indicates which of the two participants in the trade, the buyer or the seller, is more eager to trade. When such a distinction does not exist, the trade is labeled as indeterminate [28]. The Warsaw Stock Exchange (WSE) is classified as an order-driven market with an electronic order book, but the information of the best bid and ask prices is not publicly available, (e.g. [16,21]). In fact, even the non-proprietary financial databases that provide information on trades and quotes do not identify the trade direction. As a consequence, the researchers rely on indirect trade classification rules to infer trade sides [2]. There are some trade classification procedures described in the literature.

The main goal of this paper is to implement and to compare four trade classification algorithms: the quote rule (QR), the tick rule (TR), the Lee and Ready (LR) [18] and the Ellis, Michaely and O'Hara (EMO) [11] procedures, using high-frequency data for the selected WSE-listed stocks. Based on the paper [16], we have chosen three representative firms from the size groups, specifically: the KGH[3] from the BIG group, the MCI[4] from the MEDIUM group, and the ENP[5] from the SMALL group. We utilize the KGH intra-day data, as the KGH is one of the most liquid WSE-companies. The levels of liquidity for the MCI and ENP are also relatively high in comparison to the securities from the corresponding size groups, e.g. [23]. The comparison of trade classification rules both in the whole sample January 3, 2005-June 30, 2015 and over three adjacent sub-periods of equal size (436 days) [16]: (1) the pre-crisis period September 6, 2005-May 31, 2007, (2) the crisis period June 1, 2007-February 27, 2009, and (3) the post-crisis period March 2, 2009-November 19, 2010, is provided. The Global Financial Crisis on the WSE was formally set based on the papers [24,25], in which the method for direct statistical identification of market states was employed. The empirical results are novel and, to the best of the authors' knowledge, have not been presented in the literature thus far.

The remainder of the study is organized as follows. Section 2 presents a brief literature review concerning the motivation and applications of trade classification algorithms. Section 3 specifies the algorithms employed in the research. In Section 4, we describe the empirical experiment on the Warsaw Stock Exchange. We implement and compare four algorithms using the intra-day transaction data for three WSE-listed companies. The last section encompasses the conducted research with a brief summary.

2. Inferring trade direction: motivation and applications

Traditional capital market approach treats the market aggregation process as a 'black box'. News and information are processed in some obscure fashion and prices adjust to reflect the aggregate effect of this information [19]. However, recent developments

[3] KGHM Polska Miedź S.A.
[4] MCI Management S.A.
[5] ENAP Energoaparatura S.A.

in market microstructure research are beginning to unravel this black box. The increasing availability of intra-day data is opening new frontiers for financial market research, also for the use of observed trades and quotes. The study of observed trades and quotes enables us to obtain new insights about market participants. Unfortunately, most data sets do not identify trade sides. In this context, inferring trade direction from intra-day data is especially worthwhile to conduct because of its applications for a comprehensive analysis of market liquidity.

The main motivation for this study is provided by the growing interest in market liquidity, dimensions of liquidity, and commonality in liquidity that has emerged in the literature over the recent years. As the nature of liquidity is multidimensional, the interpretation of market liquidity causes some problems. A common approach consists in breaking up liquidity into three or four components. Some authors propose three dimensions of liquidity: (1) tightness, (2) depth, and (3) resiliency, e.g. [8], but usually the following four aspects or dimensions are distinguished: (1) trading time, (2) market tightness, (3) market depth, and (4) market resiliency, (e.g. [16,29,33]). However, the analysis of proxies of market dimensions requires the use of high-frequency transaction data, especially including the information about the trade direction. Similarly, the proxies of many intra-day liquidity measures are calculated based on trade and quotes data, (e.g. [6,14,15,23,29,31,33]). Moreover, the order imbalance indicators are approximated based on the information about the number, volume and value of transactions initiated by each side of the market (e.g. [6,7,21]). Notice here that there exists a large empirical financial literature employing the trade classification algorithms on international stock markets, (e.g. [1,2,3,4,5,6,7,9,10,11,12,13,14,17,18,19,20,22,26,27,28,30,32]). On the contrary, research conducted on the Polish stock market is rather scarce, e.g. [21].

3. Trade classification algorithms

The goal of trade classification is to correctly determine the initiator of the transaction [22]. However, a formal definition of a trade initiator is rarely stated in the literature. For example, the so called 'immediacy' definition describes initiators as traders who demand immediate execution, e.g. [19]. According to [22], the initiator of a transaction is the investor (buyer or seller) who placed his/her order last, chronologically (the so called 'chronological' definition). The two definitions are equivalent in many cases. In both definitions, the initiator is the person who caused the transaction to occur. As mentioned in Introduction, there are some trade classification procedures described in the literature: the tick rule, the reverse tick rule, the quote rule, the at the quote rule, the revised quote rule, the Lee-Ready (LR) algorithm, the revised LR algorithm, the Ellis-Michaely-O'Hara (EMO) algorithm, and the Bulk Volume Classification (BVC) methodology. The content of each classification rule is described as follows:

- The tick rule is based on price movements relative to previous trades. If the transaction is above (below) the previous price, then it is a buy (sell). If there is no price change but the previous tick change was up (down), then the trade is classified as a buy (sell) (e.g. [18,20]).
- The reverse tick rule uses the next trade price to classify the current trade. If the next trade occurs on an up-tick or zero up-tick, the current trade is classified as a sell. If the next trade occurs on a down-tick or zero down-tick, the current trade is classified as a buy, (e.g. [18,20]).
- The quote rule classifies a transaction as a buy if the associated trade price is above the midpoint of the bid and ask. If the trade price is below the midpoint quote, then the trade is classified as a sell (e.g. [18,20]).
- The at the quote rule classifies a transaction as a buy if the associated trade price is traded at the asking price. If the trade price is at the bidding price, then the trade is classified as a sell [20].
- The revised quote rule considers the problem of 'no bid or no offer quote'. The trade would be classified as a buy if there is only the bid-side quote and it would be classified as a sell if there is offer-side quote only [20].
- The LR algorithm [18] is a combination of the quote rule and the tick rule. In the first stage the trade is classified according to the quote rule. In the second stage the midpoint transaction is classified according to the tick rule (details in table 1).
- The revised LR algorithm [20] which first adjusts the 'no bid or no offer quote' problems, and then classifies a trade according to the quote rule and finally the tick rule.
- The algorithm proposed by Chakrabarty, Li, Nguyen and Van Ness [2] which is a hybrid of the tick and quote rules. It uses the quote rule when transaction prices are closer to the ask and bid, and the tick rule when transaction prices are closer to the midpoint.
- The EMO algorithm [11] classifies the trades by means of the at the quote rule first and then the tick rule (details in table 1).
- The BVC algorithm [10] aggregates trades over short time intervals or volume intervals and then uses the standardized price change between the beginning and the end of the interval to approximate the percentage of buy and sell volume. Unlike traditional trade classification algorithms that assign trades to be either buys or sells, the BVC approach apportions trades into buy volume and sell volume.

The rules employed in the empirical experiment on the WSE are described in details in table 1. P_t denotes the transaction price at time t. The midpoint price P_t^{mid} at time t is calculated as the arithmetic mean of the best ask price $P_t(a)$ and the best bid price $P_t(b)$ at time t:

$$P_t^{mid} = \frac{P_t(a) + P_t(b)}{2}. \tag{1}$$

The implementation of the quote rule, the tick rule, the LR and EMO procedures is presented in detail (Algorithms 1-4). The accuracy of trade classification algorithms has been examined in many studies, (e.g. [1,2,3,4,10,11,12,18,19,20,22,32]).

Among others, Lee and Radhakrishna [19] report that the active-side of each trade, as identified by the LR rule [18], is generally a good proxy for the frequency, size, and direction of incoming market orders. They use the TORQ[6] database which contains data on NYSE[7] stocks. The authors find that for those trades that can be classified, the LR algorithm is 93% accurate.

Odders-White [22] investigates the performance of the quote, tick and LR methods, using the TORQ data for NYSE stocks. She finds that the quote rule performs relatively well on the transactions that it classifies, misclassifying only 9.1% of the transactions in the sample. The tick rule misclassifies 21.4% of the transactions, while the LR algorithm misclassifies only 15.0% of the transactions. Moreover, she reports success rates of 75%, 79%, and 85%, for the quote, tick and LR rules, respectively.

Ellis, Michaely and O'Hara [11] examine the validity of several trade classification methods for NASDAQ[8] trades. They find that the quote rule, the tick rule, and the LR [18] rule correctly classify 76.4%, 77.66%, and 81.05% of the trades, respectively. The authors develop a new trade classification algorithm (the so called EMO rule, see table 1). For sorting trades into buys and sells, their algorithm achieves an overall success rate of 81.9% in the NASDAQ market.

Finucane [12] employs the tick test, the reverse tick test, and the LR method to classify trades as buys or sells, and he compares the results to the direction of the actual orders, using the TORQ database. The author concludes that for NYSE firms, the tick rule and the LR method have very similar performance accuracy in classifying trades, while the reverse tick test performs substantially worse.

Theissen [32] analyses the accuracy of the tick test and the LR trade classification algorithm on the Frankfurt Stock Exchange (FSE). He reports that the LR method classifies 72.8% of the transactions correctly, and the tick test performs al-

[6] TORQ - Trades, Orders, Reports, and Quotes
[7] NYSE - New York Stock Exchange
[8] NASDAQ - National Association of Securities Dealers Automated Quotations

most equally well. The author stresses that the accuracy of the LR method on the FSE is limited.

Lu and Wei [20] investigate the applicability and accuracy of many trade classification methods on the Taiwan Stock Exchange. They employ the tick, reverse tick, quote, at the quote, and revised quote rules, as well as the LR and the EMO algorithms. The authors proposed their own classification rule - the revised LR algorithm.

However, although a number of alternative algorithms have been developed, the Lee-Ready [18] algorithm remains the most frequently used [3].

Table 1. The quote rule (QR), the tick rule (TR), the Lee-Ready (LR) and Ellis-Michaely-O'Hara (EMO) trade classification algorithms

Rule	Conditions	
	Trade is classified as buyer-initiated	Trade is classified as seller-initiated
QR	If $P_t > P_t^{mid}$	If $P_t < P_t^{mid}$
	If $P_t = P_t^{mid}$ then a trade is not classified.	
TR	If $P_t > P_{t-1}$	If $P_t < P_{t-1}$
	In case $P_t = P_{t-1}$ trade is signed using the previous transaction price. If the sign of the last non-zero price change is positive (negative) then the trade is classified as a buy (a sell).	
LR	I stage	
	If $P_t > P_t^{mid}$	If $P_t < P_t^{mid}$
	If $P_t = P_t^{mid}$ then:	
	II stage	
	If $P_t^{mid} > P_{t-1}$	If $P_t^{mid} < P_{t-1}$
	When $P_t^{mid} = P_{t-1}$, the decision is taken using the sign of the last non-zero price change P_{t-k}. If $P_t > P_{t-k}$ then it is a buy, if $P_t < P_{t-k}$ then it is a sell.	
EMO	I stage	
	If $P_t = P_t(a)$	If $P_t = P_t(b)$
	II stage	
	Trades with prices different from bid and ask prices are categorized by the tick rule. P_t is compared to P_{t-1}: If $P_t > P_{t-1}$ then it is a buy, if $P_t < P_{t-1}$ then it is a sell.	

4. The empirical experiment on the Warsaw Stock Exchange

The high-frequency data 'rounded to the nearest second' available at www.bossa.pl was utilized. The data set contains the opening, high, low and closing (OHLC) prices for the security over one unit of time. Following [21], the transaction price P_t at time

Algorithm 1 The QR trade classification function

```
Require: data - file with the list of transactions from the earliest to the latest; each transaction is written
    in a separate line;
    line - string that contains transaction data and must possess following format:
    $[Company\_name], 0, [Date : YYYYMMDD], [TimeOfDay : HHMMSS], O, H, L, C, V;$
    date - string that contains date of the previous trade;
Ensure: The QR trade classification function itself returns 3 possibilities:
    1, when trade is classified as buyer-initiated;
    0, when trade is not classifiable;
    -1, when trade is classified as seller-initiated.
    int QR(double open, double high, double low, double close):
 1: double $mid = (high + low)/2$
 2: if $close > mid$ then
 3:     return 1
 4: end if
 5: if $close < mid$ then
 6:     return -1
 7: end if
 8: return 0
```

t was approximated by the close price. Considering that the bid and ask prices are not public information on the WSE, the midpoint price P_t^{mid} at time t was rounded by the arithmetic mean of the lowest price P_t^L and the highest price P_t^H at time t which approximated the best ask price and the best bid price respectively. Then Eq. (1) becomes Eq. (2):

$$P_t^{mid} = \frac{P_t^L + P_t^H}{2}. \tag{2}$$

On the trading days during the period from January 3, 2005 to June 30, 2015 there are 3 959 406 transactions in the KGH data set, 530 697 transactions in the MCI data set, and 87 005 transactions in the ENP data set. Every transaction is assigned using the QR, TR, LR and EMO trade classification algorithms (see Tab. 1). The opening trade is treated as unclassified in the TR, LR and EMO procedures [17]. Of course, there is inevitably some assignment error [6]. Yet, as shown e.g. in [3,11,12,19,22], the proposed algorithms are accurate enough as not to pose serious problems in our large sample study. Tables 2-4 contain empirical results of the comparison of the QR, TR, LR and EMO procedures for the KGH, MCI and ENP intra-day transaction data, in the whole sample and three investigated sub-periods, respectively.

The empirical results presented in Tables 2-4 indicate that the TR and LR algorithms are more appropriate compared to the QR and EMO procedures on the WSE. In the case of the TR and LR methods, the percentage of unclassified transactions is

Algorithm 2 The TR trade classification function

Require: $data$ - file with the list of transactions from the earliest to the latest; each transaction is written in a separate $line$;
$line$ - string that contains transaction data and must possess following format:
$[Company\_name], 0, [Date : YYYYMMDD], [TimeOfDay : HHMMSS], O, H, L, C, V;$
$date$ - string that contains date of the previous trade;
$closes[]$ - dynamic array of previous close prices; it is cleared when next day is to be processed; array allows to insert elements at the end of array $(.push(double))$, clear entire array $(.clear())$ and checking its size $(.size())$;

Ensure: The TR trade classification function itself returns 3 possibilities:
1, when trade is classified as buyer-initiated;
0, when trade is not classifiable;
-1, when trade is classified as seller-initiated.

```
    int TR(double $open$, double $high$, double $low$, double $close$, double[]$closes$):
 1: for $i \leftarrow closes.size() - 1$ to 0 do
 2:     double $pt = closes[i]$
 3:     if $close - pt < 0$ then
 4:         return -1
 5:     end if
 6:     if $close - pt > 0$ then
 7:         return 1
 8:     end if
 9: end for
10: return 0
```

relatively low. The amount of buyer- and seller-initiated trades is almost equal, with a little predominance of buyer-initiated in all investigated periods. This evidence is consistent with the literature, as some papers demonstrate that short sales are sometimes misclassified as buyer-initiated by trade classification algorithms, e.g. [1]. On the contrary, the applicability and accuracy of the QR and EMO rules is rather low, with a high percentage of unclassified trades in the KGH, MCI and ENP data sets. It is worthwhile to note that the EMO method was proposed for the NASDAQ, which is a hybrid market, while the Warsaw Stock Exchange is classified as an order-driven market. The probable explanation of discrepancies in trade classification results between markets is that stock market structure and trading mechanisms may affect the accuracy of trade classification algorithms.

5. Conclusion

The validity of many market microstructure studies depends on the ability to accurately classify trades as buyer- or seller-initiated. Despite the importance of trade

Table 2. Performance of the QR, TR, LR and EMO trade classification algorithms for the KGH (the BIG group) transaction data from the period January 3, 2005-June 30, 2015

Rule	Total number of trades	Percentage of buyer-initiated trades	Percentage of seller-initiated trades	Percentage of unclassified trades
QR				
whole sample	3 959 406	4.14	4.44	91.42
pre-crisis	390 257	2.03	2.52	95.45
crisis	502 404	5.50	5.37	89.13
post-crisis	635 692	5.45	5.66	88.89
TR				
whole sample	3 959 406	51.13	48.53	0.34
pre-crisis	390 257	50.86	48.36	0.78
crisis	502 404	51.11	48.63	0.26
post-crisis	635 692	52.04	47.78	0.18
LR				
whole sample	3 959 406	51.16	48.51	0.33
pre-crisis	390 257	50.86	48.37	0.77
crisis	502 404	51.17	48.58	0.25
post-crisis	635 692	52.06	47.76	0.18
EMO				
whole sample	3 959 406	4.42	4.15	91.43
pre-crisis	390 257	2.52	2.03	95.45
crisis	502 404	5.37	5.50	89.13
post-crisis	635 692	5.65	5.45	88.90

Table 3. Performance of the QR, TR, LR and EMO trade classification algorithms for the MCI (the MEDIUM group) transaction data from the period January 3, 2005-June 30, 2015

Rule	Total number of trades	Percentage of buyer-initiated trades	Percentage of seller-initiated trades	Percentage of unclassified trades
QR				
whole sample	530 697	6.18	6.49	87.33
pre-crisis	93 563	6.57	7.43	86.00
crisis	93 475	7.90	8.42	83.68
post-crisis	111 170	6.53	6.30	87.17
TR				
whole sample	530 697	50.54	47.70	1.76
pre-crisis	93 563	52.11	46.44	1.45
crisis	93 475	50.32	48.57	1.11
post-crisis	111 170	50.74	48.40	0.86
LR				
whole sample	530 697	50.90	47.36	1.74
pre-crisis	93 563	52.09	46.46	1.45
crisis	93 475	50.79	48.14	1.07
post-crisis	111 170	51.35	47.82	0.83
EMO				
whole sample	530 697	6.46	6.20	87.34
pre-crisis	93 563	7.43	6.57	86.00
crisis	93 475	8.40	7.92	83.68
post-crisis	111 170	6.27	6.60	87.13

Table 4. Performance of the QR, TR, LR and EMO trade classification algorithms for the ENP (the SMALL group) transaction data from the period January 3, 2005-June 30, 2015

Rule	Total number of trades	Percentage of buyer-initiated trades	Percentage of seller-initiated trades	Percentage of unclassified trades
QR				
whole sample	87 005	6.25	6.94	86.81
pre-crisis	37 902	6.15	7.00	86.85
crisis	18 962	7.45	7.99	84.56
post-crisis	8 040	5.65	5.46	88.89
TR				
whole sample	87 005	49.66	44.26	6.08
pre-crisis	37 902	51.93	45.41	2.66
crisis	18 962	50.07	45.70	4.23
post-crisis	8 040	47.43	42.95	9.62
LR				
whole sample	87 005	49.78	44.23	5.99
pre-crisis	37 902	51.92	45.45	2.63
crisis	18 962	50.34	45.53	4.13
post-crisis	8 040	47.77	42.77	9.46
EMO				
whole sample	87 005	6.84	6.21	86.95
pre-crisis	37 902	6.99	6.14	86.87
crisis	18 962	7.90	7.43	84.67
post-crisis	8 040	5.16	5.57	89.27

Algorithm 3 The LR trade classification function

Require: *data* - file with the list of transactions from the earliest to the latest; each transaction is written in a separate *line*;
line - string that contains transaction data and must possess following format:
$[Company\_name], 0, [Date : YYYYMMDD], [TimeOfDay : HHMMSS], O, H, L, C, V;$
$closes[]$ - dynamic array of previous close prices; it is cleared when next day is to be processed; array allows to insert elements at the end of array $(.push(double))$, clear entire array $(.clear())$ and checking its size $(.size())$;
Ensure: The LR trade classification function itself returns 3 possibilities:
1, when trade is classified as buyer-initiated;
0, when trade is not classifiable;
-1, when trade is classified as seller-initiated.

```
int LR(double $open$, double $high$, double $low$, double $close$, double[]$closes$):
double $mid = (high + low)/2$
if $close > mid$ then
    return 1
else
    if $close < mid$ then
        return -1
    else
        double $pt = close$
        for $i \leftarrow closes.size() - 1$ to 0 do
            if $pt - closes[i] < 0$ then
                return -1
            else
                if $pt - closes[i] > 0$ then
                    return 1
                end if
                $pt = closes[i]$
            end if
        end for
        return 0
    end if
end if
```

classification to economic research, the available data generally does not contain this information. As pointed out earlier, the trade direction is not public information on the Warsaw Stock Exchange.

The main aim of this paper was to implement and to compare the QR, TR, LR and EMO trade classification algorithms using high-frequency data for the WSE-listed stocks. The KGH, MCI, and ENP were chosen as the representative Polish companies from three size groups. The empirical experiment shows that the TR and LR procedures perform better on the WSE, while the applicability and accuracy of

Algorithm 4 The EMO trade classification function

```
Require: data - file with the list of transactions from the earliest to the latest; each transaction is written
    in a separate line;
    line - string that contains transaction data and must possess following format:
    $[Company\_name], 0, [Date : YYYYMMDD], [TimeOfDay : HHMMSS], O, H, L, C, V;$
    prevclose - variable that remembers close price of the previous trade; it is set to -1 when a new
    trading day is to be processed.
Ensure: The EMO trade classification function itself returns 3 possibilities:
    1, when trade is classified as buyer-initiated;
    0, when trade is not classifiable;
    -1, when trade is classified as seller-initiated.
    int EMO(double open, double high, double low, double close, double prevclose):
 1: if $close == low == high$ then
 2:     return 0
 3: else
 4:     if $close == low$ then
 5:         return 1
 6:     end if
 7:     if $close == high$ then
 8:         return -1
 9:     end if
10: end if
11: if $prevclose == -1$ then
12:     return 0
13: end if
14: if $close > prevclose$ then
15:     return 1
16: else
17:     if $close < prevclose$ then
18:         return -1
19:     else
20:         return 0
21:     end if
22: end if
```

the QR and EMO methods is rather low. Moreover, the empirical results turned out to be robust to the choice of the sample and rather do not depend on a firm size.

One of the possible directions for further investigation would be to implement the BVC algorithm [10] and to compare it with the TR and LR procedures, following for example the methodology proposed in [4]. To the best of the authors' knowledge, such research has not been conducted on the WSE thus far.

Acknowledgments

This work was supported by Bialystok University of Technology under the grant S/WI/1/2014.

References

[1] Asquith, P., Oman, R., Safaya, C.: Short sales and trade classification algorithms, Journal of Financial Markets 13(1), 2010, pp. 157-173.

[2] Chakrabarty, B., Li B., Nguyen, V., Van Ness, P.A.: Trade classification algorithms for electronic communications network trades, Journal of Banking and Finance 31(12), 2007, pp. 3806-3821.

[3] Chakrabarty, B., Moulton, P.C., Shkilko, A.: Short sale, long sale, and the Lee-Ready trade classification algorithm revisited, Journal of Financial Markets 15(4), 2012, pp. 467-491.

[4] Chakrabarty, B., Pascual, R., Shkilko, A.: Evaluating trade classification algorithms: Bulk volume classification versus the tick rule and the Lee-Ready algorithm, available SSRN-d2182819.pdf, 2014.

[5] Chordia, T., Roll, R., Subrahmanyam, A.: Commonality in liquidity, Journal of Financial Economics 56(1), 2000, pp. 3-28.

[6] Chordia, T., Roll, R., Subrahmanyam, A.: Order imbalance, liquidity, and market returns, Journal of Financial Economics 65, 2002, pp. 111-130.

[7] Chordia, T., Sarkar, A., Subrahmanyam, A.: An empirical analysis of stock and bond market liquidity, The Review of Financial Studies 18(1), 2005, pp. 85-129.

[8] Doman, M.: Mikrostruktura giełd papierów wartościowych, Poznań University of Economics Press, Poznań, 2011.

[9] Easley, D., López de Prado, M., O'Hara, M.: Flow toxicity and liquidity in a high-frequency world, Review of Financial Studies 25(5), 2012, pp. 1457-1493.

[10] Easley, D., López de Prado, M., O'Hara, M.: Bulk classification of trading activity, Johnson School Research Paper No. 8-2012.

[11] Ellis, K., Michaely, R., O'Hara, M.: The accuracy of trade classification rules: Evidence from Nasdaq, Journal of Financial and Quantitative Analysis 35(4), 2000, pp. 529-551.

[12] Finucane, T.J.: A direct test of methods for inferring trade direction from intraday data, Journal of Financial and Quantitative Analysis 35(4), 2000, pp. 553-576.

[13] Foster, F.D., Viswanathan, S.: Variations in trading volume, return volatility, and trading costs: Evidence on recent price formation models, Journal of Finance 48(1), 1993, pp. 187-211.

[14] Goyenko, R.Y., Holden, C.W., Trzcinka, C.A.: Do liquidity measures measure liquidity?, Journal of Financial Economics 92(2), 2009, pp. 153-181.

[15] Huberman, G., Halka, D.: Systematic liquidity, Journal of Financial Research 24(2), 2001, pp. 161-178.

[16] Jankowski, R., Olbryś, J.: Wymiary płynności rynku papierów wartościowych, Zeszyty Naukowe Uniwersytetu Szczecińskiego Nr 854. Finanse, Rynki Finansowe, Ubezpieczenia 73, 2015, pp. 645-658.

[17] Korajczyk, R., Sadka, R.: Pricing the commonality across alternative measures of liquidity, Journal of Financial Economics 87(1), 2008, pp. 45-72.

[18] Lee, C.M.C., Ready, M.J.: Inferring trade direction from intraday data, Journal of Finance 46(2), 1991, pp. 733-746.

[19] Lee, C.M.C., Radhakrishna, B.: Inferring investor behavior: Evidence from TORQ data, Journal of Financial Markets 3, 2000, pp. 83-111.

[20] Lu, Y.-C., Wei, Y.-C.: Classification of trade direction for an equity market with price limit and order match: Evidence from the Taiwan stock market, Investment Management and Financial Innovations 6(3), 2009, pp. 135-147.

[21] Nowak, S.: Order imbalance on the Warsaw Stock Exchange, 2000-2012, International Conference Financial Investments and Insurance-Global Trends and Polish Market, Wroclaw, Poland, 17-19 September, 2014.

[22] Odders-White, E.R.: On the occurrence and consequences of inaccurate trade classification, Journal of Financial Markets 3, 2000, pp. 259-286.

[23] Olbryś, J.: Wycena aktywów kapitałowych na rynku z zakłóceniami w procesach transakcyjnych, Difin Press, Warszawa, 2014.

[24] Olbryś, J., Majewska, E.: Direct identification of crisis periods on the CEE stock markets: The influence of the 2007 U.S. subprime crisis, Procedia Economics and Finance 14, 2014, pp. 461-470.

[25] Olbryś, J., Majewska, E.: Identyfikacja okresu kryzysu z wykorzystaniem procedury diagnozowania stanów rynku, Zeszyty Naukowe Uniwersytetu Szczecińskiego Nr 802. Finanse, Rynki Finansowe, Ubezpieczenia 65, 2014, pp. 699-710.

[26] Peterson, M., Sirri, E.: Evaluation of the biases in execution costs estimation using trades and quotes data, Journal of Financial Markets 6(3), 2003, pp. 259-280.

[27] Piwowar, M.S., Wei, L.: The sensitivity of effective spread estimates to trade-quote matching algorithms, Electronic Markets 16(2), 2003, pp. 112-129

[28] Plerou, V., Gopikrishnan, P., Gabaix, X., Stanley, H.E.: Quantifying stock price response to demand fluctuations, Physical Review 66(2), 2008, 027104(1-4).
[29] Ranaldo, A.: Intraday market liquidity on the Swiss Stock Exchange, Swiss Society for Financial Market Research 15(3), 2001, pp. 309-327.
[30] Sadka, R.: Momentum and post-earnings announcement drift anomalies: The role of liquidity risk, Journal of Financial Economics 80(2), 2006, pp. 309-349.
[31] Stoll, H.R.: Friction, Journal of Finance 55(4), 2000, pp. 1479-1514.
[32] Theissen, E.: A test of the accuracy of the Lee/Ready trade classification algorithm, Journal of International Financial Markets, Institutions and Money 11(2), 2001, pp. 147-165.
[33] von Wyss, R.: Measuring and predicting liquidity in the stock market, Dissertation Nr. 2899, University of St. Gallen, 2004.

PORÓWNANIE WYBRANYCH ALGORYTMÓW KLASYFIKACJI TRANSAKCJI NA GIEŁDZIE PAPIERÓW WARTOŚCIOWYCH W WARSZAWIE S.A.

Streszczenie Badania empiryczne w zakresie mikrostruktury rynku często wymagają wiedzy na temat stron transakcji. Szczególnie ważna jest identyfikacja strony inicjującej transakcję oraz podział na transakcje inicjowane przez nabywcę i sprzedającego. Niestety, większość baz danych intraday nie zawiera takich informacji. Dlatego badacze korzystają z pośrednich metod klasyfikacji. Celem pracy była implementacja i porównanie czterech algorytmów klasyfikacji transakcji na Giełdzie Papierów Wartościowych w Warszawie S.A. Badanie przeprowadzono z wykorzystaniem śróddziennych danych transakcyjnych akcji trzech reprezentatywnych firm, należących do grup spółek dużych (grupa BIG), średnich (grupa MEDIUM) i małych (grupa SMALL). Były to odpowiednio spółki: KGH, MCI oraz ENP. Wszystkie charakteryzują się stosunkowo wysoką płynnością na tle grup, do których należą. Analizy objęły okres od 3 stycznia 2005 do 30 czerwca 2015, z wyróżnieniem trzech jednakowo licznych podokresów: przed kryzysem, kryzys, po kryzysie. Okres globalnego kryzysu finansowego na giełdzie warszawskiej został ustalony w sposób formalny jako przedział czasowy czerwiec 2007-luty 2009.

Słowa kluczowe: algorytmy klasyfikacji transakcji, dane intraday, polski rynek kapitałowy

Artykuł zrealizowano w ramach pracy badawczej S/WI/1/2014.

PARAMETERS TUNING OF EVOLUTIONARY ALGORITHM FOR THE ORIENTEERING PROBLEM

Krzysztof Ostrowski

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: Various classes of algorithms solving optimization problems have some set of parameters. Setting them to appropriate values can be as important to results quality as choosing right algorithm components. Parameter calibration can be a complex optimization problem itself and many meta-algorithms were proposed to deal with it in a more automatic way. This paper presents automatic parameter tuning of an evolutionary algorithm solving the Orienteering Problem. ParamsILS method was chosen as a tuner. Obtained results show the importance of appropriate parameter setting in evolutionary algorithms: tuned algorithm achieved very high-quality solutions on known Orienteering Problem benchmarks.

Keywords: parameter tuning, evolutionary algorithms, Orienteering Problem

1. Introduction

The name Orienteering Problem (OP) derives from sport game of orienteering. Competitors start from a given point and have to reach another point within a prescribed time frame. In the meantime they can visit other control points and collect scores. The winner is the competitor who finishes with the highest score. The OP is an NP-hard optimization problem [4] and belongs to the family of Travelling Salesman Problems with Profits (TSPP) [26]. Its alternative name is Selective Travelling Salesman Problem (STSP) [15]. The OP has many practical applications including tourist trip planning, logistics and others. Various approaches (both exact and approximate) were proposed to solve the OP but because of its computational hardness most published methods were approximate. Evolutionary algorithms (EA) are among best known metaheuristics for solving optimization problems. When it comes to results quality EA parameters setting can be as important as choosing recombination operators. The paper presents parameter tuning (with ParamsILS method [30]) of an evolutionary algorithm which solves the Orienteering Problem. The article is organized as follows.

Section 2 presents mathematical definition of the OP. Section 3 presents a review of the literature on the OP. Overview of parameter tuning methods is presented in section 4. In Section 5 the structure of the proposed evolutionary algorithm (EA) is described. Section 6 contains description of EA parameter tuning and experimental results. Conclusions are drawn and further work is suggested in Section 7.

2. Mathematical definition of the Orienteering Problem

Given a weighted graph (each edge has a non-negative cost and each vertex has some non-negative profit) the purpose of the Orienteering Problem (OP) is to find a simple path (or cycle) between a given pair of vertices (s - start vertex, e - end vertex) that maximizes total collected profit (sum of profits of visited vertices). The total cost of the path (sum of costs of visited edges) is limited by a given constraint (T_{max}). Let w_{ij} be a cost associated with edge (i, j) and p_i be a profit associated with vertex i. Let x_{ij} be a binary variable equal to 1 if a solution contains edge (i, j) and 0 otherwise. Let y_i be a binary variable equal to 1 if a solution contains vertex i and 0 otherwise. Let r_i be a position of vertex i in a solution - it is defined only for vertices included in a path. The OP can be formulated mathematically:

$$max \sum_{i \in V} \sum_{j \in V} (p_i \cdot x_{ij}) \tag{1}$$

$$\sum_{i \in V} \sum_{j \in V} w_{ij} \cdot x_{ij} \leq T_{max} \tag{2}$$

$$\sum_{j \in V} x_{sj} = \sum_{i \in V} x_{ie} = 1 \tag{3}$$

$$\forall_{k \in V \setminus \{s,e\}} (\sum_{i \in V} x_{ik} = \sum_{j \in V} x_{kj} \leq 1) \tag{4}$$

$$r_s = 1 \tag{5}$$

$$\forall_{i \in V} (y_i = 1 \Rightarrow 1 <= r_i <= n) \tag{6}$$

$$\forall_{i \in V,\ j \in V \setminus \{s\},} (x_{ij} = 1 \Rightarrow r_j = r_i + 1) \tag{7}$$

Maximization of total profit is associated with formula 1 and constraint 2 relates to maximal path cost which cannot exceed T_{max}. Constraint 3 indicates that the path starts in vertex *s* and ends in vertex *e* while formula 4 implies that all other vertices can occur at most once on the path. Remaining constraints guarantee that the solution is a continuous path without subcycles.

3. Literature review of the Orienteering Problem

The Orienteering Problem was introduced by Tsiligirides [3]. Because of its NP-hardness exact solutions for the OP are only feasible in short time for graphs with a small number of nodes. The exact algorithms applied for the OP are the branch-and-cut and branch-and-bound methods. Fischetti et al. [14] have provided an exact solution tested for graphs with up to 500 vertices, and Gendreau et al. [12] have used the branch-and-cut method to exactly solve examples with up to 300 vertices. Ramesh et al. [8] have extended the branch-and-bound method via Lagrangian relaxation and have used it to solve examples with up to 150 control points. Exact algorithms for the OP usually need large amount of time to solve medium or large instances. Therefore, for practical applications, many researchers propose approximate methods to tackle the OP, based on a variety of approaches.

Tsiligirides [3] presents his S-algorithm for the OP, based on the Monte Carlo method. The algorithm constructs a large number of routes and chooses the one with the maximum profit. Then a deterministic heuristic algorithm partitions the geographic area into concentric circles and restricts the routes allowed to the sectors defined by the circles.

Golden et al. [4] introduced a three-step iterative heuristic involving route construction using a greedy algorithm and a centre-of-gravity heuristic. Ramesh et al. [7] propose a four-phase heuristic. After the best solution is chosen from iterations over a set of three phases (node insertion, edge exchange and node deletion), a fourth phase is entered, in which an attempt is made to insert unvisited nodes into the tour.

Wang et al. [27] use a neural network approach to solve the OP. They derive an energy function and a learning algorithm for a modified, continuous Hopfield neural network. Chao et al. [11] introduced a two-step iterative heuristic consisting of initialization and improvement steps. Tasgetiren [16] worked out the first genetic algorithm for the OP. The algorithm included tournament selection, injection crossover and mutation using elements of the local search method (add, omit, replace and swap operators). He also proposed a function that penalized infeasible solutions.

Gendreau et al. [13] present a tabu search heuristic for the OP. The algorithm iteratively inserts clusters of nodes into the current tour or removes a chain of nodes. Compared to the previous approaches, this method reduces the likelihood of getting trapped in a local optimum. Tests performed by the authors on randomly generated instances with up to 300 nodes show that the algorithm yields very high-quality solutions.

Vansteenwegen et al. [29] developed the guided local search method (GLS) for the Team Orienteering Problem (TOP), which modifies other methods to reduce the likelihood of becoming trapped in a local optimum. The GLS meta-heuristic method yields satisfactory results for small-sized networks and is used in the Mobile Tourist Guide [32]. Solutions for the TOP generate m routes as results and total collected profit is maximized. Each node can be included at most one time in m resultant routes in the TOP. For $m = 1$ the GLS solves classic Orienteering Problem.

Schilde et al. [28] published two metaheuristics: Variable Neighbour Search (VNS) and Ant Colony Optimization (ACO). In relatively short execution time both algorithms achieved on benchmarks better average results than Chao's method [11] and GLS [29].

Souffriau et al. [31] applied Greedy Randomized Adaptive Search Procedure (GRASP) to solve the TOP. Campos et al. [35] successfully applied a path relinking (PR) method to GRASP to solve the OP. In the GRASP algorithm they propose an initial path containing only a starting and an ending vertex. Next, based on the ratio between greediness and randomness (four methods are possible), vertices are inserted one by one for as long as T_{max} is not exceeded. In the next step, the local search procedure (exchange and insert phase) attempts to reduce the length of the path and increase its total profit. In GRASPwPR a set of different solutions is created with the GRASP method, and path relinking is performed for each pair of solutions P and Q: the P path is gradually transformed into the Q by exchanging elements between P and Q. GRASPwPR results obtained on benchmark instances are one of the best among approximate solutions.

Author's Orienteering Problem research concentrated mainly on EA-based OP solutions for networks larger than known OP benchmarks (up to 908 vertices - network of cities in Poland) [33][34][36]. Obtained results showed advantage of evolutionary algorithms over known meta-heuristics (GLS, GRASP, GRASPwPR) for larger OP instances. In addition, experiments were also performed on Chao and Tsiligirides benchmark sets [39] and EA results (compared GLS and Chao methods) were promising. Author's purpose is to develop very effective metaheuristic not only for the OP but also for the Time-Dependent Orienteering Problem [38].

4. Overview of parameter tuning methods

Setting algorithm parameters *adhoc* can be troublesome and sometimes can result in poor quality of algorithm solutions. Various methods were proposed to deal with this issue in a more automatic way. The simplest tuners are based on sampling of multi-dimensional parameter space. After testing the algorithm on a chosen set of parameter vectors the best vector is selected. Each vector evaluation is associated with running the algorithm on one or more test instances. Simplest way of choosing parameter vectors is full factorial design: we choose a few characteristic values for each parameter (for example quartiles of values ranges) and evaluate all possible combinations of these values. If there are n parameters and number of characteristic values is k for each parameter then number of evaluations is k^n. Therefore, in some cases full factorial experiment is too time consuming. Other sampling methods were proposed like Latin Hypercube Sampling and Taguchi Orthogonal Arrays [10]. In the first method parameter ranges are divided into a given number of m intervals. Afterwards m parameter vectors are randomly chosen with one constraint: for a given parameter no pair of vectors can have values from the same interval. The second method also enables to reduce vector number and is similar to full factorial design. Randomly generated vectors should meet the following condition: for each pair of parameters all combinations of parameters values are used the same number of times. It enables to reduce number of sampled vectors compared to full factorial design.

Simple sampling methods are often used as starting procedures to more complex tuning algorithms. For example instead of using one sampling procedure a tuning method can repeat it multiply and use previous sampling results to narrow parameter search space to a more promising area. Examples of such methods are Empirical Modelling of Genetic Algorithms [17] and CALIBRA [23].

Another approaches use models to estimate results for some parameter vectors instead of time-consuming algorithm tests. After some tests a model can be constructed. Its construction base on algorithm results obtained from a sample of parameter vectors. Afterwards, results from other parameter vectors can be estimated. Probably the most popular is regression model [20]. Instead of using only this one-stage procedure it is usually better to compose it with further actions. For example Coy et al. [18] use model-based approach, which is followed by a local search procedure. First, full factorial design over the whole parameter space is used to construct a linear regression model and to determine the path of steepest descent. Afterwards, this path is followed by a local search procedure and new vectors are generated and tested until the best solution stops changing. Sequential Parameter Optimization (SPO) [19] is a multi-stage procedure which updates the model after each iteration. After initial model construction the most promising vectors are tested and the results are used to update the model. This process is repeated until a given, maximum number of tests is reached. Unlike Coy's method this procedure improves the model after each iteration.

Screening methods' purpose is to determine the best parameter vector from a given set by using as few tests as possible. They concentrate only on vectors that are most promising based on tests conducted so far. For example racing technique [21] bases on partial results obtained from tests conducted so far and at each step it discards parameter vectors that are not optimal according to statistical tests. Another approach (Iterative F-RACE) [24] repeats procedures of racing and model construction (multi-variate normal distribution) until a given number of tests is performed.

Finding parameter vectors that enable the algorithm to obtain high-quality results can be a difficult task. The objective function (algorithm results quality) in multi-dimensional parameter space is non-linear and can have multiple local optimums. In such situations approaches such as evolutionary algorithms (EA) or local search metaheuristics can be a good choice. In meta-EA a population consists of parameter vectors and the goal is to find an individual with highest meta-fitness function (solution quality of the tuned algorithm run with a given parameters set). Each evaluation in meta-EA is associated with running the tuned algorithm and therefore it can be very time consuming. One of the most advanced meta evolutionary algorithm is Relevance Estimation and Value Calibration of parameters (REVAC) [25]. It is a specific type of meta-EA where the population approximates the probability density function of the most promising areas of the parameter space. Furthermore, REVAC has been extended with Racing and Sharpening techniques in order to deal with the stochasticity of the utility values more effectively.

Parameters Iterated Local Search (ParamsILS) [30] is an example of local search approach to parameter tuning. After initial random sampling the best vector is chosen and a hill climbing procedure (first improvement) is applied to this vector until no better vector is found. Neighbourhood of a given vector v consists of vectors that differ from v only by one parameter value. After finding initial local optimum ParamILS repeatedly performs perturbation (random changes of some parameters) and hill climbing procedure. Perturbation enables to escape local optimum and to potentially search other optima. FocusedILS is a variant of ParamsILS that additionally uses racing when comparing parameter vectors.

5. Description of the proposed evolutionary algorithm

Proposed method for solving the Orienteering Problem is evolutionary algorithm with embedded local search operators. Path representation is used in the EA - each gene in a chromosome is equivalent to a path vertex. Path profit is treated as fitness value but infeasible solutions (paths longer than T_{max}) have zero fitness and are not allowed in the population. At first, an initial population of P_{size} random routes is created. Afterwards, evolutionary phase takes place - operators of selection, crossover, mutation and disturb are applied repeatedly. Mutation, crossover and disturb operators can be either random or greedy (local search embedded) and ratio between randomness and greediness can be adjusted with parameters. Evolutionary phase terminates after a fixed number N_g of generations, or earlier if there have been no improvements in the last C_g generations. After the evolutionary phase all paths in the population undergo final local improvement procedure. The best feasible path (with highest total profit) obtained during algorithm run is EA final result. Three different crossover operators and three different selection methods (overall nine different algorithm configurations) were implemented and tested.

5.1 Initialization

The algorithm starts by generating an initial population of P_{size} routes. Each route is encoded as a sequence of different vertices. Let s and e be the given starting and ending points of the routes. Initialization process inserts new, random vertices into the route (at its end but before e vertex) as long as its total cost does not exceed T_{max} constraint. Only non-included vertices that can be inserted without violating T_{max} are considered during selection. Construction of the route ends when no new vertex can be inserted without exceeding T_{max}. The result of the initialization process are P_{size} random routes not exceeding a given limit T_{max}.

5.2 Selection

Three different selection procedures were tested.

Unbiased tournament selection: This method was presented by Sokolov et al. [22]. P_{size} tournaments are applied and in each tournament two random individuals are selected (tournament size is 2). The winners of each of P_{size} tournaments (individuals with the highest fitness) form parents pool. To reduce selection noise the standard procedure was modified: a random permutation p without fixed points is generated and tournaments are formed by pairs $(i, p(i))$. Thanks to this modification every individual takes part in exactly two tournaments. During crossover phase (after selection) children replace parents and create new generation.

Fitness proportionate selection with stochastic universal sampling: During this procedure P_{size} individuals are selected and form parents pool. Each selection is based on fitness values and the probability of selecting individual i (fitness $f(i)$) is $\frac{f(i)}{F}$ where F is total fitness of whole population. In order to reduce selection bias standard roulette wheel was replaced by stochastic universal sampling [5]. Instead of P_{size} random samplings only one random value is used and individuals are selected from evenly spaced intervals. During crossover phase (after selection) children replace parents and create new generation.

Deterministic crowding: No parent selection is performed and selecting mechanism is applied after crossover procedures (survival selection). In deterministic crowding [9] each offspring competes with one of its parents and the fitter individual is chosen to next generation. To preserve population diversity competition is performed between more similar child-parent pairs. To measure differences between individuals edit distance is applied: it is computed by longest common subsequence (LCS) algorithm. The distance formula for paths i and j is

$$d(i,j) = L(i) + L(j) - 2 \cdot LCS(i,j) \tag{8}$$

where $L(i)$ is number of vertices in path i while $LCS(i,j)$ is the length of longest common subsequence of paths i and j. In order to speed up the algorithm initially size of vertex sets intersection is used instead of LCS in distance formula. If calculations suggests childA-parentB and childB-parentA competition pairs (instead of childA-parentA and childB-parentB pairs) then additional check is performed using LCS.

5.3 Crossover

First $\frac{P_{size} \cdot p_k}{2}$ different pairs of parents are randomly selected from the population (p_k - crossover probability) and each pair undergoes crossover procedure. Three crossover operators were tested:

2-point crossover: Crossover method applied by the author for the OP in [39] for the first time. At the beginning an ordered set S of common vertices for both parents is determined. Vertices order in the set is the same as in first parent. Next, chromosome fragments between two successive vertices in S are exchanged and two offspring are created. If any duplicates appear in offspring individuals outside of exchanged fragments they are immediately removed. There are two types of crossover: random (classic) version chooses crossover points randomly while greedy version maximizes fitness function of the fitter offspring. The ratio between greediness and randomness is determined by parameter z_k - it is equal to the probability of selecting greedy crossover operator. Therefore z_k=0 means purely random crossover procedure while z_k=1 means purely greedy procedure. In fig. 1 there is an example of 2-point crossover.

Injection crossover: This method was applied by Tasgetiren in his OP-solving genetic algorithm [16]. A random insertion point is chosen in one parent and a random fragment in the other parent. The chosen fragment is inserted into the first parent, duplicates are removed from it and its size is reduced to previous size by cutting some genes. In this way an offspring individual is created. Tasgetiren applied random crossover method while in this paper greedy version of this operator was also created: insertion point is chosen to maximize offspring fitness. Ratio between randomness and greediness is determined by parameter z_k in the same way as in 2-point crossover.

Path relinking crossover: Path relinking method was used by Campos et al [35] as procedure complementing GRASP algorithm. It was also applied in genetic algorithm solving Orienteering Problem with Time Windows (OPTW) by Karbowska et al. [37]. In this method one route is gradually transformed into the other by inserting and removing vertices. The best intermediate route is chosen in greedy version of crossover. In random crossover version randomly selected intermediate route becomes an offspring. Ratio between randomness and greediness is determined by parameter z_k in the same way as in previous crossover operators.

One crossover method (2-point) creates two offspring individuals while remaining methods create only one offspring. To compensate this difference injection and path relinking crossover are doubled (performed two times on ordered parent pairs: P_1-P_2 and P_2-P_1).

5.4 Mutation

First $P_{size} \cdot p_m$ individuals are randomly selected from the population (p_m - mutation probability). Each of them undergoes mutation. First 2-opt procedure is performed once - it tries to change two edges in the path to shorten the path as much as possible (without changing vertex set). Example of edge exchanges is illustrated in fig. 2. From all such possible edge exchanges 2-opt chooses the option which shortens the path most.

After 2-opt procedure one vertex insertion or one vertex deletion is carried out. Probabilities of both insertion and deletion are the same (0.5). Both procedures have parents

parent A: **1** - 3 - 9 - **7** - 4 - **5** - 12 - 11 - 14 - **16**

parent B: **1** - 6 - 8 - 2 - **7** - 13 - 10 - **5** - 15 - **16**

option I

offspring 1: **1** - 6 - 8 - 2 - **7** - 4 - **5** - 12 - 11 - 14 - **16**

offspring 2: **1** - 3 - 9 - **7** - 13 - 10 - **5** - 15 - **16**

option II

offspring 1: **1** - 3 - 9 - **7** - 13 - 10 - **5** - 12 - 11 - 14 - **16**

offspring 2: **1** - 6 - 8 - 2 - **7** - 4 - **5** - 15 - **16**

option III

offspring 1: **1** - 3 - 9 - **7** - 4 - **5** - 15 - **16**

offspring 2: **1** - 6 - 8 - 2 - **7** - 13 - 10 - **5** - 12 - 11 - 14 - **16**

Fig. 1. Example of 2-point crossover. Given two parents which have four common points (in bold: 1, 7, 5, 16) there are three possible fragment exchanges between successive points. Option I shows children created by fragments exchange between vertices 1 and 7, option II illustrates children created by segments exchange between points 7 and 5 while in option III one can see offspring individuals made by exchange of fragments between vertices 5 and 16. Random variant of crossover chooses random option while greedy variant chooses the option with highest fitness value of the better child.

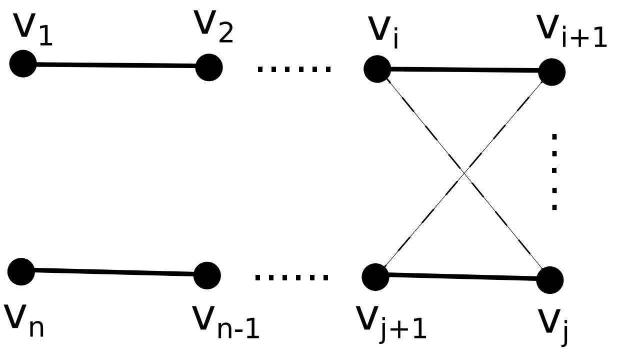


Fig. 2. Example of 2-opt edge change. For a path $(v_1, v_2, ..., v_i, v_{i+1}, ..., v_j, v_{j+1}, ..., v_n)$ two edges $((v_i, v_{i+1})$ and $(v_j, v_{j+1}))$ are replaced by two other edges $((v_i, v_j)$ and (v_{i+1}, v_{j+1}) - green) and this exchange results in a path $(v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+1}, v_{j+1}, ..., v_n)$.

two types: greedy (local search) and random. Greedy versions of insert and delete methods chose most favourable vertices in terms of their profits and change of path cost. During greedy insert all non-present vertices and all insertion points are considered and the option with highest ratio $\frac{ProfitIncrease}{CostIncrease}$ is chosen. Similarly local search used in delete procedure removes the vertex which minimizes ratio $\frac{ProfitDecrease}{CostDecrease}$. Random versions choose randomly selected vertices for insertion/deletion. However, random insert procedure adds new, random vertex in the insertion point which minimizes route cost increase. The ratio between greediness and randomness is determined by parameter z_m in the same way as by parameter z_k in crossover phase. Mutation examples are shown in fig. 3 and 4. Mutation operators do not allow paths to exceed T_{max} constraint. In order to speed up local search procedure (in insertion mutation) for a given number of vertex pairs (i, j) ordered lists of vertices are stored - they are sorted by $\frac{ProfitIncrease}{CostIncrease}$ heuristic associated with inserting a given vertex between vertices i and j.

5.5 Disturb

Disturb procedure is a different type of mutation and causes bigger changes in routes. First $P_{size} \cdot p_z$ individuals are randomly selected from the population (p_z - disturb probability). Each of them undergoes disturb procedure which removes some route fragment (but no more than 10 percent of route vertices). Again there are two types of disturb procedure: greedy variant from all fragments of a given length chooses the one which minimizes ratio $\frac{ProfitDecrease}{CostDecrease}$ while random variant chooses route segment randomly. The ratio between greediness and randomness is determined by parameter

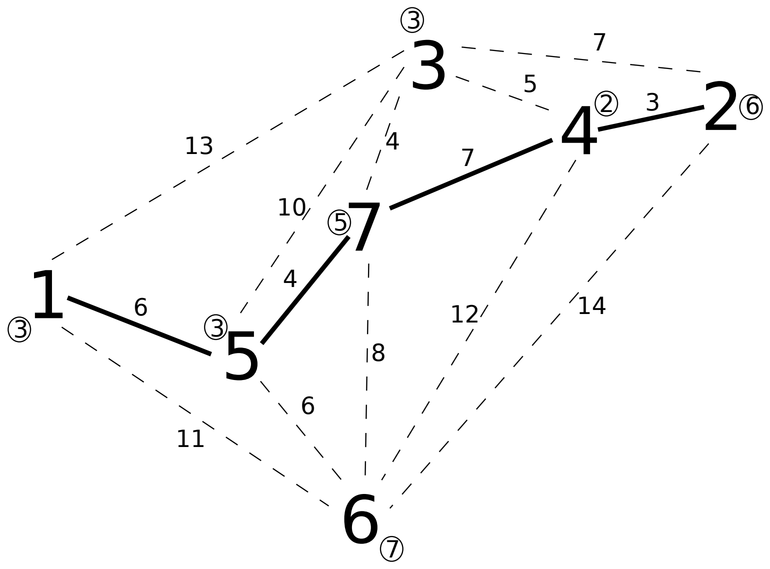


Fig. 3. Example of inserting mutation. Given a graph with 7 vertices and a path (1, 5, 7, 4, 2) there are two vertices not present in the path (3 and 6). Each of them can be inserted into one of 4 insertion points. That gives a total of 8 insertion options. Greedy insertion chooses the option maximizing $\frac{ProfitIncrease}{CostIncrease}$ ratio - inserting vertex 3 between vertices 7 and 4 gives the best ratio of 1.5 (profit increase is 3 and path cost increase is 2). Random insertion chooses random vertex and inserts it into a place which minimizes path cost increase (either vertex 3 between vertices 7 and 4 or vertex 6 between vertices 5 and 7. Graph weights are given beside edges and profits beside vertices (in circles).

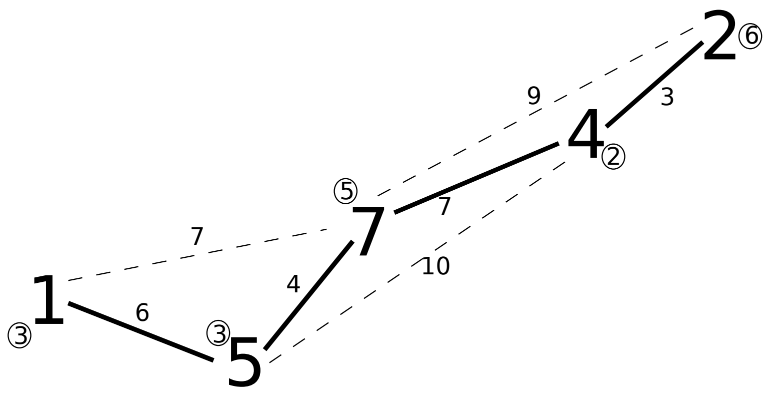


Fig. 4. Example of deleting mutation. Given a path (1, 5, 7, 4, 2) there are three options of vertex deletion (5, 7 or 4). Greedy deletion chooses the option minimizing $\frac{ProfitDecrease}{CostDecrease}$ ratio - removing vertex 5 gives the best ratio of 1.0 (profit decrease is 3 and path cost reduction is 3). Random deletion chooses random vertex from all candidates. Graph weights are given beside edges and profits beside vertices (in circles).

z_z. This procedure enables routes to escape from local optimum and gives a chance to explore a different fragment of solution space.

5.6 Local improvement phase

After evolutionary phase is over each path undergoes final, local improvement procedure. First a series of 2-opt procedures is performed on a given path. Afterwards one greedy vertex deletion and a series of greedy vertex insertions are performed until T_{max} is reached (insertions and deletions described in mutation section). This process is repeated until no improvement is found. This procedure also tries to improve the best path found by the algorithm during evolutionary phase by performing a series of greedy insertions until T_{max} is reached. In practice, for fine-tuned EAs this phase usually gives only small improvement.

6. Parameter tuning of EA

After choosing algorithm components parameter tuning was carried out. This process was performed for each of 9 EA configurations (3 different selections x 3 different crossovers).

6.1 Tuning methodology

Params Iterated Local Search (ParamsILS) [30] algorithm was used for parameter tuning because of its simplicity and lower time consumption compared to meta-EAs. This meta-algorithm searches multi-dimensional parameter space using local search procedures. Parameter vectors (meta-solutions) processed by ParamsILS are evaluated by averaging results from multiple runs of the tuned EA. In the first phase N vectors are randomly chosen and the best of meta-solutions is selected for hill-climbing procedure. In this local improvement phase neighbourhood of the current solution is searched in a random order and the current solution is updated if a better neighbour is found. Hill-climbing procedure is performed as long as current solution has any better neighbour. Afterwards disturb procedure occurs: random S parameters of the best found meta-solution are changed (escaping from local optimum) and hill-climbing is performed again. Procedures of disturb and hill-climbing are iterated until a specific number of evaluations (or amount of time) is reached (during this experiment time limit for tuning was set to 4 hours). In some iterations current solution is again randomly initialized (large jump in solution space) - the probability of random restart is $P_{restart}$. Neighbourhood of parameter vector v is defined as all vectors that differs from v only on one parameter. In this case parameter discretization is needed. In table 1 there are descriptions of all tuned parameters and their set of values used during tuning procedure. Therefore meta-algorithm operates on 6-dimensional parameter space. Disturb procedure of the tuned EA has additional, small values of probability because it modifies large parts of routes and can be destructive to solutions quality when overused. Some basic EA parameters were set to specific values (table 2). Population size is a compromise between exploration ability and computation time. Parameters associated with generations number were determined during earlier experiments and should not stop EA prematurely.

Table 1. Tuned parameters of EA

parameter	**description**	**values set**
p_k	crossover probability	{0, 0.1, 0.3, 0.5, 0.7, 0.9, 1}
p_m	mutation probability	{0, 0.1, 0.3, 0.5, 0.7, 0.9, 1}
p_z	disturb probability	{0, 0.01, 0.03, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 1}
z_k	crossover greediness/randomness ratio	{0, 0.2, 0.4, 0.6, 0.8, 1}
z_m	mutation greediness/randomness ratio	{0, 0.2, 0.4, 0.6, 0.8, 1}
z_z	disturb greediness/randomness ratio	{0, 0.2, 0.4, 0.6, 0.8, 1}

Table 2. Set parameters of EA

parameter	**description**	**value**
P_{size}	population size	100
Ng	maximum number of generations	5000
Cg	maximum number of generations without improvement	500

In order to better explore meta-solution space first phase of the algorithm was modified. For each parameter p (with values range $<p_{min}, p_{max}>$) two subranges of values were defined: lower subrange $<p_{min}, \frac{p_{min}+p_{max}}{2})$ and upper subrange $<\frac{p_{min}+p_{max}}{2}, p_{max}>$. If number of parameters is n then parameter space can be divided into 2^n parts. Two vectors belong to the same part only if all of their parameters belong to the same subrange. In the first part of the meta-algorithm 2^6 random vectors from different parts of parameter space are selected. After initial selection meta-algorithm works in the way described above. Parameters of tuner are presented in table 3.

Table 3. Parameters of meta-algorithm

parameter	**description**	**value**
N	number of initial random vectors	64
S	number of parameters changed during disturb phase	2
$P_{restart}$	probability of random restart of current solution	0.3

Table 4. All problem instances with T_{max} values

Class	Instance name	T_{max}	Class	Instance name	T_{max}
	kroA100	10641		eil101A	158
	kroB100	11071		cmt121A	137
	kroC100	10375		cmt151A	175
	kroD100	10647		cmt200A	191
	kroE100	11034		gil262A	595
	rd100	3955		eil101B	315
	eil101	315		cmt121B	273
	lin105	7190	II	cmt151B	350
	pr107	22152		cmt200B	382
	gr120	3471		gil262B	1189
	pr124	29515		eil101C	472
	bier127	59141		cmt121C	409
	pr136	48386		cmt151C	525
	gr137	34927		cmt200C	573
I	pr144	29269		gil262C	1784
	kroA150	13262			
	kroB150	13065			
	pr152	36841			
	u159	21040			
	rat195	1162			
	d198	7890			
	kroA200	14684			
	kroB200	14719			
	ts225	63322			
	pr226	40185			
	gil262	1189			
	pr264	24568			
	pr299	24096			
	lin318	21015			
	rd400	7641			

6.2 Problem instances

All OP instances (divided into two classes) are presented in table 4. Class I tests are Travelling Salesman Problem (TSP) instances adapted to OP by Fischetti et al. [14]. They come from TSPLIB library (XML format of distance matrices) created by Reinelt [6] and profits of vertices were generated by Fischetti according to pseudo-random formula:

$$p_i = 1 + (7141 \cdot i + 73)(mod\, 100) \tag{9}$$

where p_i is profit of vertex i. T_{max} values for class I instances were set as 50 percent of shortest hamiltionian cycles. Distances between vertices were truncated to integer numbers.

Class II tests were Vehicle Routing Problem (VRP) instances adapted to OP by Fischetti et al [14]. Some of them (eil101, gil262) come from TSPLIB library while other instances were created by Christofides et al. [2]. In these instances customer demands from VRP were interpreted as vertices profits in the OP. T_{max} values were set as 25, 50 and 75 percent of shortest hamiltonian cycles. Depending on T_{max} value instances have additional letter (A, B or C) in their names. For each instance from both classes number of vertices is encoded into its name. Distances between vertices were rounded to nearest integers. For all instances of both classes paths start and end in vertex 1.

6.3 Tuning results

All experiments (calibration and testing) were carried out on Intel i7 3.6 GHz processor. Programs were implemented in C++ and executed in Linux operating system. Calibration process was carried out on three problem instances: pr299 and rd400 (from class I) as well as gil262 (from class II). From all OP instances with known exact solutions these three were hardest to obtain close to optimal solutions for published methods and for evolutionary algorithm during earlier tests. Evaluation of a given parameter vector consisted of 30 EA runs (10 runs for each of the calibration networks) and the result was average gap to the optimal solution. The gap was calculated as $100 \cdot (1 - \frac{P_{alg}}{P_{opt}})$ where P_{alg} is route profit obtained by EA while P_{opt} is profit of optimal route. Tuning process was performed for nine different EA configurations (various types of selection and crossover).

In table 5 calibration results are presented for nine different EA configurations. It can be seen that generally the hardest instance (in terms of obtained results) is rd400 - large number of vertices (400) is one of reasons for biggest gaps among three calibration networks. In terms of average gap the best results were obtained when

Table 5. Calibration results for different EA configurations with best found sets of parameters and average gaps to optimal results (their 95 percent confidence intervals given beside them) for calibration networks. Crossover types: 2P - two point crossover, INJ - injection crossover, PR - path relinking crossover. Selection types: TUR - unbiased tournament selection, SUS - fitness proportionate selection with stochastic universal sampling, CRO - deterministic crowding. The result of the best EA configuration in bold.

Crossover	Selection	Calibrated parameters		rd400	pr299	gil262C	All 3 networks
		p_k p_m p_z	z_k z_m z_z	gap (%)	gap (%)	gap (%)	avg. gap (%)
2-P	TUR	0,6 1,0 0,70	0,4 0,6 0,4	3,47	3,32	1,14	2,64 ±0.50
	SUS	0,6 1,0 0,10	0,8 0,8 0,6	2,37	1,42	0,81	1,53 ±0.38
	CRO	1,0 1,0 0,10	0,6 0,8 1.0	0,63	0,91	0,38	**0.64 ±0.12**
INJ	TUR	0,8 1,0 0,50	0,6 0,6 0,6	4,84	2,40	1,53	2,92 ±0.56
	SUS	0,6 1,0 0,00	0,8 0,8 -	3,20	2,37	0,74	2,10 ±0.54
	CRO	1,0 1,0 0,00	0,4 0,6 -	5,96	2,34	2,54	3,61 ±0.32
PR	TUR	0,6 1,0 0,70	1,0 0,6 0,6	3,86	1,90	0,63	2,13 ±0.60
	SUS	0,4 1,0 0,10	1,0 0,8 0,6	2,09	2,63	0,41	1,71 ±0.34
	CRO	0,8 1,0 0,03	0,4 0,8 0,2	2,36	0,93	0,45	1,25 ±0.30

Table 6. Comparison of average gap (in percent) for rd400 instance (average from 30 runs) depending on z_k and z_m greediness/randomness parameters (for crossover and mutation respectively). EA configuration: 2-point crossover and deterministic crowding. Remaining parameters were the same as in table 5. 95 percent confidence intervals for average gap are given by ± sign. Average gaps after evolution phase but before final local improvement phase are given in parentheses. The best parameter set in bold.

$z_k \setminus z_m$	0.0	0.2	0.4	0.6	0.8	1.0
0.0	11.07 ±0.45 (29.40)	8.06 ±0.46 (18.67)	4.51 ±0.32 (7.81)	2.11 ±0.30 (2.90)	2.28 ±0.23 (2.79)	2.49 ±0.21 (2.93)
0.2	8.44 ±0.69 (13.64)	5.04 ±0.28 (6.58)	2.54 ±0.21 (2.77)	0.79 ±0.17 (0.83)	1.10 ±0.15 (1.13)	1.03 ±0.16 (1.07)
0.4	6.97 ±0.68 (8.97)	4.23 ±0.27 (4.65)	2.34 ±0.16 (2.42)	0.79 ±0.14 (0.83)	0.76 ±0.14 (0.78)	0.89 ±0.15 (0.93)
0.6	6.59 ±0.59 (7.33)	3.79 ±0.21 (4.04)	2.31 ±0.19 (2.34)	1.01 ±0.11 (1.04)	**0.69** ±**0.10** **(0.71)**	0.91 ±0.14 (0.93)
0.8	6.14 ±0.55 (6.42)	3.83 ±0.22 (3.98)	2.38 ±0.14 (2.42)	1.19 ±0.13 (1.20)	0.79 ±0.09 (0.80)	0.84 ±0.09 (0.87)
1.0	6.19 ±0.38 (6.36)	3.89 ±0.17 (3.91)	2.62 ±0.12 (2.66)	1.42 ±0.11 (1.43)	0.96 ±0.12 (0.97)	1.00 ±0.12 (1.02)

calibrating EA with two-point crossover type and deterministic crowding survival selection type. Average gap is only 0.64 per cent and EA gives the best results for all three calibration networks. One can see that algorithm configurations with two-point crossover give the best average results regardless of selection type. Tuned EAs with deterministic crowding selection give the best results in two (out of three) crossover types. Crossover probabilities chosen by tuner were generally high or medium and best configurations had very high crossover probability (0.8-1.0). Crossover greediness/randomness ratio is medium or high and its values for best configurations (0.4-0.6) suggest importance of both elements (greediness and randomness) when creating offspring individuals. Mutation probability is very high (1.0) in all cases and its greedy-random ratio is medium or high (similarly to crossover). These results show importance of local search operations in mutation phase. It can be seen that disturb probability varies strongly depending on selection type - it is high for tournament selection and low for other selection types. It is associated with stronger tournament selection pressure compared to other selection types - increasing disturb probability counters selection effect and assures better diversity (and longer convergence) of the population.

In table 6 there are results (rd400 test instance) for the best EA configuration (2-point crossover and deterministic crowding) depending on z_k and z_m values. Remaining parameters were set by tuning procedure. It can be seen that solution quality rises quickly as greediness of crossover and mutation grows. The difference between purely random parameters configuration and the best configurations is about 10 percent. One can see that gaps for randomness-favouring parameter sets are even bigger without post-evolution local improvement phase. For more greedy parameter sets final improvement phase brings little or no change to solutions quality. The best parameters values are about 0.6-0.8 for z_m and 0.4-0.8 for z_k. When greediness grows to 1.0 (for both parameters) results quality drops gently. One can see that variance of EA results (shown by length of confidence intervals) is dependent on z_k and z_m ratios - more greedy configurations give steadier results. Confidence interval for gap in the best parameter set is rather short but a few other parameter sets are not significantly worse in terms of results. Local search methods are essential in this configuration but random component also plays its role. Large advantage of greediness over randomness arises from selection mechanism - in deterministic crowding selection pressure is lower and randomness-favouring parameter sets cannot converge to good solutions. In this algorithm configuration local search operators are probably more responsible for convergence than selection.

Table 7. Experiment results for different, tuned EA configurations (average gaps and 95 percent confidence intervals for them) for all test instances from both problem classes. Crossover types: 2P - two point crossover, INJ - injection crossover, PR - path relinking crossover. Selection types: TUR - unbiased tournament selection, SUS - fitness proportionate selection with stochastic universal sampling, CRO - deterministic crowding. The results of the best EA configuration in bold.

Crossover	Selection	calibration networks	class I networks	class II networks
		avg. gap (%)	avg. gap (%)	avg. gap (%)
2-P	TUR	2.64	2.12 ±0.10	2.06 ±0.16
	SUS	1.53	1.20 ±0.08	1.21 ±0.10
	CRO	**0.64**	**0.41 ±0.03**	**0.34 ±0.03**
INJ	TUR	2.92	2.66 ±0.13	2.90 ±0.23
	SUS	2.10	2.01 ±0.12	1.53 ±0.18
	CRO	3,61	1.52 ±0.04	2.24 ±0.08
PR	TUR	2.13	2.06 ±0.13	1.71 ±0.11
	SUS	1.71	1.51 ±0.10	1.30 ±0.12
	CRO	1.25	1.55 ±0.09	1.43 ±0.12

6.4 Test instances results

For each problem instance EA was executed 30 times and average profit was computed. In addition 95 percent confidence intervals for average gaps were presented. In table 7 average results of tuned EA configurations for all instances from both classes are displayed and compared to results from calibration phase. One can see that the best algorithm configuration from tuning phase (2-point crossover and deterministic crowding survival selection) achieves on average the best results on all test instances as well - average gap is only 0.41 and 0.34 percent (for class I and class II respectively). In most cases better results for calibration networks implied superiority for all instances but exceptions also appeared (configuration injection crossover + deterministic crowding behaves much better overall than for tuning networks). It can be seen that in most cases overall average gaps for class I and class II are smaller than average gaps for calibrated networks (despite fitting EAs to them). It results from the fact that calibration was performed on larger instances (and probably the hardest to obtain high-quality results) with 262-400 vertices while most test networks had 100-200 nodes.

Table 8. Comparison of results of the best tuned EA configuration (2-point crossover + deterministic crowding) with results of GRASP, GRASPwPR and best known solutions (class I). Execution time is given in seconds. 95 percent confidence intervals for gaps are given beside gaps.

Instance	EA				GRASP		GRASP PR		Best solution
	profit	gap (%)		time	profit	gap (%)	profit	gap (%)	
kroA100	3177.8	0.10	±0.05	1.3	3135	1.45	3181	0	3181
kroB100	3191	0.13	±0.00	1.3	3183	0.38	3191	0.13	3195
kroC100	3025.7	0.60	±0.32	1.5	3044	0	3044	0	3044
kroD100	3222.3	0.11	±0.02	1.7	3152	2.29	3212	0.43	3226
kroE100	3303.9	0.18	±0.17	1.2	3260	1.51	3310	0	3310
rd100	3448.7	0.61	±0.07	1.1	3449	0.61	3453	0.49	3470
eil101	3667.4	0.02	±0.03	1.1	3596	1.96	3645	0.63	3668
lin105	3576.7	0.01	±0.01	1.4	3577	0	3577	0	3577
pr107	2681	0.00	±0.00	0.8	2681	0	2681	0	2681
gr120	4198.5	0.58	±0.13	1.4	4138	2.01	4201	0.52	4223
pr124	3840	0.00	±0.00	1.8	3840	0	3840	0	3840
bier127	5374.7	0.02	±0.03	4.1	5154	4.13	5254	2.27	5376
pr136	4214.2	0.21	±0.04	1.7	4170	1.26	4213	0.24	4223
gr137	4272.1	0.44	±0.03	1.7	4255	0.84	4284	0.16	4291
pr144	3911.5	2.07	±0.57	2	3902	2.3	3994	0	3994
kroA150	4916.8	0.04	±0.03	2.3	4768	3.07	4915	0.08	4919
kroB150	5014.5	0.05	±0.09	2.1	4967	1	5001	0.32	5017
pr152	4192.4	0.09	±0.05	1.9	4094	2.43	4175	0.5	4196
u159	5028.3	0.31	±0.08	2.3	4809	4.66	4987	1.13	5044
rat195	5895.1	0.69	±0.08	2.9	5693	4.09	5693	4.09	5936
d198	6507.9	0.48	±0.07	3.3	6347	2.94	6476	0.96	6539
kroA200	6583.3	0.49	±0.06	3.3	6447	2.55	6551	0.98	6616
kroB200	6581.6	0.23	±0.08	3.5	6357	3.64	6409	2.85	6597
ts225	6732.1	1.17	±0.32	4.2	6701	1.63	6784	0.41	6812
pr226	6685	0.09	±0.10	6.5	6375	4.72	6614	1.15	6691
gil262	9135.8	0.25	±0.05	5.6	8847	3.41	8941	2.38	9159
pr264	6666	0.00	±0.00	3.8	6666	0	6666	0	6666
pr299	9010	1.07	±0.14	6.3	8645	5.07	8689	4.59	9107
lin318	10795.6	1.52	±0.12	8.4	10074	8.1	10339	5.68	10962
rd400	13461.3	0.69	±0.10	16.2	12365	8.78	12365	8.78	13555
Avg.	**5410.4**	**0.41**	**±0.03**	**3.2**	**5256.4**	**2.49**	**5322.8**	**1.29**	**5437.2**

Table 9. Comparison of results of the best tuned EA configuration (2-point crossover + deterministic crowding) with results of GRASP, GRASPwPR and best known solutions (class II). Execution time is given in seconds. 95 percent confidence intervals are given beside gaps.

Instance	EA				GRASP		GRASPwPR		Best solution
	profit	gap (%)		time	profit	gap (%)	profit	gap (%)	
eil101A	571.9	0.02	±0.02	0.7	566	1.05	572	0	572
cmt121A	408.9	0.75	±0.27	0.7	412	0	412	0	412
cmt151A	824	0.00	±0.00	0.8	815	1.09	824	0	824
cmt200A	1204.7	0.02	±0.02	2	1145	4.98	1181	1.99	1205
gil262A	4492.9	0.38	±0.20	2.4	3916	13.17	4050	10.2	4510
eil101B	1047.1	0.18	±0.08	1.2	1024	2.38	1032	1.62	1049
cmt121B	712.7	0.32	±0.17	1.5	699	2.24	707	1.12	715
cmt151B	1535.7	0.08	±0.03	2.1	1482	3.58	1528	0.59	1537
cmt200B	2172.4	1.16	±0.15	5.2	2073	5.69	2105	4.23	2198
gil262B	8420.6	0.42	±0.09	5.7	7946	6.03	8074	4.52	8456
eil101C	1333.6	0.18	±0.06	2.7	1295	3.07	1302	2.54	1336
cmt121C	1129.5	0.40	±0.13	2.1	1120	1.23	1125	0.79	1134
cmt151C	1993.3	0.48	±0.04	6	1965	1.9	1996	0.35	2003
cmt200C	2873.3	0.27	±0.04	10.9	2791	3.12	2824	1.98	2881
gil262C	11153.6	0.37	±0.02	13.3	10938	2.3	11046	1.33	11195
Avg.	**2658.3**	**0.34**	**±0.03**	**3.8**	**2545.8**	**3.46**	**2585.2**	**2.08**	**2668.5**

Table 10. New, best solution found for gil262A.

instance	profit	route
gil262A	4510	1-164-225-83-158-250-63-238-178-70-191-124-119-4-216- 105-142-26-247 -209-181-6-4-217-47-188-162-180-129-49-197-175-144-99-241-118-93-179-211- 42-111-110-21-228-30-98-133-172-182-60-163-76-220-16-103-40-39-224- 44-226-242-12-215-132-58-204-102-115-149-27-10-154-231-171-3-126-65-1

In table 8 there is a comparison (problem class I) between best tuned EA configuration and other algorithms (GRASP, GRASPwPR) [35] as well as best known results obtained by branch-and-cut exact algorithm (results in [35] but algorithm proposed by [14]). EA with average gap of only 0.41 percent is almost 1 percent better than GRASPwPR and over 2 percent better than GRASP. Confidence intervals for average gap show that EA results are significantly better for most test instances. EA advantage is very clear for larger instances (3-8 percent over GRASPwPR). In most cases EA obtains close to optimal results in reasonably short execution time (less than 5 seconds in most cases). In table 9 there is a similar comparison for problem class II. EA again clearly outperforms GRASP and GRASPwPR methods by 3.1 and 1.7 percent respectively (gaps are statistically significant) and its execution time is still reasonably short. Results obtained by tuned EA are clearly better than other methods and on average are very close to the best known solutions obtained by exact algorithm. For one instance (gil262A) new, best solution (profit 4510) was obtained by EA (presented in table 10) - previous best solution (profit 4466) was obtained by branch-and-cut algorithm but for some networks its execution time limit (5 hours) was reached and resulting solutions could be worse than optimal.

7. Conclusions and further research

In the paper parameter tuning of evolutionary algorithm solving the Orienteering Problem was carried out. Nine different algorithm configurations (varying in selection and crossover phases) were tuned and tested. Parameters of EA were tuned with ParamsILS local search algorithm. Results show the importance of both choosing algorithm components and parameter calibration when developing EAs. Parameter tuning done in an automatic way have advantages over *adhoc* calibration. For a given algorithm configuration the described tuner was able to find a very good parameters set in a few hours. Tuned EA achieved high-quality solutions and clearly outperformed GRASP and GRASPwPR methods (reaching results close to optimal for most test instances).

Further research is concentrated on other aspects of OP solving evolutionary algorithms i.e. different types of paths initialization (local search methods), infeasible solutions in the population and combining a few crossover operators in one EA. The author is also working on solutions for the Time-Dependent Orienteering Problem (TDOP) [38] particularly for trip planners in public transport networks. Methods working well for the classic OP (EAs in particular) can probably be adapted successfully to time-dependent version of the problem.

Acknowledgment

The author gratefully acknowledges support from the Polish Ministry of Science and Higher Education at the Bialystok University of Technology (grant W/WI/4/2014).

References

[1] Croes, G. A.: A method for solving traveling salesman problems, Operations Research, vol. 6, 791-812, 1958.

[2] Christofides, N., Mingozzi, A., Toth, P.: The Vehicle Routing Problem. Combinatorial Optimization, 315-338, 1979.

[3] Tsiligirides, T.: Heuristic methods applied to orienteering. Journal of the Operational Research Society, vol. 35 (9), 797-809, 1984.

[4] Golden, B., Levy, L., Vohra, R.: The orienteering problem. Naval Research Logistics, vol. 34, 307-318, 1987.

[5] Baker, J.E.: Reducing Bias and Inefficiency in the Selection Algorithm. Proceedings of the Second International Conference on Genetic Algorithms and their Application, Hillsdale, New Jersey: L. Erlbaum Associates, 14-21, 1987.

[6] Reinelt, G.: TSPLIB - A Travelling Salesman Problem Library. ORSA Journal of Computing, vol. 3, 155-165, 1991.

[7] Ramesh, R., Brown, K.: An efficient four-phase heuristic for the generalized orienteering problem. Computers and Operations Research. vol 18, 151-165, 1991.

[8] Ramesh, R., Yoon, Y., Karwan, M.: An optimal algorithm for the orienteering tour problem. ORSA Journal on Computing, vol. 4, 155-165, 1992.

[9] Mahfoud, S.W.: Crowding and preselection revisited. Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature (PPSN II), Brussels, Belgium, 1992. Elsevier, Amsterdam, The Netherlands, 27-36, 1992.

[10] Taguchi, G., Yokoyama, T.: Taguchi Methods: Design of Experiments, ASI Press, 1993.

[11] Chao, I., Golden, B., Wasil, E.: Theory and methodology - a fast and effective heuristic for the orienteering problem. European Journal of Operational Research, vol. 88, 475-489, 1996.

[12] Gendreau, M., Laporte, G., Semet, F.: A branch-and-cut algorithm for the undirected selective traveling salesman problem, Networks, vol. 32(4), 263-273, 1998.

[13] Gendreau, M., Laporte, G., Semet, F.: A tabu search heuristic for the undirected selective travelling salesman problem. European Journal of Operational Research, vol. 106, 539-545, 1998.

[14] Fischetti, M., Salazar, J., Toth, P.: Solving the orienteering problem through branch-and-cut. INFORMS Journal on Computing, vol. 10, 133-148, 1998.

[15] Feillet, D., Dejax, P., Gendreau, M.: Traveling Salesman Problems With Profits, An Overview. Transportation Science, vol. 38, 188-205, 2001.

[16] Tasgetiren, M.: A genetic algorithm with an adaptive penalty function for the orienteering problem. Journal of Economic and Social Research, vol. 4 (2), 1-26. 2001.

[17] Myers, R., Hancock, E.A.: Empirical modelling of genetic algorithms, Evolutionary Computation, vol. 9, 461-493, 2001.

[18] Coy, S. P., Golden, B. L., Runger, G. C., Wasil, E. A.: Using experimental design to find effective parameter settings for heuristics, Journal of Heuristics, vol. 7, 77-97, 2001.

[19] Bartz-Beielstein, T., Parsopoulos, K., Vrahatis, M.: Analysis of Particle Swarm Optimization Using Computational Statistics, in Chalkis (Ed.), Proceedings of the International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2004), Wiley, pp. 34-37, 2004.

[20] Ramos, I., Goldbarg, R., Goldbarg, E., Neto, A.: Logistic regression for parameter tuning on an evolutionary algorithm, in: Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation, IEEE Press, Edinburgh, UK, vol. 2, 1061-1068, 2005.

[21] Birattari, M.: Tuning Metaheuristics, Springer, 2005.

[22] Sokolov, A., Whitley, D.: Unbiased tournament selection, Proceedings of Genetic and Evolutionary Computation Conference. ACM Press, 1131-1138, 2005.

[23] Adenso-Diaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental designs and local search, Oper. Res. , vol. 54, 99-114, 2006.

[24] Balaprakash, P., Birattari, M., Stutzle, T.: Improvement strategies for the F-Race algorithm: Sampling design and iterative refinementace algorithm: Sampling design and iterative refinement, in: T. Bartz-Beielstein, M. Blesa Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, M. Sampels (Eds.), Hybrid Metaheuristics, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, vol. 4771, 108-122, 2007.

[25] Nannen, V., Eiben, A. E.: Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters, in: M. M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, 1034?1039, 2007.

[26] Jozefowiez, N., Glover, F., Laguna, M.: Multi-objective Meta-heuristics for the Traveling Salesman Problem with Profits. Journal of Mathematical Modelling and Algorithms, vol. 7, 177-195, 2008.

[27] Wang Q.,Sun X., Golden B. L. and Jia J.,: Using artificial neural networks to solve the orienteering problem, emph Annals of Operations Research, vol.61, 111-120, 2008.

[28] Schilde, M., Doerner, K., Hartl, R., Kiechle, G.: Metaheuristics for the biobjective orienteering problem. Swarm Intelligence, vol. 3, 179-201, 2009.

[29] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G. and Oudheusden, D.V.: A guided local search metaheuristic for the team orienteering problem. European Journal of the Operational Research, vol. 196(1), 118-127, 2009.

[30] Hutter, F., Hoos, H.H., Leyton-Brown, K., Stutzle, T.: ParamILS: an automatic algorithm configuration framework, Journal of Artificial Intelligence Research, vol. 36, 267-306, 2009.

[31] Souffriau, W., Vansteenwegen, P., Vanden Berghe, G. and Oudheusden, D.V.: A path relinking approach for the team orienteering problem, Computers and Operations Research, vol. 37, 1853-1859, 2010.

[32] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.: The City Trip Planner: An expert system for tourists, Expert Systems with Applications, vol. 38(6), 6540-6546, 2011.

[33] Ostrowski, K, Koszelew, J.: The comparison of genetic algorithm which solve orienteering problem using complete and incomplete graph, Zeszyty Naukowe, Politechnika Bialostocka. Informatyka, vol. 8 61-77, 2011.

[34] Karbowska-Chilinska, J., Koszelew, J., Ostrowski, K., Zabielski, P.: Genetic algorithm solving orienteering problem in large networks, Frontiers in Artificial Intelligence and Applications, vol. 243, 28-38, 2012.

[35] Campos, V., Marti, R.,Sanchez-Oro, J., Duarte, A.: Grasp with Path Relinking for the Orienteering Problem. Journal of the Operational Research Society, vol. 156, 1-14, 2013.

[36] Koszelew. J., Ostrowski. K.: A Genetic Algorithm with Multiple Mutation which Solves Orienteering Problem in Large Networks. Computational Collective Intelligence. Technologies and Applications, LNCS 8083, 356-366, 2013.

[37] Karbowska-Chilinska, J., Zabielski, P.: Genetic Algorithm with Path Relinking for the Orienteering Problem with Time Windows, Fundamenta Informaticae, vol. 135, 419-431, 2014.

[38] Ostrowski, K.: Comparison of Different Graph Weights Representations Used to Solve the Time-Dependent Orienteering Problem, Trends in Contemporary Computer Science, Podlasie 2014, Bialystok University of Technology Publishing Office, 144-154, 2014.

[39] Zabielski. P., Karbowska-Chilinska, J., Koszelew, J., Ostrowski, K.: A Genetic Algorithm with Grouping Selection and Searching Operators for the Orienteering Problem, Lecture Notes in Artificial Intelligence, LNCS 9012, 31-40, 2015.

Krzysztof Ostrowski

KALIBRACJA PARAMETRÓW ALGORYTMU EWOLUCYJNEGO ROZWIĄZUJĄCEGO ORIENTEERING PROBLEM

Streszczenie Różne klasy algorytmów rozwiązujących problemy optymalizacyjne posiadają zestawy parametrów. Ustawienie odpowiednich wartości parametrów może być równie ważne, co dobór odpowiednich komponentów algorytmu. Kalibracja parametrów sama w sobie może być skomplikowanym problemem optymalizacyjnym i wiele meta-algorytmów zostało zaproponowanych by przeprowadzać ten proces automatycznie. Artykuł prezentuje automatyczną kalibrację parametrów algorytmu ewolucyjnego rozwiązującego Orienteering Problem. W tym celu wybrano metodę ParamsILS. Otrzymane rezultaty ukazują jak ważny jest odpowiedni dobór parametrów: algorytm po kalibracji uzyskał bardzo wysokiej jakości rozwiązania dla znanych sieci testowych.

Słowa kluczowe: kalibracja parametrów, algorytmy ewolucyjne, Orienteering Problem

MONITORING VMWARE-BASED CLOUD COMPUTING INFRASTRUCTURE WITH NAGIOS

Dariusz Sosnowski, Krzysztof Bielawski

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: Cloud computing (CC) is a very popular model of service. It provides clients with dynamic infrastructure or platform to perform various tasks. Monitoring of underlying infrastructure enables providers to assure a certain level of quality of service. This article describes requirements and methodology of deploying monitoring system based on Nagios, for CC infrastructure based on VMware virtualization software. It also presents a case study of such system.

Keywords: monitoring, VMware, Nagios, cloud computing

1. Introduction

1.1 Cloud computing model

Cloud computing (CC) is defined [1] as a service that provides on-demand network access to a pool of configurable computing resources. Depending on level of abstraction, the end user has access to defined instances of software or to whole infrastructure. Basically such functionality is implemented using multiple virtualization solutions, either commercial (e.g. VMware vSphere, Microsoft Hyper-V) or open-source (Linux KVM, Xen).

In available articles (e.g [2], [3]) CC infrastructure is divided into several layers. For the purpose of this paper, simplified model is presented:

– **Hardware and network infrastructure** - lowest layer in the model; physical servers, network infrastructure (routers, firewall, switches), storage arrays are representative elements of this layer; those entities provide resources (computing power - CPU time, RAM memory, disk space, etc.) that will be shared with the clients;

– **Virtualization hypervisor** - virtualization systems and software used for management; enables service providers to distribute desired resources to clients, leveraging virtual machines and virtualized networks;
– **Virtual machines, operating system layer** - virtual machines created by entities in hypervisor layer, which purpose is to distribute computing resources to client; in this layer we also locate operating systems installed on those machines that are managed by provider or client (in Infrastructure as a Service - IaaS model);
– **Application** - software or platform accesible by cloud's end users; this layer is provided by service provider (Platform as a Service - PaaS or Infrastructure as a Service - IaaS model) or it is independent from provider (as in IaaS model, where client manages virtual machines and applications installed by himself);

In this paper focus is mostly put on IaaS model. This model is characterized by its separation of concerns. Clients manage ordered virtual machines and other resources by themselves. It means that service provider works only in **Hardware and network** and **Hypervisor** layer. Additionally provider might be working in **Operating system** layer, because some of the management servers can be virtualized.

1.2 Monitoring cloud computing infrastructure

From the provider perspective it is very important to have a reliable and robust monitoring system. It helps provider to assure a certain level of quality of service. Having such enables infrastructure administrators to quickly react on critical events and diagnose their causes. Information gathered and presented by monitoring system, can be assigned to certain layers of CC model:

– **Hardware and network infrastructure**
  • servers accesibility on the network
  • network traffic
  • temperature of CPU cores on virtualization hosts
  • power usage and load of server's power supplies and *power distribution units* (PDUs)
– **Virtualization hypervisor**
  • virtualization hosts' accessibility
  • virtualization hosts' memory and CPU usage
  • usage of disk storage supplied for storing virtual machines images
– **Virtual machines, operating system layer**
  • virtual machines' accessibility
  • virtual machines' memory and CPU usage

  - disk usage
  - checking if certain processes are working
- **Application** - maintained by the clients

There are automated solutions provided by virtualization software providers. VMware developed software named vCenter Operations Manager (vCOps) [4], which provides overview of hypervisor layer - state of virtual machines, resource usage of virtualization hosts. There are two major drawbacks of this solution. Firstly, it introduces addtional licensing cost for service provider. While it is not a problem for bigger corporations, still eliminating those costs might reduce price of the service for clients and in the case of smaller provider companies reduce operating costs. Secondly, its monitoring scope is limited to hypervisor layer. It does not allow us to inspect operating systems installed on virtual machines, review state of particular applications or monitor hardware specific metrics of underlying bare metal infrastructure. Goal of this article is to present a monitoring system, that eliminates those two disadvantages. It must be capable of integrating into CC infrastructure based on VMware virtualization stack and give access to operating system layer, what vCOps prevents us from. Eliminating licensing costs can be achieved by using open-source components.

In this article a monitoring system based on Nagios [5] is presented. Following sections provide more insight into Nagios and methods used to integrate Nagios with the infrastructure. Also a case study of the monitoring system built on top of Nagios and integrated into CC infrastructure is described.

2. Nagios monitoring software

Nagios is an open-source monitoring software, that is well recognized in professional community. This system can be used to monitor a low-level hardware and network infrastructure, but also high-level entities such as operating systems, web servers and other services.

Internally, an infrastructure is defined as a set of objects. *Host* objects represent any network-accessible hardware. *Host* objects have assigned multiple *Service* objects. Every *Service* object is responsible for one metric, service, application or any other property of the *Host*. State of *Service* is determined by the output of assigned *Command*, which represent a plugin to be run to pull *Host*/*Service* state. To every *Service* and *Host*, a *Contact* objects can be assigned, which hold contact information.

Following sections present features of Nagios, useful for monitoring CC infrastructure.

2.1 Plugin system

Nagios uses plugins to inspect a state of servers and applications. Initial set of plugins contains programs such as:

- **check_disk** - check local host's disk usage
- **check_swap** - check local host's swap usage
- **check_snmp** - perform an SNMP query and compare the result to given numeric thresholds or regular expressions
- **check_http** - perform an HTTP(S) request to given server and compare output and output to given pattern
- **check_by_ssh** - runs a command on remote server using a SSH protocol, and uses that command output

System administrator can easily create his own plugins. It can be written in any programming language and Nagios uses only its standard output and return code. Return code is used to determine host/service status. Standard output is used for textual representation of this state. This paper presents technologies and methods that can be used to create plugins for Nagios, to integrate monitoring system into VMware virtualization infrastructure. Because of Nagios plugin system's flexibility it can also be included in many others CC infrastructures.

2.2 Active and passive checks

By default Nagios supports active state checking. In this pattern, a monitoring system establishes a connection with certain host and pulls necessary information to determine service state. Another pattern is passive checking, where parts of the system send their state to Nagios. Nagios has implemented this paradigm and it can be enabled for subset or whole infrastructure. Using this paradigm in monitoring system can lead to increased performance of the system, by reducing load on monitoring server.

2.3 Notifications

In case of critical states of host or service, system administrators must be informed. Nagios has implemented simple system of notifications. When host or service changes its state, a predefined command is executed. That command can send an email, SMS, etc. - it depends on configuration. It has access to host/service name, state, plugin output and contact information.

2.4 Event handlers

For every host or service state change, system administrator can define an optional command to be run. It allows monitoring system to be an active member of infrastrcture, by reacting to certain events. This way Nagios can restart failed services, log events to database, enter ticket to helpdesk center, etc.. This feature is optional, but using it can provide much more sophisticated system.

3. Nagios integration with cloud computing infrastructure based on VMware stack

3.1 Hardware and network layer

Hardware and network layer of CC infrastructure includes:

– physical servers that provide computing resources, i.e. CPU and RAM
– storage arrays (specialized servers capable of managing multiple hard drives, usually with RAID support)
– network switches, routers and firewalls
– power distribution units (PDU)

Aside from monitoring tools provided by manufacturers of these devices, most of them implement *Simple Network Management Protocol* (SNMP) [7]. It is a protocol defined by *Internet Engineering Task Force* (IETF). SNMP exposes management data in form of variables with explicitly defined *object identifiers* (OID). OIDs are sequences of numbers, which represent namespaces of variables and specific variables. For example OID 1.3.6.1.2.1.3 represents variable which holds system uptime.

SNMP allows to use *Management Information Bases* (MIBs), which provide textual representations of OIDs, e.g. OID for sysUpTime variable is defined in RFC1213-MIB MIB and can be referred as RFC1213-MIB::sysUpTime. Every manufacturer can support the set of standard modified MIBs (IF-MIB for network interfaces, HOST-RESOURCES-MIB for physical server resources) and provide MIBs specialized for their hardware (Dell provides IDRAC-MIB-SMIv2 to access management information for its blade servers).

SNMP by default uses UDP protocol on port 161. Port 162 is used by monitoring server, to receive *traps*. Traps are hardware-generated (or more generally - client-generated) SNMP messages, that indicate critical events happening in the system.

There are standard utilities for Linux systems to manage servers that accept SNMP requests. On Debian-based distributions *snmpget* and *snmpwalk* tools can be used for sending GET requests. OIDs or MIB variables can be used for SNMP objects identification. Nagios has *check_snmp* plugin (written in C) packaged with standard plugins. It can be defined to pull certain OIDs from given host.

In monitoring hardware and network layer in cloud computing infrastructure IF-MIB information base can be used. It is defined to access hardware interfaces data such as interface description, bytes received, bytes sent, MTU, etc.. All those values are located in IF-MIB::ifTable table. Every row (IF-MIB::ifEntry objects) of this table represents one interfaces of the device and contains multiple variables including:

- ifDescr - interface name (possibly defined by operating system)
- ifInOctets - counts bytes received
- ifOutOctets - counts bytes sent

Those variables can be used to monitor traffic on interfaces of physical servers and switches. Usage of this data, along with hardware-specfic MIBs is presented in the case study.

3.2 Hypervisor layer

Hypervisor layer, as noted in introduction, consists of virtualization systems that enable provider to distribute resources to clients, using virtual machines. Main part of VMware virtualization stack is vSphere ESXi [8] operating system, that acts as hypervisor - software used for creation and management of virtual machines. Hypervisor is installed on every physical server (referred later as *host*). In VMware systems an underlying disk storage is abstracted for hypervisors as *Datastores*. Datastore provides unified interface for vSphere ESXi to store virtual machines images, and can wrap multiple storage backends (e.g. NFS, iSCSI, FibreChannel). Every host, datastore, virtualized network can be controlled by vCenter server that should be configured in infrastructure.

vCenter server exposes HTTP API known as *vSphere API* [9] that can be used to pull state of hypervisor layer. It is designed as set of managed objects and each represents a single element of virtualization infrastructure. Those objects provide information about structure and functioning of infrastructure. VMware provides bindings for Ruby language - RbVmomi [11]. This paper presents methods how to get information about hypervisor hosts and datastores state and use it to create Nagios plugins.

RbVmomi is constructed as a direct mapping of vSphere API objects structure. Code snippet below presents how to pull hypervisor host (describes by IP address) memory usage:

```
 1 conn = RbVmomi::VIM.connect(host: ip,
 2                             user: username,
 3                             password: password,
 4                             insecure: true)
 5 datacenter = conn.serviceInstance
 6                 .find_datacenter('Datacenter')
 7 host = datacenter.hostFolder
 8                 .findByIp(host, RbVmomi::VIM::HostSystem)
 9 perfManager = conn.servinceInstance.content.perfManager
10 memory_max_mb = host.hardware.memorySize / (1024 * 1024)
11 memory_used_mb = perfManager.retrieve_stats([host],
12                                             ['mem.consumed'])
13 usage = (memory_used_mb / memory_max_mb) * 100.0
```

In the **1st** line a connection to vCenter server is established by using *connect* method of *VIM* class. Following - a *Datacenter* object is pulled in the **5th** line. *Datacenter* represents a group of hosts, datastores, networks, etc. which usually are located in one physical place. This object encapsulates objects of other entities. In the **7th** line *HostSystem* object is pulled to variable *host*, which represents a single vSphere ESXi host in infrastructure, identified by its IP address in datacenter. *Host* object is used to gather maximum memory of this host (in **10th** line, value is hold in bytes). *PerformanceManager* object is used to gather various metrics of the system. It used to retrieve *host's* consumed memory (metric names as *'mem.consumed'*) in **11th** line. Finally, the percentage usage of host's memory is calculated (line **13**).

Host's CPU usage can be accessed nearly the same way, using alternative metric in query to *PerformanceManager* object - *'cpu.usagemhz'*. Full list of metrics is accesible in [10].

Datastore usage can be accessed similarly. The following code snippet presents the usage of RbVmomi to access given datastore usage data (datastores are recognized by their name).

```
conn = RbVmomi::VIM.connect(host: ip,
                            user: username,
                            password: password,
                            insecure: true)
datacenter = conn.serviceInstance
                .find_datacenter('Datacenter')
datastore = datacenter.find_datastore(datastore_name)
summary = datastore.summary
capacity, free = summary.capacity, summary.free
used_bytes = capacity - free
usage = (used_bytes / capacity) * 100.0
```

Beginning of the code is a standard procedure for establishing the connection with vCenter server. Then, *Datastore* object is pulled from *Datacenter* (line **7**). *DatastoreSummary* object, which is *Datastore*'s property, encapsulated metrics of the queried datastore. Overall capacity and free space is pulled from this object (line **9**) and percent of usage is calculated.

This methods can be used to create Nagios plugins. Additionally some user-defined thresholds can be applied (for warning and critical states) to further improve a quality of information presented to administrators.

3.3 Operating system layer

Operating system layer consists of virtual machines and specific operating systems installed on them. In this paper we want to monitor some set of machines (with Linux-based or Windows operating system) that are managed by service provider. Those machines are used to manage service resources, infrastructure and serve as backend for website or used by clients to loan resources.

Linux-based systems Nagios provides preferred solution for monitoring Linux-based systems. Nagios Remote Plugin Executor (NRPE) [6] is a software consisting of two parts - plugin for Nagios server, and daemon for monitored server. Daemon must be placed on monitored server. NRPE daemon accepts connection over SSL or SSH and executes commands issued by Nagios server. It is configured to accept only specified set of commands. Nagios server issues execution of plugins, by using *check_nrpe* plugin.

Windows systems Every newer Windows system have builtin system for monitoring. Windows Management Instrumentation (WMI) [12] is the infrastructure for accessing management data of running Windows machine. System administrators are provided with SQL-like language for accessing operating system metrics, such as:

- CPU and memory usage
- running processes list
- disk storage

Those informations can be accessed both locally and remotely. *wmic* [13] is a Linux implementation of WMI client and can be used by Nagios plugins to monitor Windows machines.

3.4 Gathering performance metrics

Nagios by itself does not store metrics returned by plugins. It can be configured to execute certain command to save plugin outputs or sent it to other systems. Although, preferable solution is to use system designed for gathering, storing and presenting metrics. Cacti [14] is an example of such system. It allows us to store metrics in Round Robin Databases (RRD) [15] which are design for storing time-series data. By default Cacti uses SNMP protocol to gather information, but it can also use user created plugins. Plugins developed for Nagios can be used for this task, with slight modification of output format. For more information, refer to documentation found at [14].

4. Case study

In this section, the case study of developed monitoring system is presented. Description is provided, by dividing aspects of the system like it was described in introduction.

4.1 Hardware and network layer

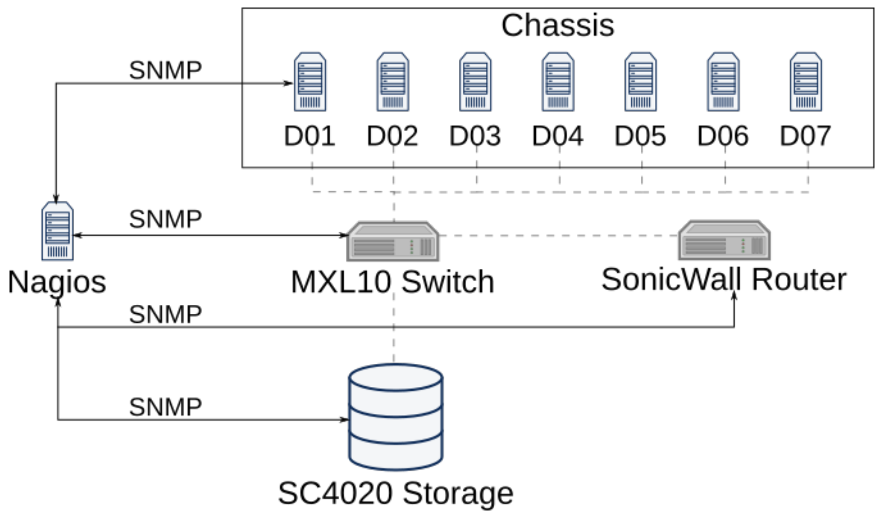


Fig. 1. Hardware layer in the case study

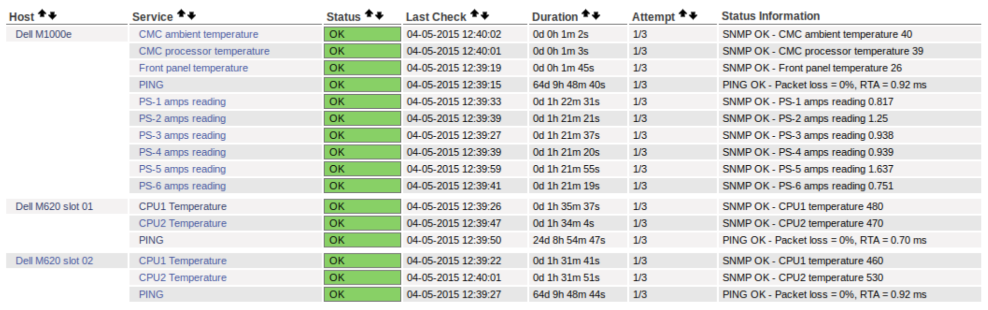
Hardware layer is presented in Figure 1. It consists of several Dell M620 servers, connected to Dell M1000e chassis which provide power and networking for servers. Servers and chassis are connected to Force MXL10 switch. Dell SC4020 storage array is also connected to the switch. SonicWall router is responsible for routing network traffic and acts as firewall.

Interest is put on monitoring accessibility of all those elements in network and network traffic on MXL10 switch and SonicWall router. Temperature of CPU cores of blade servers, temperature inside chassis and its power usage are also needed to monitor.

Accessibility is checked by using ICMP ping standard, realised in *check_ping* Nagios plugin. As noted in section 3.1 network traffic can be monitored by using variables degined in IF-MIB information base for SNMP protocol.

Dell provides system administrators with *iDRAC-MIB-SMIv2* information base for its blade servers, including M620 servers. This MIB holds table named *temperatureProbeTable*, which provides readings from temperature probes located near CPU cores. Every row of table consists of fields like, *temperatureProbeLocationName* - probe location and *temperatureProbeReading* - probe reading.

For Dell M1000e *DELL-RAC-MIB* information base can be used. This base holds table named *drsCMCPSUTable* which contains data of chassis power supplies. Every row of this table contains fields like *drsCMCPSULocation* - location of the probe, *drsCMCPSUAmpsReading* - amperage of power supply unit and *drsCMCPSUVoltsReading* - voltage on power supply. Fields *drsCMCAmbientTemperature*, *drsCMCProcessorTemperature* and *drsChassisFrontPanelAmbientTemperature* defined in MIB can be used to pull respectively CMC ambient temperature (CMC stands for Chassis Management Controller), internal CMC processor and temperature of chassis front panel.



Host	Service	Status	Last Check	Duration	Attempt	Status Information
Dell M1000e	CMC ambient temperature	OK	04-05-2015 12:40:02	0d 0h 1m 2s	1/3	SNMP OK - CMC ambient temperature 40
	CMC processor temperature	OK	04-05-2015 12:40:01	0d 0h 1m 3s	1/3	SNMP OK - CMC processor temperature 39
	Front panel temperature	OK	04-05-2015 12:39:19	0d 0h 1m 45s	1/3	SNMP OK - Front panel temperature 26
	PING	OK	04-05-2015 12:39:15	64d 9h 48m 40s	1/3	PING OK - Packet loss = 0%, RTA = 0.92 ms
	PS-1 amps reading	OK	04-05-2015 12:39:33	0d 1h 22m 31s	1/3	SNMP OK - PS-1 amps reading 0.817
	PS-2 amps reading	OK	04-05-2015 12:39:39	0d 1h 21m 21s	1/3	SNMP OK - PS-2 amps reading 1.25
	PS-3 amps reading	OK	04-05-2015 12:39:27	0d 1h 21m 37s	1/3	SNMP OK - PS-3 amps reading 0.938
	PS-4 amps reading	OK	04-05-2015 12:39:39	0d 1h 21m 20s	1/3	SNMP OK - PS-4 amps reading 0.939
	PS-5 amps reading	OK	04-05-2015 12:39:59	0d 1h 21m 55s	1/3	SNMP OK - PS-5 amps reading 1.637
	PS-6 amps reading	OK	04-05-2015 12:39:41	0d 1h 21m 19s	1/3	SNMP OK - PS-6 amps reading 0.751
Dell M620 slot 01	CPU1 Temperature	OK	04-05-2015 12:39:26	0d 1h 35m 37s	1/3	SNMP OK - CPU1 temperature 480
	CPU2 Temperature	OK	04-05-2015 12:39:47	0d 1h 34m 4s	1/3	SNMP OK - CPU2 temperature 470
	PING	OK	04-05-2015 12:39:50	24d 8h 54m 47s	1/3	PING OK - Packet loss = 0%, RTA = 0.70 ms
Dell M620 slot 02	CPU1 Temperature	OK	04-05-2015 12:39:22	0d 1h 31m 41s	1/3	SNMP OK - CPU1 temperature 460
	CPU2 Temperature	OK	04-05-2015 12:40:01	0d 1h 31m 51s	1/3	SNMP OK - CPU2 temperature 530
	PING	OK	04-05-2015 12:39:27	64d 9h 48m 44s	1/3	PING OK - Packet loss = 0%, RTA = 0.92 ms

Fig. 2. Excerpt from Nagios view of hardware layer

check_snmp plugin can be utilized to monitor network traffic and temperatures in physical hardware of the hardware layer. Accessing hardware and network layer data through SNMP layer, provides provider with more insight into infrastructure state than using only vCOps. Because of its limitation to inspect only hypervisor layer, even simple SNMP monitoring used in this Nagios-based systems extends monitoring scope, eliminating one of the disadvantages of the system.

4.2 Hypervisor layer

Hypervisor layer is presented in Figure 3. It consists of several hosts with vSphere ESXi hypervisor installed, vCenter server used for management of those hypervisors and set of VMware datastores named *MGMT-xx* and *PROD-xx*, where *xx* is an zero-padded number. Only way to monitor those entities from Nagios is to utilize vSphere API exposed by vCenter server. To integrate those two systems, plugins were developed, which used vSphere API Ruby implementation RbVmomi, highlighted in section 3.2. Utilising the methodology presented in this section, we developed plugins to check memory and CPU usage by vSphere ESXi hosts and usage of VMware datastores. Accessibility of vSphere ESXi hosts in the network is checked with *check_ping* plugin.

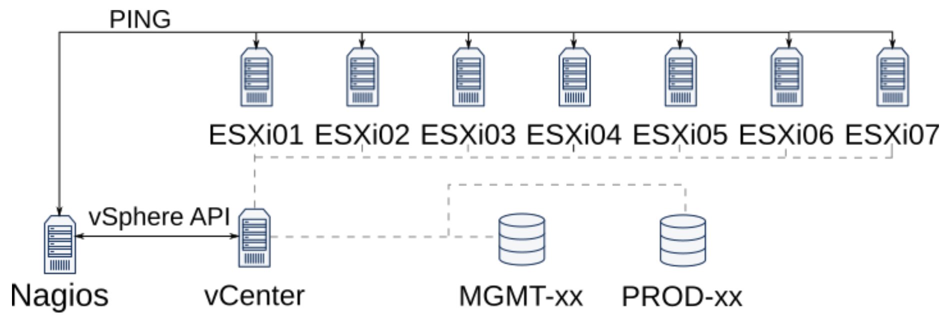
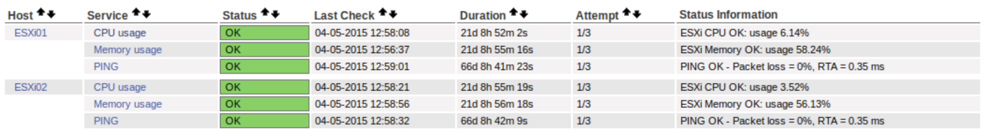


Fig. 3. Hypervisor layer in the case study

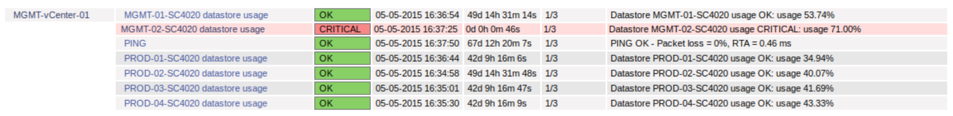
Despite certain advantages of this solution (simplicity of flow of data and certainty about data correctness), having such monitoring introduces a single point of failure. If vCenter server stops working (e.g. internal software error), whole hypervisor layer becomes invisible to hypervisor layer. Though this unwanted situation is by all means critical, it would cause a wave of events concerning unavailability of



Host	Service	Status	Last Check	Duration	Attempt	Status Information
ESXi01	CPU usage	OK	04-05-2015 12:58:08	21d 8h 52m 2s	1/3	ESXi CPU OK: usage 6.14%
	Memory usage	OK	04-05-2015 12:56:37	21d 8h 55m 16s	1/3	ESXi Memory OK: usage 58.24%
	PING	OK	04-05-2015 12:59:01	66d 8h 41m 23s	1/3	PING OK - Packet loss = 0%, RTA = 0.35 ms
ESXi02	CPU usage	OK	04-05-2015 12:58:21	21d 8h 55m 19s	1/3	ESXi CPU OK: usage 3.52%
	Memory usage	OK	04-05-2015 12:58:56	21d 8h 56m 18s	1/3	ESXi Memory OK: usage 56.13%
	PING	OK	04-05-2015 12:58:32	66d 8h 42m 9s	1/3	PING OK - Packet loss = 0%, RTA = 0.35 ms

Fig. 4. Excerpt from vSphere ESXi hosts state from Nagios

vSphere ESXi hosts, which is not the case. This certain payoff must be accepted because of the nature of VMware virtualization stack. It is closed source software and one cannot interfere into internal functioning of hypervisors. Also it is discouraged by professional community to place third-party software inside working hypervisors.



MGMT-vCenter-01	MGMT-01-SC4020 datastore usage	OK	05-05-2015 16:36:54	49d 14h 31m 14s	1/3	Datastore MGMT-01-SC4020 usage OK: usage 53.74%
	MGMT-02-SC4020 datastore usage	CRITICAL	05-05-2015 16:37:25	0d 0h 0m 46s	1/3	Datastore MGMT-02-SC4020 usage CRITICAL: usage 71.00%
	PING	OK	05-05-2015 16:37:50	67d 12h 20m 7s	1/3	PING OK - Packet loss = 0%, RTA = 0.46 ms
	PROD-01-SC4020 datastore usage	OK	05-05-2015 16:36:44	42d 9h 16m 6s	1/3	Datastore PROD-01-SC4020 usage OK: usage 34.94%
	PROD-02-SC4020 datastore usage	OK	05-05-2015 16:34:58	49d 14h 31m 48s	1/3	Datastore PROD-02-SC4020 usage OK: usage 40.07%
	PROD-03-SC4020 datastore usage	OK	05-05-2015 16:35:01	42d 9h 16m 47s	1/3	Datastore PROD-03-SC4020 usage OK: usage 41.69%
	PROD-04-SC4020 datastore usage	OK	05-05-2015 16:35:30	42d 9h 16m 9s	1/3	Datastore PROD-04-SC4020 usage OK: usage 43.33%

Fig. 5. Excerpt from Datastores state from Nagios. Here error message about overusage of storage.

4.3 Operating system layer

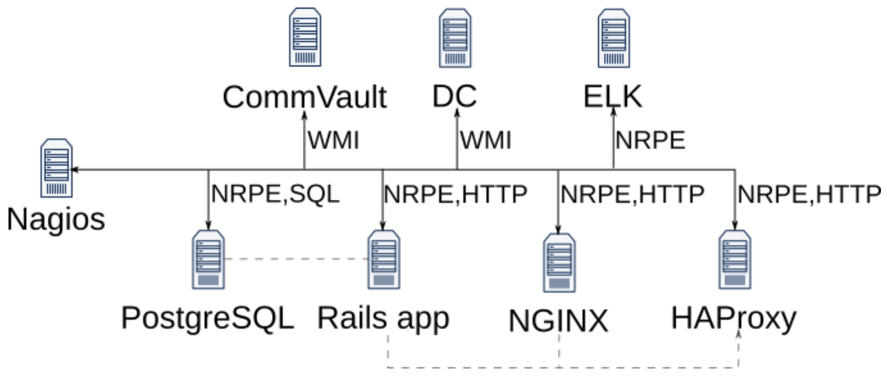
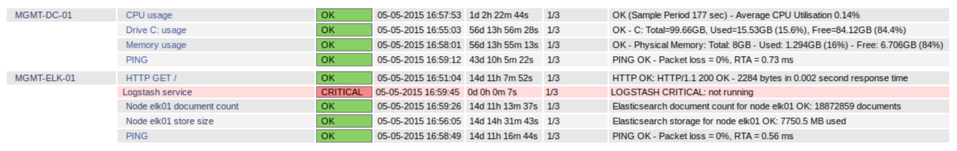


Fig. 6. Operating system layer in the case study

Operating system layer is presented in Figure 6. It consists of several hosts, including 2 virtual machines with Windows operating system *CommVault* - with backup software and *DC* - domain controller for Active Directory. There are also several Debian hosts, i.e. server with PostgreSQL database, server with running Ruby on Rails application, server that servers static content through NGINX, server with HAProxy software (reverse proxy) and ELK server with Elasticsearch database for log gathering and Logstash for log receiving. Windows hosts are monitored using WMI infrastructure described in section 3.3 and utilising *Check WMI Plus* plugin [16], that uses *wmic* Linux client.



MGMT-DC-01	CPU usage	OK	05-05-2015 16:57:53	1d 2h 22m 44s	1/3	OK (Sample Period 177 sec) - Average CPU Utilisation 0.14%
	Drive C: usage	OK	05-05-2015 16:55:03	56d 13h 56m 28s	1/3	OK - C: Total=99.66GB, Used=15.53GB (15.6%), Free=84.12GB (84.4%)
	Memory usage	OK	05-05-2015 16:58:01	56d 13h 55m 13s	1/3	OK - Physical Memory: Total: 8GB - Used: 1.294GB (16%) - Free: 6.706GB (84%)
	PING	OK	05-05-2015 16:59:12	43d 10h 5m 22s	1/3	PING OK - Packet loss = 0%, RTA = 0.73 ms
MGMT-ELK-01	HTTP GET /	OK	05-05-2015 16:51:04	14d 11h 7m 52s	1/3	HTTP OK: HTTP/1.1 200 OK - 2284 bytes in 0.002 second response time
	Logstash service	CRITICAL	05-05-2015 16:59:45	0d 0h 0m 7s	1/3	LOGSTASH CRITICAL: not running
	Node elk01 document count	OK	05-05-2015 16:59:26	14d 11h 13m 37s	1/3	Elasticsearch document count for node elk01 OK: 18872859 documents
	Node elk01 store size	OK	05-05-2015 16:56:05	14d 14h 31m 43s	1/3	Elasticsearch storage for node elk01 OK: 7750.5 MB used
	PING	OK	05-05-2015 16:58:49	14d 11h 16m 44s	1/3	PING OK - Packet loss = 0%, RTA = 0.56 ms

Fig. 7. Excerpt from Operating System layer monitoring. Error message of Logstash service stopped working is shown.

Linux hosts are monitored for CPU load, memory usage and disk storage usage using Nagios basic plugins in conjunction with NRPE software. Additonally state of web services (Ruby on Rails application, NGINX web server and HAProxy reverse proxy) are checked issuing HTTP requests with *check_http* plugin. PostgreSQL is monitored (only connection accessibility) using *check_postgres* plugin [17], which establishes connection to databasse and sending simple query (version check). Elasticsearch database can be monitored by its HTTP API documented here [18]. Plugins in Ruby language were developed to monitor this database. State of Logstash service is checked using NRPE and script placed locally on ELK host.

5. Conclusion

This article presents methodologies and technologies that might be used to integrate Nagios monitoring software with VMware virtualization stack. Monitoring solutions provided by VMware have certain disadvatanges - cost and limitation of scope of monitoring. Nagios eliminates those drawbacks, by allowing system administrator to integrate it into every layer of infrastructure (not only those based on VMware stack). Basic version of Nagios (named Nagios Core) can be used free of charge, thus eliminating cost disadvantage of vCOps. Monitoring system presented in this paper gives an overview of state of whole infrastructure and notifies system administrators about critical events. Methods presented, are de facto standards (SNMP protocol, WMI, vSphere API, etc.) used to manage infrastructures and that gives provider a certain level of reliability and stability.

Further work can be done by improving monitoring capabilities of the system. Methodologies to integrate Nagios with other virtualization stacks and with specific server software (e.g. databases, HTTP server) can be researched. Also data produced by monitoring system (hardware and software metrics, performance data etc.) can be used by other researchers as the test data for machine learning experiments. In this kind of data certain patterns can be found, for example regular spikes in CPU load present regular backups of the infrastructure data. Another type of pattern can be missing metrics, which can represent some errors in working system. Patterns found in this data are not trivial, because of the fact that they can present different types of events in the infrastructure - deterministic (e.g. planned backup) or nondeterministic events (e.g. error in databases). Further research possibly can provide ways to automate diagnosis of those problems and determining their type.

References

[1] P. Mell, T. Grance: *The NIST Definition of Cloud Computing*, NIST Special Publiction 800-145

[2] J. Spring: *Monitoring Cloud Computing by Layer, Part 1*, IEEE Security & Privacy March/April 2011

[3] J. Spring: *Monitoring Cloud Computing by Layer, Part 2*, IEEE Security & Privacy May/June 2011

[4] vCenter Operations Manager: [https://www.vmware.com/support/pubs/vcops-pubs.html]

[5] Nagios: [http://www.nagios.org/]

[6] Nagios Remote Plugin Executor: [http://exchange.nagios.org/directory/Addons/Monitoring-Agents/NRPE--2D-Nagios-Remote-Plugin-Executor/details]

[7] RFC 1448: *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)* [https://tools.ietf.org/html/rfc1905]

[8] vSphere ESXi: [https://www.vmware.com/products/vsphere/features/esxi-hypervisor]

[9] vSphere 5.5 API: [https://pubs.vmware.com/vsphere-55/index.jsp]

[10] vSphere 5.5 API - Performance metrics: [https://www.vmware.com/support/developer/converter-sdk/conv55_apireference/vim.PerformanceManager.html]

[11] RbVmomi: [https://github.com/vmware/rbvmomi]

[12] Window Management Instrumentation: `[https://msdn.microsoft.com/library/aa394582.aspx]`

[13] wmic: `[http://www.aldeid.com/wiki/Wmic-linux]`

[14] Cacti: `[http://www.cacti.net/]`

[15] RRDtool: `[http://oss.oetiker.ch/rrdtool/doc/index.en.html]`

[16] Check WMI Plus: `[http://exchange.nagios.org/directory/Plugins/Operating-Systems/Windows/WMI/Check-WMI-Plus/details]`

[17] check_postgres: `[https://github.com/bucardo/check_postgres]`

[18] Elasticsearch API: `[http://www.elastic.co/guide/]`

MONITOROWANIE INFRASTRUKTURY CLOUD COMPUTING OPARTEJ O NADZORCĘ VMWARE PRZY UŻYCIU NAGIOSA

Streszczenie Cloud computing (CC) jest w dzisiejszych czasach bardzo popularnym modelem usług. W tym modelu, klient posiada dostęp do dynamicznej platformy lub infrastruktury do wykonywania różnych zadań. Monitorowanie infrastruktury będącej podstawą takiej usługi, pozwala usługodawcom na zapewnienie wysokiej jakości usługi. Ten artykuł opisuje wymagania oraz metody implementacji systemu monitorowania, bazując na oprogramowaniu Nagios, dla infrastruktury CC opartej o oprogramowanie do wirtualizacji firmy VMware. Dodatkowo przedstawia on studium przypadku takiego systemu.

Słowa kluczowe: monitorowanie, VMware, Nagios, cloud computing

ALGORITHM OF ADDING THE *M*-BIT NUMBERS

Katarzyna Woronowicz

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: In the classic algorithm of adding two m-bit numbers with carries we add a single bits of the added numbers on each of the m positions. If we assume for a single iteration of the algorithm to calculate the value of a single bit of the sum, then for each pair of m-bit numbers the algorithm executes m iterations. In this paper we propose a recursive algorithm of adding two numbers for which the number of the executed iterations is variable and ranges from 0 to m.

Keywords: addition, adding two numbers

1. Introduction

Adding numbers is a basic arithmetic operation performed by a computer. The best known algorithm of adding two binary numbers with carries works in the same way as the algorithm of adding two numbers represented in any positional system - it calculates the sum of the numbers in each position, and if the sum exceeds the base of the system, adds one to the next position. The classical algorithm and its time complexity is described, among others, in positions [2] and [1]. The practicalities of the issue of adding numbers is described for instance in [3]. In this paper we look at the binary numbers as sequences of bits, on which we perform logical operations. As a result, we found a new algorithm for calculating the sum of two numbers. Each iteration of the algorithm determines the successive approximations of the sum of two input numbers. Execution time depends on the structure of the binary representation of these numbers - the maximum number of iterations, required to determine the sum of two numbers, is equal to the length of the binary representation of the input numbers.

In this paper we present an algorithm of adding two positive integers which uses three bitwise operations: xor, and, shift to the left by one position. We prove its correctness and also present an alternative version of the algorithm, based on a recursive equation of the second degree. We calculate the average and the pessimistic time complexity of the algorithm. In the last section we summarize the average and pessimistic time complexity of the algorithm for different sizes of input data.

In this paper we use the following notation: additive group modulo 2^m is denoted by $(\mathbb{Z}_{2^m}, \oplus)$ and the inverse of the element $p \in \mathbb{Z}_{2^m}$ is denoted by $\ominus p$. The group of bit sequences X with xor operation is denoted as $(X, \otimes)$. The expressions $p \ll k$, $p \gg k$ mean shifting a sequence of bits p of k positions to the left or to the right respectively. The length of a bit sequence is denoted by m. In the binary representation of the number p bits are numbered from right to the left, i.e. $p = [p^{m-1}, p^{m-2}, \ldots, p^1, p^0]$. The i-th bit of the number p is denoted by p^i. Notation $p^{[j \cdots i]}$, where $j > i$, means a fragment of the binary representation of the number p from the i-th bit to the j-th bit.

2. Algorithm

In this section we introduce the recursive algorithm of adding two m-bit numbers. The basic form of the algorithm is presented in the Theorem 1. The alternative form of the algorithm is described in the Proposition 1.

Lemma 1. *Let $p, q \in \mathbb{Z}_{2^m}$. Then:*

1. $p \otimes q = (p \vee q) \oplus (\ominus(p \wedge q))$,
2. $p \oplus q = (p \vee q) \oplus (p \wedge q)$,
3. $p \oplus q = (p \otimes q) \oplus ((p \wedge q) \ll 1)$.

Proof.

1. Note that:

$$(p \wedge q) \otimes p = (\neg(p \wedge q) \wedge p) \vee (p \wedge q \wedge \neg p) = (\neg(p \wedge q) \wedge p) =$$

$$((\neg p \vee \neg q) \wedge p) = (\neg p \wedge p) \vee (\neg q \wedge p) = p \wedge \neg q,$$

therefore:

$$p \otimes q = (p \vee q) \wedge (\neg(p \wedge q)) = ((p \vee q) \wedge (p \wedge q)) \otimes (p \vee q) = (p \wedge q) \otimes (p \vee q).$$

If the i-th bit of the expression $p \wedge q$ is equal 1, then the i-th bit of the expression $p \vee q$ is also equal 1. Therefore, when we subtract $p \wedge q$ of $p \vee q$ we have no borrows and an operation $(p \vee q) \oplus (\ominus(p \wedge q))$ is equivalent to xor operation $(p \vee q) \otimes (p \wedge q)$.

2. Consider two numbers $p,q \in \mathbb{Z}_{2^m}$. Note that:

$$p \vee (\neg p \wedge q) = (p \vee \neg p) \wedge (p \vee q) = 1 \wedge (p \vee q) = p \vee q$$

and

$$q \wedge (\neg(\neg p \wedge q)) = q \wedge (p \vee \neg q) = (q \wedge p) \vee (q \wedge \neg q) = (p \wedge q) \vee 0 = p \wedge q.$$

Because $p \wedge (\neg p \wedge q) = 0$, i.e. there is no index i such that the i-th bit of the expression p is equal 1 and the i-th bit of the expression $(\neg p \wedge q)$ is equal 1, so when we add p and $\neg p \wedge q$ we have no carries and:

$$p \vee (\neg p \wedge q) = p \oplus (\neg p \wedge q).$$

Similarly if the i-th bit of $\neg p \wedge q$ is equal 1, then the i-th bit of q also is equal 1, so:

$$q \wedge (\neg(\neg p \wedge q)) = q \oplus (\ominus(\neg p \wedge q)).$$

Thus, finally:

$$p \oplus q = p \oplus (\neg p \wedge q) \oplus q \oplus (\ominus(\neg p \wedge q)) = (p \vee q) \oplus (p \wedge q).$$

3. Because $p \otimes q = (p \vee q) \oplus (\ominus(p \wedge q))$, so $p \vee q = (p \otimes q) \oplus (p \wedge q)$. Thus:

$$p \oplus q = (p \vee q) \oplus (p \wedge q) = (p \otimes q) \oplus (p \wedge q) \oplus (p \wedge q) = (p \otimes q) \oplus ((p \wedge q) \ll 1).$$

□

Theorem 1. *Let $p,q \in \mathbb{Z}_{2^m}$. Then $p \oplus q = p_m$, where p_m is the m-th term of a sequence defined as follows:*

$$\begin{cases} p_0 = p \\ q_0 = q \\ p_{n+1} = p_n \otimes q_n \\ q_{n+1} = (p_n \wedge q_n) \ll 1 \end{cases} \tag{1}$$

Proof. In the proof we use induction due to the m.

1. If $m = 1$ then $p \oplus q = p \otimes q = p_1$.

2. Let $n < m$.
   Suppose that: if $p, q \in \mathbb{Z}_{2^n}$, then $p \oplus q = p_n$.
   We show that: if $p, q \in \mathbb{Z}_{2^{n+1}}$, then $p \oplus q = p_{n+1}$.
   For the proof, consider the first iteration of the algorithm. Note that:

$$(p \oplus q)^0 = (p \otimes q)^0.$$

Indeed, the expression $(p_n \wedge q_n) \ll 1$ is always an even number, which means that $((p_n \wedge q_n) \ll 1)^0 = 0$. Therefore:

$$(p \oplus q)^0 = (p \otimes q)^0 \otimes ((p_n \wedge q_n) \ll 1)^0 = (p \otimes q)^0 \otimes 0 = (p \otimes q)^0 = p_1^0.$$

Denote $(p \otimes q)^0$ as r_0. Now we consider the remaining n bits of the numbers p_1 and q_1:

$$p_1^{[n\ldots1]} = (p \otimes q) \gg 1,$$

$$q_1^{[n\ldots1]} = ((p \wedge q) \ll 1) \gg 1 = p \wedge q.$$

Note that $(n+1)$-th term of the sequence $\{p_k\}_{k \in \mathbb{N}}$ for numbers p, q is equal to n-th term of the sequence $\{p_k\}_{k \in \mathbb{N}}$ for numbers $p \otimes q = p_1$ and $(p \wedge q) \ll 1 = q_1$. Furthermore:

$$p \oplus q = p_1 \oplus q_1 = (p \otimes q) \oplus ((p \wedge q) \ll 1).$$

Since, by Lemma (1), $p \oplus q = (p \otimes q) \oplus ((p \wedge q) \ll 1)$ and because the length of sequences $p_1^{[n\ldots1]}$, $q_1^{[n\ldots1]}$ equals n, so from the assumption:

$$p_1^{[n\ldots1]} \oplus q_1^{[n\ldots1]} = (p_1^{[n\ldots1]} \otimes q_1^{[n\ldots1]}) \oplus ((p_1^{[n\ldots1]} \wedge q_1^{[n\ldots1]}) \ll 1) = p_{n+1}^{[n\ldots1]}.$$

If we include bits of p^0 and q^0 we obtain:

$((p_1 \oplus q_1) \ll 1) \oplus r_0 = ((((p \otimes q) \gg 1) \oplus (p \wedge q)) \ll 1) \oplus r_0 =$
$((((((p \vee q) \oplus (\ominus(p \wedge q))) \gg 1) \oplus (p \wedge q))) \ll 1) \oplus r_0 =$
$((((p \vee q) \gg 1) \oplus ((p \wedge q) \gg 1)) \ll 1) \oplus r_0 = (p \vee q) \oplus (p \wedge q) \oplus r_0 = p \oplus q.$ □

Lemma 2. *Let $p, q \in \mathbb{Z}_{2^m}$. Then:*

$$p \wedge (p \oplus q) = p \wedge \neg q.$$

Proof. Note that:

$$p \wedge (p \oplus q) = p \wedge ((\neg p \wedge q) \vee (p \wedge \neg q)) =$$
$$= (p \wedge (\neg p \wedge q)) \vee (p \wedge (p \wedge \neg q)) = 0 \vee (p \wedge \neg q) = (p \wedge \neg q).$$

□

Proposition 1. *System of recursive equations (1) can be converted into the recursive equation of the second degree:*

$$p_{n+2} = p_{n+1} \otimes ((p_n \wedge \neg p_{n+1}) \ll 1). \tag{2}$$

Proof. By definition $p_{n+2} = p_{n+1} \otimes q_{n+1}$ and $q_{n+1} = (p_n \wedge q_n) \ll 1$. Because $p_{n+1} = p_n \otimes q_n$, so $q_n = p_n \otimes p_{n+1}$. Thus:

$$p_{n+2} = p_{n+1} \otimes q_{n+1} = p_{n+1} \otimes ((p_n \wedge q_n) \ll 1) =$$
$$= p_{n+1} \otimes ((p_n \wedge (p_n \otimes p_{n+1})) \ll 1).$$

By Lemma (2) we have $p_n \wedge (p_n \otimes p_{n+1}) = p_n \wedge \neg p_{n+1}$, so we obtain:

$$p_{n+2} = p_{n+1} \otimes ((p_n \wedge \neg p_{n+1}) \ll 1).$$

□

2.1 Stopping conditions

Note that if the term $q_n = 0$, then $q_{n+1} = \ldots = q_m = 0$ and $p_n = p_{n+1} = \ldots = p_m$. Indeed, if $q_n = 0$, then:

$$p_{n+1} = p_n \otimes q_n = p_n \otimes 0 = p_n$$

and:

$$q_{n+1} = (p_n \wedge q_n) \ll 1 = (p_n \wedge 0) \ll 1 = 0.$$

It means that if $q_n = 0$, then for all $i = n+1, \ldots, m$ the term $p_i = p_n$, therefore if $q_n = 0$, then $p \oplus q$ is equal to p_n and we can stop the algorithm - successive iterations will not change the result. Consequently, if $q = 0$, the number of iterations is equal to 0. Hence we can formulate the following lemma:

Lemma 3.

1. The algorithm (1) executes more than n iterations if and only if $q_n \neq 0$.
2. The algorithm (1) executes exactly $n+1$ iterations if and only if $q_n \neq 0$ and $q_{n+1} = 0$.

2.2 Example

Let $p = p_0 = 22_{10} = 10110_2$, $q = q_0 = 27_{10} = 11011_2$, $m = 5$. Then:

$$p_1 = p_0 \otimes q_0 = 10110 \otimes 11011 = 01101$$
$$q_1 = (p_0 \wedge q_0) \ll 1 = (10110 \wedge 11011) \ll 1 = 00100$$

$$p_2 = p_1 \otimes q_1 = 01101 \otimes 00100 = 01001$$
$$q_2 = (p_1 \wedge q_1) \ll 1 = (01101 \wedge 00100) \ll 1 = 01000$$

$$p_3 = p_2 \otimes q_2 = 01001 \otimes 01000 = 00001$$
$$q_3 = (p_2 \wedge q_2) \ll 1 = (01001 \wedge 01000) \ll 1 = 10000$$

$$p_4 = p_3 \otimes q_3 = 00001 \otimes 10000 = 10001$$
$$q_4 = (p_3 \wedge q_3) \ll 1 = (00001 \wedge 10000) \ll 1 = 00000$$

According to the Lemma 3, the next iteration is not necessary: the term $p_5 = p_4 \otimes q_4 = 10001 \otimes 00000 = p_4$ and the term $q_5 = (p_4 \wedge q_4) \ll 1 = (10001 \wedge 00000) \ll 1 = 00000 = q_4$. Note that in one case the order of input numbers p and q is important. If $p \neq 0$ and $q = 0$ then the algorithm executes no iteration, but if $p = 0$ and $q \neq 0$ then is executed one iteration.

3. Time complexity

The single iteration of the basic algorithm (1) executes three operations: xor, and, shift to the left by one position. Let us assume that the elementary operation of the algorithm is to execute one of those three bitwise operations. This is a simplification, because each of this operation in fact works on m-bit sequences. However, note that if k-th bit is the first nonzero bit of the number $p_n \wedge q_n$, then $p_n^{[k,k-1,\ldots,0]} = p_{n+1}^{[k,k-1,\ldots,0]} = \ldots = p_m^{[k,k-1,\ldots,0]}$. Consequently, it is not necessary to execute operations on all m bits. In this paper we have focused mainly on the number of iterations executed by an algorithm, so we assume this simplification. Then the number of elementary operations which executes algorithm while adding of two numbers is equal to the number of iterations multiplied by three. Therefore, in order to examine the time complexity of the algorithm, we should count pessimistic and average number of executed iterations. Note that the minimum number of iterations is equal to 0 and the maximum number of iterations is equal to m.

3.1 Average time complexity

Let $p, q \in \mathbb{Z}_m$. To determine the average time complexity of the algorithm we should calculate for how many pairs (p, q) the algorithm executes a given number of iterations. By Lemma (3), the algorithm executes more than 0 iterations if and only if:

$$q_0 \neq 0. \tag{3}$$

It is easy to count that the number of pairs (p, q) for which condition 3 is fulfilled is equal to $2^m \cdot (2^m - 1)$.

By Lemma (3), a necessary and sufficient condition that the number of executed iterations is greater than 1 is:

$$q_1 = (p_0 \wedge q_0) \ll 1 \neq 0. \tag{4}$$

We count for how many pairs (p, q) this condition is fulfilled. Expression $p \wedge q$ is non-zero if and only if there exists index i such that $p^i = q^i = 1$. Because we consider the expression $(p \wedge q) \ll 1$, so for $i = m - 1$ we have:

$$p \wedge q = 2^{m-1} \oplus (p \wedge q)^{[m-2,\ldots,0]}$$

and:

$$(p \wedge q) \ll 1 = (2^{m-1} \oplus (p \wedge q)^{[m-2,\ldots,0]}) \ll 1 = (p \wedge q)^{[m-2,\ldots,0]} \ll 1,$$

so such choice of an index i does not guarantee that the condition (4) is fulfilled. Thus $i \in \{0, 1, \ldots, m-2\}$.

To compute for how many pairs (p, q) there exists at least one index i such that $p^i = q^i = 1$ we use the inclusion-exclusion principle. First we calculate the number of pairs (p, q) such that an index i is fixed and remaining bits of numbers p, q take all possible values. It means, that for a fixed index i there exists an index $j \neq i$ such that $p^j = q^j = 1$ and some pairs (p, q) are computed more than once - sets of pairs (p, q) such that $p^i = q^i = 1$ and $p^j = q^j = 1$ are not disjoint. Thus in the next step we compute for how many pairs there exists two indices i_1, i_2 such that $p^{i_1} = q^{i_1} = 1$ and $p^{i_2} = q^{i_2} = 1$, then we do the calculations for the $3, 4, \ldots, k$ indices and we apply the inclusion-exclusion principle.

We consider the case when there is at least one fixed index i such that $p^i = q^i = 1$. We can choose the index i in $m - 1$ ways, the remaining $m - 1$ bits of the number p and $m - 1$ bits of the number q we fill in all possible ways, therefore there is $(m-1) \cdot (2^{m-1})^2$ pairs (p, q) satisfying the condition.

Now we count for how many pairs there are at least two fixed indices i_1, i_2 such that $p^{i_1} = q^{i_1} = 1$ and $p^{i_2} = q^{i_2} = 1$. We can choose indices in $\binom{m-1}{2}$ ways, the remaining $m-2$ bits of p and q take all possible values, so there is $\binom{m-1}{2} \cdot (2^{m-2})^2$ possibilities.

Generally for k indices $i_1, \ldots, i_k$, $k \leq m-1$, there is $\binom{m-1}{k} \cdot (2^{m-k})^2$ pairs such that $p^{i_1} = q^{i_1} = 1, \ldots, p^{i_k} = q^{i_k} = 1$. Thus, from inclusion-exclusion principle, a number of pairs for which $(p \wedge q) \ll 1 \neq 0$ is equal to:

$$\sum_{i=1}^{m-1} (-1)^{i+1} \cdot \binom{m-1}{i} \cdot (2^{m-i})^2.$$

Using similar reasoning we count for how many pairs the algorithm executes more than two iterations. By lemma (3), this is true if and only if:

$$q_2 = (p_1 \wedge q_1) \ll 1 = ((p_0 \otimes q_0) \wedge ((p_0 \wedge q_0) \ll 1)) \ll 1 \neq 0. \tag{5}$$

The condition (5) is fulfilled if and only if for a pair (p,q) there exists an index i such that:

$$(p \otimes q)^i = ((p \wedge q) \ll 1)^i = 1 \tag{6}$$

and

$$(p \otimes q)^{i+1} \otimes ((p \wedge q) \ll 1)^{i+1} = 1. \tag{7}$$

We count the number of the pairs (p,q) for which there exists at least one fixed index i such that conditions (6), (7) are fulfilled. Note that after the second iteration of the algorithm we execute two shifts to the left by one position. It means that, as in the previous case, that if $i = m-2$, then condition (5) need not to be fulfilled. It is obvious, that if $i = m-1$, then (7) is not satisfied, because p and q are m-bit numbers and bits p^m, q^m do not exist. Therefore $i \in \{0, 1, \ldots, m-3\}$. Besides, the condition (7) means that if we choose an index i, then we have two possibilities for bits p^{i+1} and q^{i+1}

$$p^{i+1} = 0 \text{ and } q^{i+1} = 1 \tag{8}$$

or:

$$p^{i+1} = 1 \text{ and } q^{i+1} = 0 \tag{9}$$

Thus the number of pairs p, q for which there exists at least one fixed index i satisfying the conditions (6) and (7) is equal to $(m-2) \cdot (2^{m-2})^2 \cdot 2$.

Denote the number of possible choices of k indices $i_1, \ldots, i_k$, $k \leq \lfloor \frac{(m-1)}{2} \rfloor$, as $a^2_{m,k}$. If we choose sets of k indices for the pair $(p^{[m-2,\ldots,0]}, q^{[m-2,\ldots,0]})$, then all of these sets of k indices can be chosen for the pair (p,q). Additionally we can choose sets containing index $m-1$. Since the choice of index i causes that index $i+1$ cannot be selected, we should consider all possible $k-1$-element sets of indices without index $m-2$ and add index $m-1$ to all of them, which means that we choose all sets of indices for the pair $p^{[m-3,\ldots,0]}$, $q^{[m-3,\ldots,0]}$. Therefore a sequence $\{a^2_{m,k}\}$ is defined by the following recursive equation:

$$\begin{cases} a^2_{m,k} = m-2 & ,k=1, m\geq 3, \\ a^2_{m,k} = a^2_{m-1,k} + a^2_{m-2,k-1} & ,k>1, m\geq 3. \end{cases} \tag{10}$$

If we choose k indices $i_1,\ldots,i_k$, then each pair of bits p_{i_j+1}, q_{i_j+1}, $j=1,\ldots,k$ must satisfy condition (7), so each such pair fulfill condition (8) or (9). The remaining $m-2k$ bits of p and q take all possible values, so from inclusion-exclusion principle, the number of pairs for which conditions (6) and (7) are satisfied is equal to:

$$\sum_{i=1}^{\lfloor\frac{(m-1)}{2}\rfloor} (-1)^{i+1}\cdot a^2_{m,i}\cdot 2^{2m-3i}.$$

Now we consider for how many pairs the algorithm executes more than three iterations. This is true if and only if:

$$\begin{gathered} q_3 = (p_2\wedge q_2) \ll 1 = \\ = \Big(((p_0\otimes q_0)\otimes((p_0\wedge q_0)\ll 1))\wedge((p_0\otimes q_0)\wedge((p_0\wedge q_0)\ll 1))\ll 1\Big) \ll 1 \neq 0. \end{gathered} \tag{11}$$

Similarly as in the previous case, the condition (11) is fulfilled if for pair (p,q) exists index i such that:

$$(p\otimes q)^i = ((p\wedge q)\ll 1)^i = 1 \tag{12}$$

and

$$(p\otimes q)^{i+1}\otimes((p\wedge q)\ll 1)^{i+1} = 1 \tag{13}$$

and

$$(p\otimes q)^{i+2}\otimes((p\wedge q)\ll 1)^{i+2} = 1. \tag{14}$$

If there is at least one such fixed index i, then, because we made already three shifts to the left, we can select it from indices $\{0,1,\ldots,m-4\}$, so we can choose i in $m-3$ ways. We can fill pairs of bits (p^{i+1},q^{i+1}) and (p^{i+2},q^{i+2}) in $2\cdot 2$ ways, the remaining $m-3$ bits of p and $m-3$ bits of q we can fill in all possible ways, so there is $(m-3)\cdot(2^{m-3})^2\cdot 2^2$ pairs p, q satisfying the conditions (12), (13) and

(14) for which there exists at least one such index i. Now we define sequence $a^3_{m,k}$ analogously to the sequence (10).

If we choose sets of k indices for pair $p^{[m-2,\ldots,0]}$, $q^{[m-2,\ldots,0]}$, $k \leq \lfloor\frac{(m-1)}{3}\rfloor$, then all of these sets of indices can be chosen for pair p, q. We should join all sets containing index $m-1$ to the selected family of sets. Since the choice of index i causes that indices $i+1$ and $i+2$ cannot be selected, we should consider all $k-1$-elements sets of indices for pair $p^{[m-4,\ldots,0]}$, $q^{[m-4,\ldots,0]}$. We obtain the sequence defined as follows:

$$\begin{cases} a^3_{m,k} = m-3 & , k=1, m \geq 4, \\ a^3_{m,k} = a^3_{m-1,k} + a^3_{m-3,k-1} & , k>1, m \geq 4. \end{cases} \tag{15}$$

The number of pairs for which conditions (12), (13) and (14) are satisfied is equal to:

$$\sum_{i=1}^{\lfloor\frac{(m-1)}{3}\rfloor} (-1)^{i+1} \cdot a^3_{m,i} \cdot 2^{2m-4i}.$$

Now we define the general formula for the number of pairs, for which algorithm (1) executes more than n iterations. The necessary and sufficient condition, by lemma (3) is that $q_n \neq 0$. We count how many pairs satisfy this condition. First, we define a sequence $a^n_{m,k}$. After the n-th iteration algorithm executed n shifts to the left by one position. The choice of index i causes that $p^{i+1} \otimes q^{i+1} = 1, p^{i+2} \otimes q^{i+2} = 1, \ldots, p^{i+(n-1)} \otimes q^{i+(n-1)} = 1$. The remaining bits of p, q we can fill in all possible ways. The recursive formula for the general case is defined as follows:

$$\begin{cases} a^n_{m,k} = m-n & , k=1, m \geq n+1, \\ a^n_{m,k} = a^n_{m-1,k} + a^n_{m-n,k-1} & , k>1, m \geq n+1. \end{cases} \tag{16}$$

Theorem 2. *An explicit formula for the term $a^n_{m,k}$ is given as follows:*

$$\begin{aligned} a^n_{m,k} = \frac{\big(m-((n-1)\cdot k+1)\big)\cdot\big(m-((n-1)\cdot k+2)\big)\cdot\ldots\cdot\big(m-n\cdot k\big)}{k!} = \\ = \binom{m-((n-1)\cdot k+1)}{k} \end{aligned} \tag{17}$$

Proof. We know that:

$$a^n_{m,k} = a^n_{m-1,k} + a^n_{m-n,k-1}.$$

We can use the formula (17) for terms $a^n_{m-1,k}$ and $a^n_{m-n,k-1}$:

$$a^n_{m-1,k} = \frac{\big(m-1-((n-1)\cdot k+1)\big)\cdot\big(m-1-((n-1)\cdot k+2)\big)\cdot\ldots\cdot\big(m-1-n\cdot k\big)}{k!}$$

and

$$a^n_{m-n,k-1} = \frac{\big(m-n-((n-1)\cdot(k-1)+1)\big)\cdot\big(m-n-((n-1)\cdot(k-1)+2)\big)\cdot\ldots\cdot\big(m-n-n\cdot(k-1)\big)}{(k-1)!}.$$

Note that:

$$m-1-((n-1)\cdot k+1) = m-n-((n-1)\cdot(k-1)+1),$$

$$m-1-((n-1)\cdot k+2) = m-n-((n-1)\cdot(k-1)+2),$$

$$\ldots$$

$$m-1-((n-1)\cdot k+k-1) = m-n-(n\cdot(k-1)).$$

Thus:

$$\begin{gathered}
a^n_{m,k} = a^n_{m-1,k} + a^n_{m-n,k-1} = \\
= \frac{\big(m-1-((n-1)\cdot k+1)\big)\cdot\big(m-1-((n-1)\cdot k+2)\big)\cdot\ldots\cdot\big(m-1-n\cdot k\big)}{k!} + \\
+ \frac{\big(m-n-((n-1)\cdot(k-1)+1)\big)\cdot\big(m-n-((n-1)\cdot(k-1)+2)\big)\cdot\ldots\cdot\big(m-n-n\cdot(k-1)\big)}{(k-1)!} = \\
= \frac{\Big(\big(m-1-((n-1)\cdot k+1)\big)\cdot\big(m-1-((n-1)\cdot k+2)\big)\cdot\ldots\cdot\big(m-1-((n-1)\cdot k+k-1)\big)\Big)\cdot\Big(m-1-n\cdot k+k\Big)}{k!} = \\
= \frac{\Big(\big(m-((n-1)\cdot k+2)\big)\cdot\big(m-((n-1)\cdot k+3)\big)\cdot\ldots\cdot\big(m-((n-1)\cdot k+k)\big)\Big)\cdot\Big(m-((n-1)\cdot k+1)\Big)}{k!} = \\
= \frac{\big(m-((n-1)\cdot k+1)\big)\cdot\big(m-((n-1)\cdot k+2)\big)\cdot\ldots\cdot\big(m-n\cdot k\big)}{k!} = \\
= \binom{m-((n-1)\cdot k+1)}{k}.
\end{gathered}$$

□

Therefore, the general formula for computing the number of pairs (p,q) for which the algorithm (1) executes more than n iterations is given as follows:

$$\sum_{i=1}^{\lfloor\frac{(m-1)}{n}\rfloor} (-1)^{i+1}\cdot a^n_{m,i}\cdot 2^{2m-(n+1)\cdot i}. \tag{18}$$

Denote the sum from (18) as $A_{n,m}$. Let $A_{0,m} = 2^m\cdot(2^m-1)$ and, for the formalities, let $A_{-1,m} = 2^{2m}$. Then the average number of the algorithm iterations is described by the formula:

$$I_{\text{avg}}(m) = \frac{1}{2^{2m}}\cdot\sum_{i=0}^{m}(A_{i-1,m}-A_{i,m})\cdot i = \frac{1}{2^{2m}}\cdot\sum_{i=0}^{m-1}A_{i,m} - m\cdot A_{m,m}, \tag{19}$$

and, because $A_{m,m} = 0$, we obtain:

$$I\text{avg}(m) = \frac{1}{2^{2m}} \cdot \sum_{i=0}^{m-1} A_{i,m}. \tag{20}$$

The average time complexity of the algorithm is equal to:

$$T\text{avg}(m) = \frac{3}{2^{2m}} \cdot \sum_{i=0}^{m-1} A_{i,m}. \tag{21}$$

3.2 Pessimistic time complexity

The number of pairs (p,q), for which the algorithm (1) executes more than $m-1$ iterations is equal to $A_{m,m-1}$. In the pessimistic case the algorithm executes m iterations, so there are $A_{m,m-1} = 2^m$ such pairs (p,q). Therefore pessimistic data, for which algorithm executes $3 \cdot m$ elementary operations are $\frac{1}{2^m}$ of possible data.

4. Experiments

Up to now we have focused on the average number of iterations of the algorithm, necessary for calculating the sum of the two numbers. We not yet managed to find a compact form of the function, which estimates the average time complexity of the algorithm. The average number of iterations and average time complexity, presented in the Table (1), are calculated by formulas (20) and (21) respectively.

For a few selected cases we compared the calculated results with the average complexity obtained by inserting a counter into the code of implemented algorithm and counting complexity by executing the algorithm for all possible pairs of numbers. The results are consistent. In the tests for the m-bit sequences we received average numbers of iterations presented in the Table (1). The results obtained show that, for the tested cases, the average number of iterations of the algorithm is close to the square root of m, and for m bigger than 20 does not exceed the square root of m.

In future studies we plan to determine the compact function expressing the average number of algorithm iterations depending on the size of input data. Another issue that we intend to explore is the exact time complexity of the algorithm, understood as the average number of logical operations executed on the individual bits of the input, instead, as in this paper, on the sequences of bits. In the calculations we will take into account the fact that, depending on the structure of the input numbers, in the second and successive iterations logical operations can be executed on sequences of the changeable length. Making such analysis will allow us to compare the time complexity of the proposed algorithm with the complexity of the classical algorithm.

Table 1. Average time complexity.

m	Average number of iterations	Average time complexity
3	1,4375	4,3125
4	1,82812	5,48438
5	2,16797	6,50391
6	2,46582	7,39746
7	2,72632	8,17896
8	2,95636	8,86908
9	3,16039	9,48116
10	3,34288	10,02865
11	3,50719	10,52158
12	3,65619	10,96857
13	3,79216	11,37647
14	3,91700	11,75099
15	4,03225	12,09674
16	4,13919	12,41756
17	4,23887	12,71662
18	4,33219	12,99657
19	4,41987	13,25961
20	4,50255	13,50764
25	4,85694	14,57081
30	5,14105	15,42316
35	5,37821	16,13464
40	5,58179	16,74536
45	5,76012	17,28036
50	5,91879	17,75637
55	6,06171	18,18512
60	6,19172	18,57515
65	6,31096	18,93289
70	6,42110	19,26330
75	6,52342	19,57025
80	6,61895	19,85686
85	6,70856	20,12567
90	6,79292	20,37876
95	6,87262	20,61786
100	6,94815	20,84444
200	7,96248	23,88744
300	8,55218	25,65653
400	8,96957	26,90870
500	9,29291	27,87873

5. Acknowledgment

The author is grateful to Professor Czesław Bagiński and Professor Wiktor Dańko for their help during the preparation of the paper, insightful observations and useful tips.

References

[1] D. E. Knuth. *The art of computer programming, Vol. 2 Seminumerical Algorithms*. Reading, Massachusetts: Addison-Wesley.

[2] A. A. Karatsuba. The complexity of computations. *Proceedings of the Steklov Institute of Mathematics*, 1995.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest Introduction to Algorithms. *MIT Press and McGraw-Hill*, 1990.

ALGORYTM DODAJĄCY DWIE M-BITOWE LICZBY

Streszczenie W klasycznym algorytmie dodawania dwóch m-bitowych liczb z przeniesieniami dodajemy po kolei bity na poszczególnych pozycjach binarnych reprezentacji danych wejściowych. Jeśli przyjmiemy za iterację algorytmu wyznaczenie wartości pojedynczego bitu sumy, to dla każdej pary m-bitowych liczb algorytm wykonuje m iteracji. W niniejszej pracy proponujemy rekurencyjny algorytm dodawania dwóch liczb, który w pojednynczej iteracji wykonuje trzy operacje logiczne, a liczba iteracji wynosi od 0 do m.

Słowa kluczowe: dodawanie, dodanie dwóch liczb

THE LIST OF REVIEWERS (2015)

1. Tomasz Dziubich
2. Piotr Jerzy Durka
3. Agata Filipowska
4. Henryk Josiński
5. Marcin Kołodziej
6. Marta Kraszewska
7. Tomasz Krzeszowski
8. Michał Lech
9. Krzysztof Lichy
10. Mariusz Oszust
11. Marek Parfieniuk
12. Adam Słowik
13. Cezary Sobaniec
14. Zbigniew Suraj
15. Piotr M. Szczypiński
16. Jacek Widuch
17. Beata Zielosko