Spis treści • *Contents*

Program Maple do szyfrowania i deszyfrowania plików, zapisanych na dyskach i na nośnikach wymiennych

Maple implementation of an interactive application for cryptographic protection of files stored on hard disk and portable memory devices

Czesław Kościelny[1]

Treść: Opisano przykład interaktywnej aplikacji typu `workhsheet` uruchamianej w środowisku Maple 2015.1 i realizującej zadania „bezkluczowego" szyfrowania i deszyfrowania plików. Aplikacja jest prosta w obsłudze, ponieważ użytkownik nie wprowadza żadnych danych tylko używa myszy. Program posiada prosty graficzny interfejs użytkownika i przeznaczony jest głównie do kryptograficznej ochrony plików przechowywanych na dyskach i na nośnikach wymiennych. Aplikacja wyjątkowo skutecznie i niezawodnie chroni pliki przed nieupoważnionym dostępem.

Słowa kluczowe: funkcja biblioteczna Maple convert/base, szyfrowanie symetryczne plików.

Abstract: An example of an interactive implementation Maple worksheet application which performs the „keyless" file encryption or decryption by means of the symmetric cipher has been presented. The application has simple graphical user interface and may be used mainly for cryptographic protection of files stored in disks and in portable memory devices. The application very effectively protect files against unauthorized access.

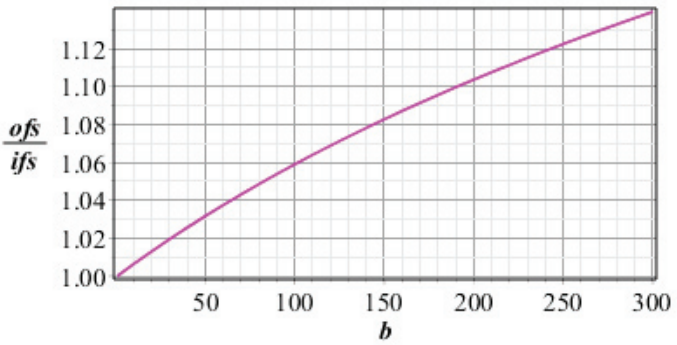
Keywords: Maple convert/base built-in function, symmetric file encryption

1. Wstęp

W pracy [1] przedstawiono program umożliwiający ochronę załączników poczty elektronicznej przed nieupoważnionym dostępem. W niniejszym artykule opisano oryginalną aplikację w postaci programu Maple typu worksheet o nazwie **sdpdnw.mw**, dostępnego w witrynie WWSIS pod adresem
http://www.wydawnictwo.horyzont.eu/publikacje/Informatyka/,
którego zadaniem jest kryptograficzna ochrona plików przechowywanych na dyskach twardych i na nośnikach wymiennych. Aplikację zrealizowano stosując najnowszą wersję tego matematycznego narzędzia, Maple 2015.1.

Rys. 1. Logo Maple 2015.1

Oryginalność prezentowanego rozwiązania polega na zastosowaniu funkcji bibliotecznej programu Maple o nazwie **convert/base** jako przekształcenia kryptograficznego. W procedurach realizujących zadanie szyfrowania i deszyfrowania tę funkcję biblioteczną wywołuje się z parametrem zawierającym zmienną typu **posint** o nazwie **b**. Od wartości tej zmiennej zależy rozmiar zaszyfrowanego pliku oraz długość



Ryc. 2. Zależność stosunku **ofs/ifs** od zmiennej **b** dla **ifs** = 36496 bajtów,
ofs – rozmiar pliku zaszyfrowanego w bajtach , **ifs** – rozmiar pliku niezaszyfrowanego w bajtach.

tajnego klucza. Jeśli **b** = 1, to **ofs/ifs** = 1 a aplikacja realizuje szyfrowanie jedynie nazwy pliku, zapisując pod tą nazwą plik wybrany do zaszyfrowania i usuwając z dysku lub nośnika plik wejściowy. Przy **b** większym lub równym 2 aplikacja szyfruje zarówno nazwę jak i zawartość pliku. W przypadku gdy **b** = 2 a ifs = 36496 stosunek **ofs/ifs** = 1,00071. Na Rys. 2. pokazano zależność tego stosunku od zmiennej **b** w zakresie od 1 do 300 jeśli rozmiar szyfrowanego pliku wynosi 36496 bajtów. Teoretycznie **b**

1. Wrocławska Wyższa Szkoła Informatyki Stosowanej, ul. Wejherowska 28, 54-239 Wrocław

może mieć wartość dowolnie dużej liczby naturalnej akceptowaną przez system Maple. W tabeli 1. zamieszczono wartość **ofs/ifs** dla ośmiu wartości **b** w zakresie od 500 do 200 000.

Tabela 1. Zależność stosunku **ofs/ifs** od zmiennej **b** o wartościach w zakresie od 500 do 200000 przy ifs = 36496

b	$5\cdot10^2$	$1\cdot10^3$	$2\cdot10^3$	$5\cdot10^3$	$1\cdot10^4$	$2\cdot10^4$	$5\cdot10^4$	$1\cdot10^5$	$2\cdot10^5$
$\frac{ofs}{ifs}$	1, 19	1, 27	1, 38	1, 55	1, 66	1, 78	1, 95	2, 08	2, 2

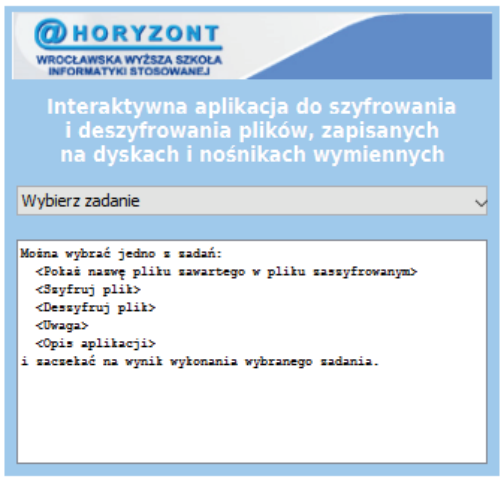
Procedura **fne** szyfruje a procedura **fnd** deszyfruje nazwę pliku, natomiast procedury **af2sf** i **sf2af** wykonują szyfrowanie/deszyfrowanie zawartości pliku. Dlatego też tajny klucz, „zainstalowany" w aplikacji, ma dwie składowe: **kn** - klucz dla procedur **fne** i **fnd**, i **kf** - klucz dla procedur **af2sf** i **sf2af**. Długość tych kluczy w bitach można obliczyć za pomocą instrukcji

```
>vf:=(b-1)*nops(convert(convert(fn,bytes),base,128,6))!;
  kn:=nops(convert(vf,base,2));
  kf:= nops(convert((b-1)*ifs!,base,2));
```

gdzie **fn** oznacza nazwę pliku wybranego do zaszyfrowania a **ifs** rozmiar tego pliku w bajtach. Największe znaczenie dla kryptograficznej ochrony pliku ma klucz **kf**, dlatego nie warto stosować dużych wartości zmiennej **b**, ponieważ wzrost tej zmiennej powoduje zarówno wzrost rozmiaru pliku zaszyfrowanego jak i nieznaczny wzrost bitów klucza, zaś liczba bitów klucza **kf** wzrasta bardzo szybko ze wzrostem **ifs**.

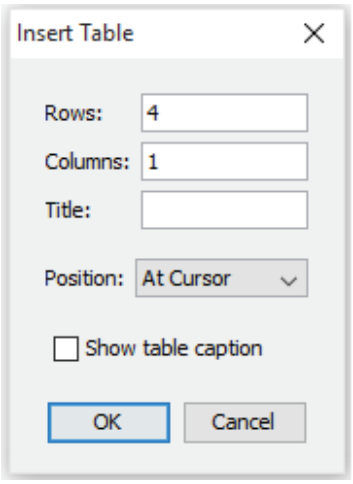
2. Graficzny interfejs użytkownika aplikacji

Po otwarciu programu **sdpdnw.mw** w sesji Maple zostaje wyświetlony graficzny interfejs użytkownika pokazany na Ryc. 3. Taki interfejs można z łatwością skonstruować przy pomocy palety **`Components`**. Należy uruchomić program Maple i najpierw kliknąć na belce narzędzi **File/ New/Document Mode** i do wyświetlonego szablonu pokazanego na Ryc. 4



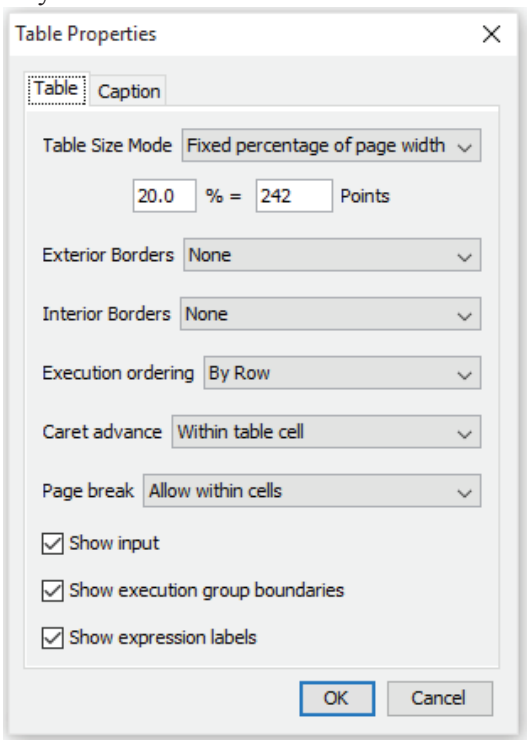
Ryc. 3. Graficzny interfejs użytkownika programu sdpdnw.mw.

należy wpisać ile elementów ma zawierać tworzona tablica.



Ryc. 4. Szablon ustalania liczby wierszy i liczby kolumn tablicy graficznego interfejsu.

Tablicę trzeba zwymiarować, używając szablonu pokazanego na Ryc. 5.



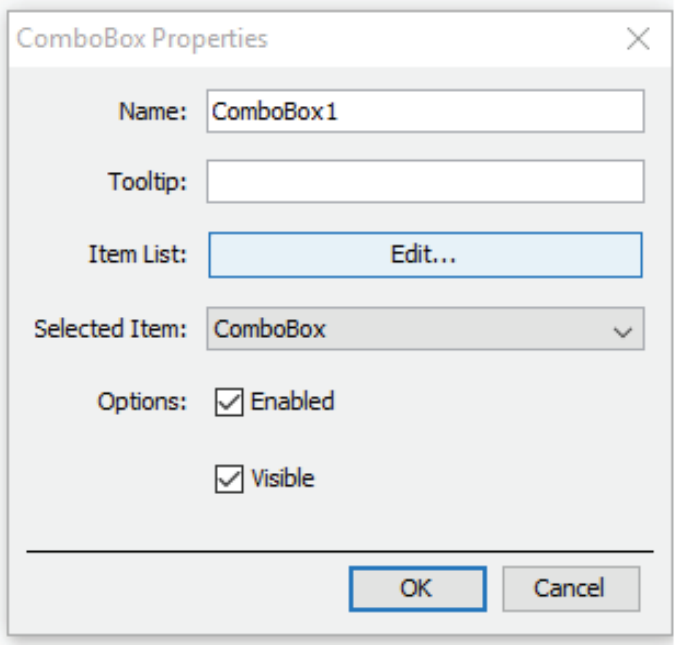
Ryc. 5. Szablon wymiarowania rozmiarów tablicy tworzącej graficzny interfejs.

Do pierwszego wiersza tablicy można wstawić dowolny „obrazek" a w drugim wierszu należy wpisać tytuł aplikacji natomiast do trzeciego wiersza trzeba ściągnąć z palety **`Components`** element o nazwie **`Combo Box`**. Do czwartego wiersza z palety ściąga się element **`Text Area`** wyznaczając liczbę znaków w wierszu i liczbę wierszy:

`Visible Character Width:` 60
`Visible Rows:` 12

Ostatecznie ustala się własności elementu `Combo Box` edytując listę zadań:

`Pokaż nazwę pliku zawartego w pliku zaszyfrowanym`
`Szyfruj plik`
`Deszyfruj plik`
`Uwaga`
`Opis aplikacji`



Ryc. 6. Szablon edycji listy zadań elementu `Combo Box`.

przy pomocy szablonu pokazanego na Ryc. 6.

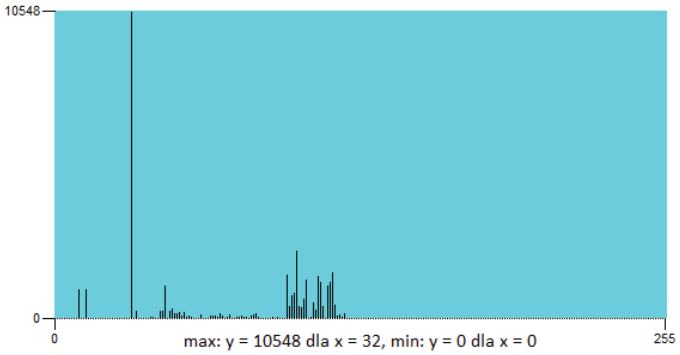
3. Kod źródłowy aplikacji

Kod źródłowy aplikacji umieszczony jest w obszarze kodu startowego i w obszarze wyboru zadań elementu **`Combo Box`**. W obszarze kodu startowego są instrukcje przypisania wartości zmiennym **sfs**, **sfn** i **b** oraz instrukcje procedur **sf2ed**, **fr**, **af2sf**, **sf2af**, **fne** i **fnd**. Pierwsza procedura umożliwia wybranie pliku do szyfrowania lub deszyfrowania, druga pozwala usunąć niepotrzebny plik, trzecia szyfruje wybrany plik, czwarta wybrany plik deszyfruje, piąta szyfruje nazwę pliku wybranego do zaszyfrowania i ostatnia nazwę zaszyfrowanego pliku deszyfruje. Nazwą zaszyfrowanego pliku jest zaszyfrowana nazwa pliku wejściowego procedury szyfrowania. Dzięki temu nie wiadomo co zawiera plik zaszyfrowany. Strukturę algorytmów służących do szyfrowania/deszyfrowania można łatwo prześledzić przeglądając kod procedur **af2sf**, **sf2af**, **fne** i **fnd**. W procedurach **fne** i **fnd** zastosowano funkcję biblioteczną **convert/base** jako przekształcenie kryptograficzne. Przekształceniami kryptograficznymi w procedurze **af2sf** są funkcja **convert/base** i dodawanie liczb naturalnych. W procedurze **sf2af** zastosowano również funkcję **convert/base** jako przekształcenie kryptograficzne a drugim przekształceniem kryptograficznym jest tu odejmowanie liczb naturalnych. Zaszyfrowany plik jest wyjątkowo skutecznie chroniony przed nieupoważnionym dostępem, ponieważ przestrzeń tajnego klucza jest ogromna i zależy od rozmiaru pliku wejściowego. Plik zaszyfrowany jest zapamiętywany w tym samym folderze co plik niezaszyfrowany.

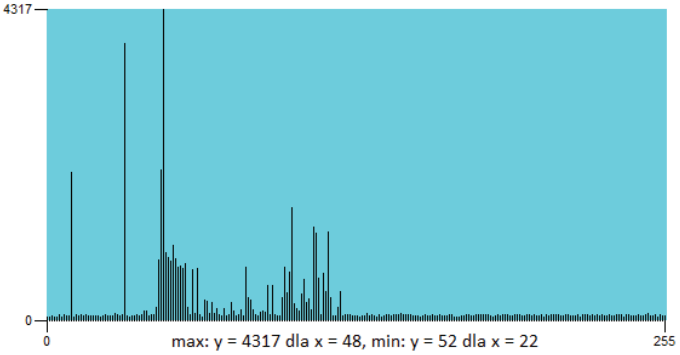
Graficzny interfejs użytkownika umożliwia wybranie trzech podstawowych zadań: **`Szyfrowanie pliku`**, **`Deszyfrowanie pliku`** i **`Deszyfrowanie nazwy pliku zawartego w pliku zaszyfrowanym`**. Po wykonaniu wybranego zadania w obszarze tekstowym interfejsu użytkownik otrzymuje wyczerpującą informację o wykonanym zadaniu, a plik wejściowy zostaje usunięty. Należy pamiętać, że tajny klucz jest zawarty w kodzie aplikacji i każdy użytkownik może na wiele sposobów zamontować swój własny klucz. Najprościej może on na przykład zmienić wartości zmiennych **b**, **sfn** i **sfs**. Poza tym aplikacja musi mieć prawo do zapisywania i usuwania przetwarzanych plików.
Drugą część kodu zawiera element **`Combo Box`**. Znajdują się tam przede wszystkim instrukcje warunkowe uzależnione od interakcji użytkownika, który wybiera określone zadanie do wykonania. Program jest dosyć skomplikowany i po oszczędnym wylistowaniu zajmuje cztery strony instrukcji języka Maple. Mimo to aplikację mogą używać nie tylko biegli znawcy programu Maple, ale też początkujący użytkownicy tego programu. Ze względów bezpieczeństwa aplikacja powinna być zapisana na pendrajwie, który należy skutecznie pilnować.

4. Przykład

Do zilustrowania działania aplikacji wybrano dwa pliki[2] dostępne w internecie: **plik rfc4648.txt** o rozmiarze 36496 bajtów i plik **rfc4648.pdf**, którego rozmiar wynosi 56198 bajtów.



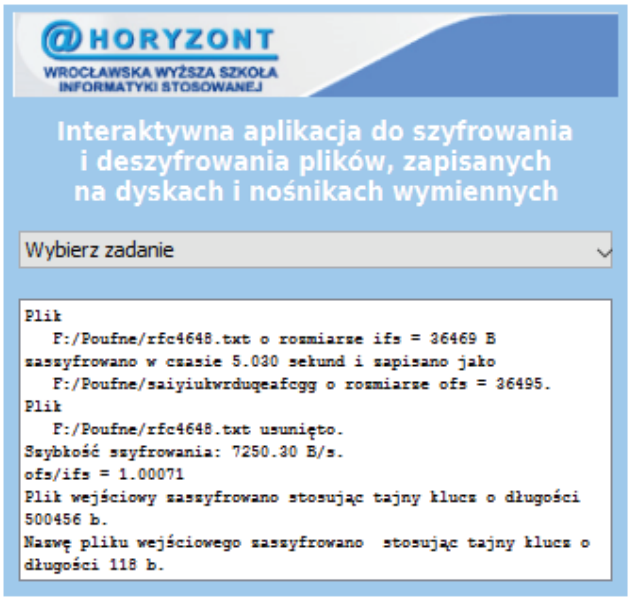
Ryc. 7. Histogram częstotliwości występowania znaków w pliku **rfc4648.txt**.



Ryc. 8. Histogram częstotliwości występowania znaków w pliku **rfc4648.pdf**.

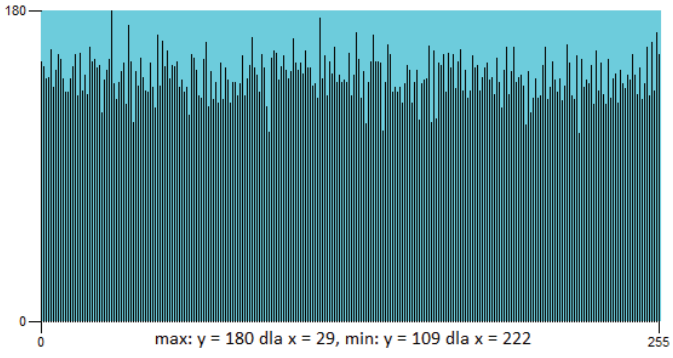
2. S. Josefsson, The Base 16, Base 32, and Base 64 Data Encodings, http://www.o-nline.com/nettools/rfc/rfcs/rfc4648.shtml.

Pierwszy plik zawiera tylko znaki 7-bitowe, w pliku drugim występują wszystkie znaki 8-bitowe. Są to więc dwa pliki różniące się znacznie strukturą znakową.
Najpierw zaszyfrowano za pomocą prezentowanej aplikacji plik tekstowy. Po wykonaniu zadania w polu tekstowym interfejsu pojawia się szczegółowy opis procesu szyfrowania. Podana jest m. in. wielkość tajnego klucza, posiadającego długość 500 456 bitów, przy pomocy którego plik został zaszyfrowany.



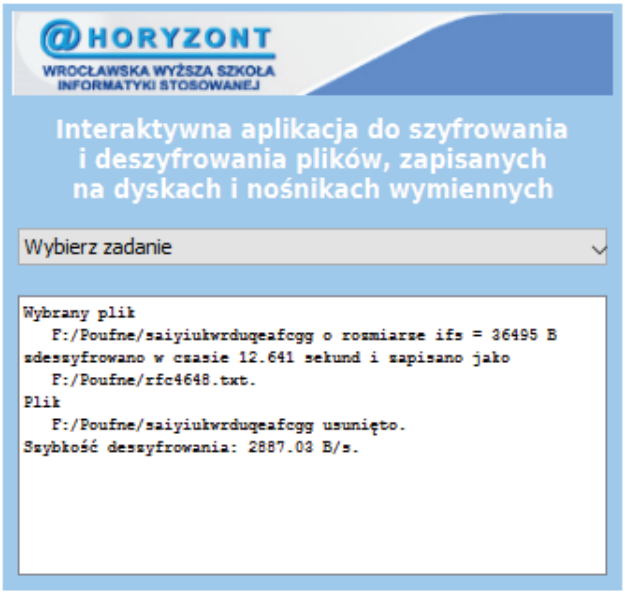
Ryc. 9. Graficzny interfejs użytkownika po wykonaniu zadania szyfrowania pliku rfc4648.txt.

Strukturę znakową zaszyfrowanego pliku ilustruje Ryc. 10. Można zauważyć, że histogramy częstotliwości występowania znaków dla pliku niezaszyfrowanego i zaszyfrowanego znacznie się różnią. Plik zaszyfrowany zawiera dosyć równomierny pseudolosowy ciąg znaków o wartościach bajtowych w zakresie



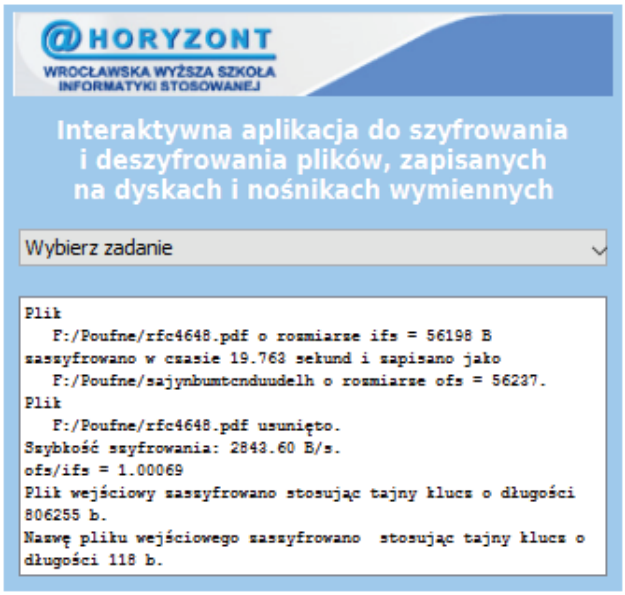
Ryc. 10. Histogram częstotliwości występowania znaków w pliku zaszyfrowanym saiyiukwrduqeafcgg.

0 .. 255. Podobnie po wykonaniu procesu deszyfrowania w polu tekstowym interfejsu można zobaczyć jaki jest rozmiar zaszyfrowanego pliku, jaką nazwę ma plik zdeszyfrowany i jaka jest szybkość deszyfrowania w bajtach/s (Ryc. 11.).

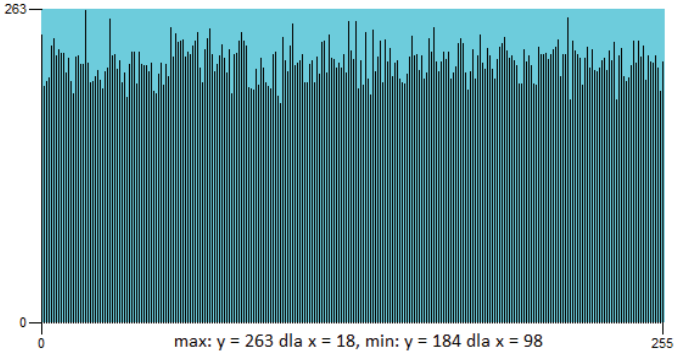


Ryc. 11. Graficzny interfejs użytkownika po wykonaniu zadania deszyfrowania pliku saiyiukwrduqeafcgg.

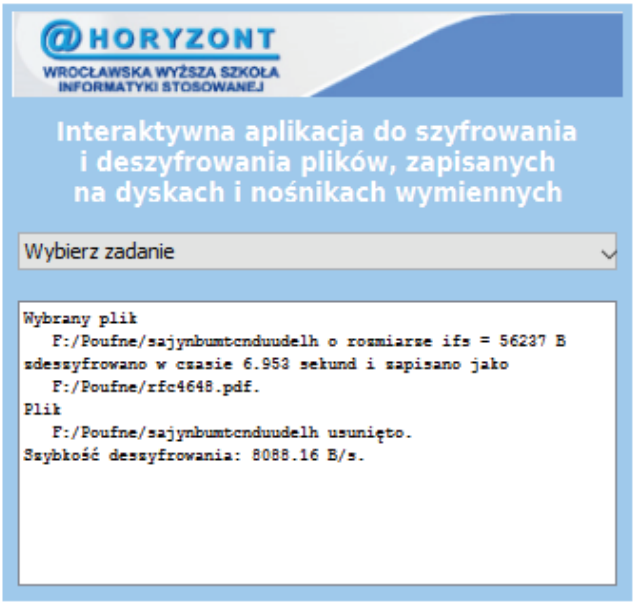
Proces szyfrowania i deszyfrowania drugiego pliku można prześledzić na Ryc. od 12 do 14.



Ryc. 12. Graficzny interfejs użytkownika po wykonaniu zadania szyfrowania pliku **rfc4648.pdf**.



Ryc. 13. Histogram częstotliwości występowania znaków w pliku zaszyfrowanym pliku **sajynbumtcnduudelh**.



Ryc. 14. Graficzny interfejs użytkownika po wykonaniu zadania deszyfrowania pliku **sajynbumtcnduudelh**.

Mimo że wejściowe pliki tekstowe mają bardzo różną strukturę, ich kryptogramy są bardzo podobne. Fakt ten świadczy o dużej mocy szyfru, zastosowanego w procedurach **af2sf** i **sf2af**.

4. Podsumowanie i wnioski

Opisano realizację stosunkowo prostych algorytmów kryptograficznych w których pozycyjny zapis liczb naturalnych o dowolnej wartości bazy jest przekształceniem kryptograficznym. Inaczej mówiąc, zrealizowano w środowisku Maple aplikację stosującą funkcję biblioteczną **convert/base** w procedurach szyfrujących i deszyfrujących, która „potrafi" zaszyfrować każdy plik przy zastosowaniu tajnego klucza o ogromnej długości, dzięki czemu plik jest praktycznie stuprocentowo chroniony przed nieupoważnionym dostępem. Poza tym aplikacja jest wyjątkowo „życzliwa" dla użytkownika, który posługuje się tylko myszą i nie wprowadza żadnych danych, ponieważ tajne klucze kryptograficzne są zawarte w kodzie aplikacji. Takie podejście może być źródłem bardzo wielu podobnych rozwiązań[3, 4, 5]. W pracy nie podano kodu programu typu worksheet o nazwie sdpdnw.mw, aby nie zwiększać objętości artykułu.

Literatura

[1] C. Kościelny, Realizacja szyfru „bezkluczowego" c80k395 do kryptograficznej ochrony załączników poczty elektronicznej w środowisku Maple, Biuletyn Naukowy Wrocławskiej Wyższej Szkoły Informatyki Stosowanej. Informatyka, 2013 3.

3. Maple Implementation of Transport Encryption Scheme Using the Secret Key of Length 479 Bits, http://www.maplesoft.com/applications/Author.aspx?mid=1638.
4. Base 64 "Keyless" File Encryption, http://www.maplesoft.com/aplications/Author.aspx?mid=16738.
5. Maple `Keyless` Base b Encryption Scheme,_http://www.maplesoft.com/applications/Author.aspx?mid=16738.

Training sequence decomposition

Dekompozycja ciągu uczącego

Swietłana Lebiediewa[1]

Treść: Sformułowano problem dekompozycji ciągu uczącego (CU). Zdefiniowano dwa rodzaje dekompozycji CU dla scentralizowanej bazy danych (SBD). Udowodniono twierdzenia dotyczące zajętości pamięci przez CU po dekompozycji. Oszacowano złożoność obliczeniową algorytmów dekompozycji. Przedstawiono wyniki eksperymentu obliczeniowego ilustrującego zajętość pamięci w zależności od rodzaju dekompozycji, redundancji cech w drzewie i na ścieżce oraz wysokości drzewa.

Słowa kluczowe: baza danych, rozpoznawanie wieloetapowe, ciąg uczący, dekompozycja

Abstract: The problem of decomposition of a training sequence (TS) is formulated. Two types of TS decomposition for a centralized database are formulated. Theorems concerning memory occupancy by the TS after decomposition are proved. The calculation complexity of decomposition algorithms is estimated. The results of a calculation experiment are presented that illustrate memory occupancy depending on the decomposition type, the redundancy of features in the tree, the path, and the tree height.

Keywords: database, multistage recognition, training sequence, decomposition

1. The problem formulated

Recognition algorithms with learning use the training sequence (TS) [1, 3, 4]. The training sequence is a sequence of properly classified patterns, the elements of the TS being (x_k,k) pairs, where x is the vector of the values of the features of the pattern, and k is the class number [1, 2, 4, 5]. An example is the decision tree (DT) in Figure 1 and the corresponding TS in Figure 2.

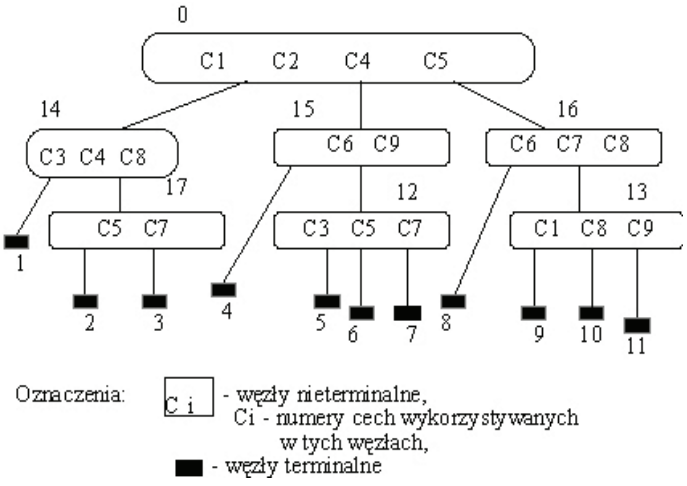


Figure 1. Decision tree No. 1a.

The training sequence presented in Figure 2 has the symbol * at the place of some features. This symbol means that a given feature is irrelevant to the pattern being recognized and not taken into account by the recognition algorithm. The multistage recognition process uses only some elements of the pattern feature vector at various recognition stages. For instance, recognition algorithms only use features C1, C2, C4, and C5 at node 0, and features C3, C4, and C8 at node 14. Additionally, certain features do not occur on each path at all.

C1	C2	C3	C4	C5	C6	C7	C8	C9	CLASS
16	3.44	3	6.55	40.0	48	116	*	14.9	7
15	3.45	2	8.45	40.0	*	*	120	*	1
14	2.4	1	8.33	38.1	*	110	*	*	2
18	3.0	*	7.5	35.0	40	*	*	15.0	4
15	2.7	*	6.0	20.0	35	110	112	*	8

Figure 2. A fragment of the training sequence for DT No. 1a.

Let us use the following designations: CN – the number of elements of feature vector C_n (n=1, 2, ..., CN); K – the number of classes; EN_k – the number of TS elements for class k; UCN_k – the number of features irrelevant to class k (features not used in the process of recognizing a pattern belonging to class k). The number of unused memory units is expressed by the following formula:

$$\sum_{k=1}^{K} LCN_k * LE_k \qquad (1)$$

For the decision tree in Figure 1, the number of all features $CN = 9$, the number of all classes $K = 11$. Assume that the TS element number is the same for every class, e.g. 100; then, under formula (1), the memory waste is 1900 units, i.e. more than 19%. The memory waste is the greater, the fewer features are used by the recognition algorithm on the path from the root of the decision tree to the non-terminal node that is the direct predecessor of the class. E.g., for tree No. 1b (Fig. 3), the memory waste under the same assumptions is over 54.11%.

1. Faculty of Computer Science, Wroclaw School of Information Technology, ul. Wejherowska 28,54-239 Wrocław, Poland, swietlana@lebiediewa.com

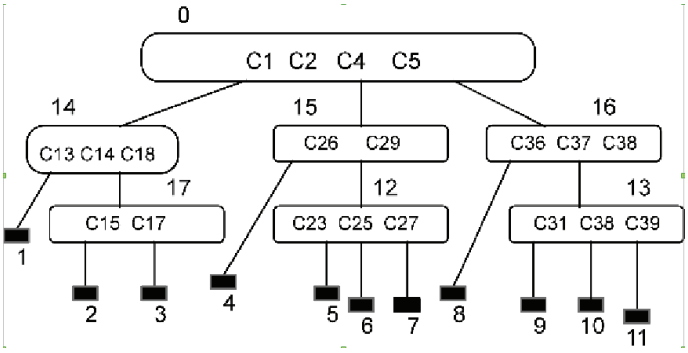


Figure 3. Decision tree No. 1b.

To conserve memory, it is proposed that the TS be decomposed. For a centralized database (CDB), I propose two types of decomposition : decomposition of the first type and decomposition of the second type. In decomposition of the first type, for every node that is the direct predecessor of a terminal node, a TS subsequence is formed that includes the values of only those features that are used along the entire path from the root of the decision tree (DT) to the node concerned until the decision is made to classify the pattern as a member of the class available directly from the node. In decomposition of the second type, for every non-terminal node, a training sequence subsequence is formed that includes the values of only those features that are used in the node.

It can be seen from the recognition algorithms presented in [5, 6] that the time of RA work at each node consists of the time needed to create the data segment and the RA work time. An appropriate form of the TS may shorten the time needed to create a data segment. The purpose of decomposition is to split the TS into subsequences that

1. minimize memory occupancy by the TS;
2. minimize the time needed for pattern recognition.

2. TS decomposition for a centralized database

Decomposition algorithms use information about the TS and the structure of the DT included in the SEQUENCE, CLASS, NODE, and FEATURE relations of the DB conceptual model. The SEQUENCE relation contains information about the TS. The CLASS relation describes the dependency between the class number and the number of the node that is the direct predecessor of the class. For every class, its predecessor is indicated. The NODE relation contains information about the number of every node, its direct successors and predecessors. The FEATURE relation contains information about which nodes use each feature [5, 6].

ALGORITHM 1. (decomposition algorithm 1 of the TS type)

Data: REC DB conceptual model – SEQUENCE and CLASS relations

To be found: For every node that is the direct predecessor of a terminal node, form a TS subsequence that includes the values of only those features that are used along the path from the root of the decision tree to the node concerned until the decision is made.

STEP 1. Take the numbers of nodes that are the direct predecessors of the terminal nodes from the CLASS relation.

STEP 2. Create a training subsequence for every such node: take rows from the TS for which CLASS NO = “class reachable directly from the node”,placing theresult in the *R* relation;

STEP 3. Map the *R* relation to attributes, the numbers of features used on the path from the root to the node and the CLASS NO).

STOP

Memory occupancy by a subsequence connected with node *i* is expressed by the following formula:

$$ZPD1_i = \sum_{k=1}^{K_i} (LC_i + 1) * LE_k \qquad (2)$$

We will designate memory occupancy by all subsequences of the training sequence obtained as a result of ALGORITHM 1 as *MOD1*. Memory occupancy by these subsequences as expressed by formula (3): where where J - the number of nodes that are direct predecessors classes:

$$ZPD1 = \sum_{i=1}^{J} \sum_{k=1}^{K_i} (LC_i + 1) * LE_k \qquad (3)$$

Decomposition of the first type leads to the most optimal form of the TS in terms of memory occupancy; every TS reduction obtained as a result of decomposition of the first type would lead to a loss of information. What may happen, however, is that the time needed to form a TS fragment used in a certain segment (FRAGSEQ*) from the TS obtained as a result of the decomposition of the first type can be longer than the time needed to form a FRAGSEQ* relation from an undecomposed TS. The time is longer as a result of the need to perform a larger number of operations.

In decomposition of the second type, for every non-terminal node, a training sequence subsequence is formed that includes the values of only those features that are used in the node.

ALGORITHM 2. (decomposition algorithm 2 of the TS type for the CDB)

Data: DB conceptual model – SEQUENCE, NODE, and FEATURE relations

To be found: For every non-terminal node, form a training sequence subsequence that

includes the values of only those features that are used in the node.

STEP 1. Add a column containing TS element identifiers to the SEQUENCE relation storing the TS.

STEP 2. Take the numbers of all non-terminal nodes from the NODE relation.

STEP 3. For every non-terminal node, form a training subsequence (take those row from the TS for which CLASS NO = " call reachable from a given node"; remove those attributes from the obtained relation that contain feature values unused in the node).

STOP

Let n be the number of rows of the TS. The TS memory complexity is given by the formula $O(n^2)$.

Denote the number of all non-terminal nodes of the DT as N; the number of features used by the recognition algorithm at node j as C_j; the number of classes reachable from node j as K_j; the number of TS elements for class k as EN_k; and memory occupancy by the training sequence obtained as a result of decomposition of the second type as *MOD2*. Then, the formula for FRAGSEQ* TS memory occupancy at node j (on the assumption that each sequence contains an additional column for the value of the training sequence element identifier) has the following form:

$$ZDD2_j = \sum_{k=1}^{K_j} (C_j + 2) * LE_k \qquad (4)$$

The formula for memory occupancy by all TSs obtained as a result of decomposition of the second type is as follows:

$$ZPD2 = \sum_{j=1}^{N} \sum_{k=1}^{K_j} (C_j + 2) * LE_k \qquad (5)$$

The following conclusions follow from the memory occupancy formulas:

Conclusion 1. Memory occupancy by TSs arising from decomposition of the first type is unaffected by the redundancy of features on the path from the tree root to the class or by the length of the path. It only depends on the number of features on the path.

Conclusion 2. *MOD2* is unaffected by the length of the DT feature vector. It only depends on the number of features used at each node and on the number of classes reachable from that node.

Conclusion 3. *MOD2* depends both on the redundancy of features on the path from the root to the class or on the length of the path.

We denote by TMO total memory contents for the TS, and by UCN - number of characteristics unused (irrelevant) in the process of recognition.

Theorem 1. Memory occupancy by TSs obtained as a result of decomposition of the first type is no greater than memory occupancy by the TSs before the decomposition, so the following dependency occurs:

$$MOD1 \leq TMO$$

(the inequality is sharp if $UCN \neq 0$)

Theorem 2. Memory occupancy by TSs obtained as a result of decomposition of the first type is always smaller than memory occupancy by the TSs obtained as a result of decomposition of the second type, so the following dependency occurs:

$$MOD1 < MOD2$$

3. Calculation experiment

A calculation experiment was carried out examining memory occupancy of TSs obtained as a result of decomposition of the first and of the second type for various trees, depending on the number of features in the tree and on the path, the height of the tree, and earlier-stage recognition. The following trees were considered: No. 4a (Fig. 4a), No. 4b (Fig. 4b), and No. 4c (Fig. 4c). The number of features: 20; the number of classes: 10; tree height: 3; feature use on the path: 47%; TS size before decomposition: 21000. Tree No. 4a has no redundancy of features on the path. Tree No. 4b has the redundancy of two features at nodes on different paths. Tree No. 4c has the redundancy of three features at nodes on different paths. Memory occupancy by TSs obtained as a result of decomposition of the first type and the memory savings after decomposition are presented in Table 4.4a. Memory occupancy by TSs obtained as a result of decomposition of the second type and the memory savings after decomposition are presented in Table 4.4b. A chart showing the TS memory savings after decomposition of first and the second type for trees No. 4a–4c is presented in Figure 5. Memory occupancy by TSs obtained as a result of decomposition of the first type and the memory savings after decomposition are presented in Table 4.5 a. Memory occupancy by TSs obtained as a result of decomposition of the second type and the memory savings after decomposition are presented in Table 4.5b. A chart showing the TS memory savings after decomposition of first and the second type for trees No. 5a–5c is presented in Figure 4.16.

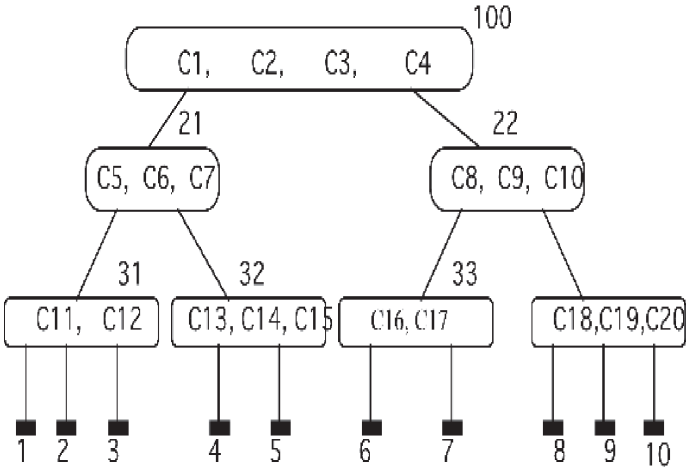


Figure 4a. Decision tree No. 4a.

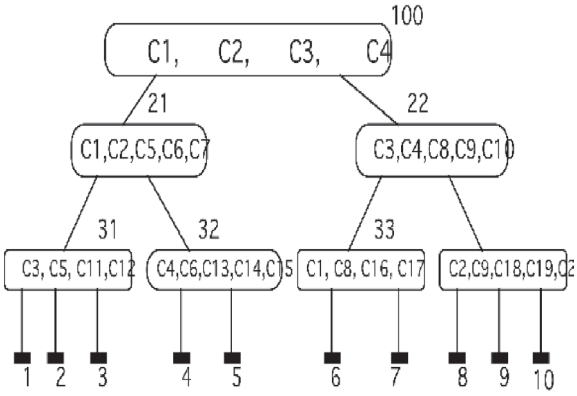


Figure 4b. Decision tree No. 4b.

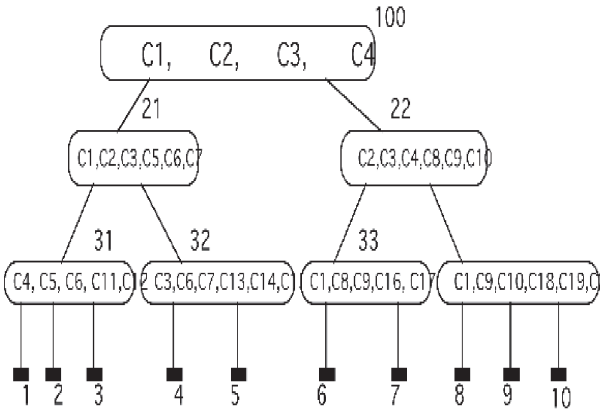


Figure 4c. Decision tree No. 4c.

Table 4.4a. Memory occupancy by TSs obtained as a result of decomposition of the first type (CDB).

TREE NO.	Tree No. 4a	Tree No. 4b	Tree No. 4c
MEMORY SAVINGS	50%	50%	50%

Table 4.4b. Memory occupancy by TSs obtained as a result of decomposition of the second type (CDB).

TREE NO.	Tree No. 4a	Tree No. 4b	Tree No. 4c
MEMORY SAVINGS	26.19%	7.14%	(2.38%)

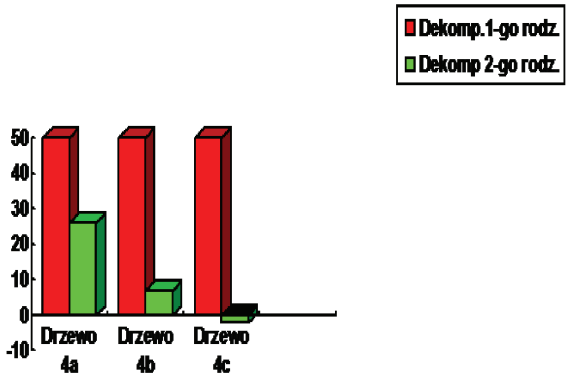


Figure 4.5. A chart showing the TS memory savings after decomposition of first and the second type for trees No. 4a–4c.

Decomposition of the first type always results in memory savings, especially where the number of all features in a DT is much greater than the number of features used on any path (cf. tree No. 1c in Figure 3). Decomposition of the second type is not always favourable in terms of memory occupancy. Decomposition of the second type gives the best results if there is no redundancy of features on any path. The smaller the percentage of feature use on a path, the greater the memory savings upon decomposition both of the first and of the second type.

A smaller percentage of feature use on a path improves the results of decomposition both of the first and of the second type. Even in the case of the redundancy of four features on a path, there are no memory wastes. The result is even better where some of the patterns are recognized in the first stage.

We observe clear memory savings for decomposition of first type. For decomposition of the second type, memory savings occur if there is no redundancy. In the case of the redundancy of four features on a path, TSs obtained as a result of decomposition of the second type require more memory than undecomposed TSs. If some patterns are recognized at the first level, there are clear memory savings for decomposition of both the first type and the second type even in the case of the redundancy of features on a path. A binary tree is a good example indicating a tendency for memory occupancy to be reduced in the case of a decomposed TS. Memory savings for two-, four-, six-, and eight-level binary trees are shown in Figure 4.

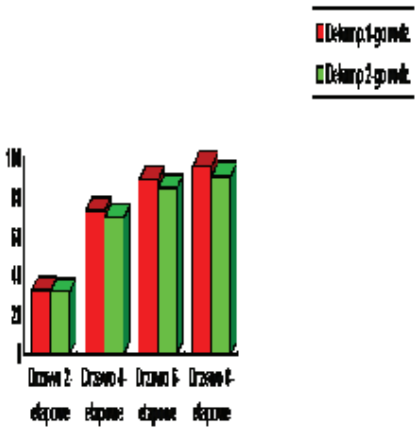


Figure 4. A chart showing the TS memory savings (percentage) after decomposition of the first type and the second type for two-, four-, six-, and eight-level binary trees.

4. Conclusions

The examples cited show that the greatest memory saving effect results from the use of decomposition of the first type in the case of a CDB because decomposition of the first type is not sensitive to feature redundancies on a path or to path lengths. Particularly good results are obtained in the case of a smaller number of features on a path relative to the number of features in the DT. Decomposition of the second type does not always result in memory savings. Decomposition of the second type requires more memory than decomposition of the first type. The best results are given by decomposition of the second type in the case of no feature redundancy and 'longer' trees. For both decompositions, the recognition of a certain number of classes

at earlier stages has a positive effect on memory savings. With large numbers of features and 'long' trees, both decompositions have a very large effect on memory savings. Training sequences obtained as a result of decomposition of the second type have the additional advantage that they are identical with fragments of TSs used by recognition algorithms at the node, so if decomposition of the second type is used, it is unnecessary to create special FRAGSEQ*i* relations (*i* being the node number) when creating an external model, which means a considerably shorter processing time.

References

[1] Bubnicki, Z. 'Knowledge-Based Approach as a Generalization of Pattern Recognition Problems and Methods'. *Systems Science* Vol. 19, No 2 (1993), pp. 5–21.
[2] Flasiński, M. *Wstęp do sztucznej inteligencji*. Warsaw: PWN, 2011.
[3] Kurzyński, M. *Algorytmy rozpoznawania wieloetapowego oraz ich zastosowania medyczne i techniczne*. Wrocław: Wyd. PWr., 1987.
[4] Józefczyk, J. 'Rozpoznawanie i zastosowania biomedyczne' [in:] *Problemy automatyki i informatyki*, Wrocław: Wyd. Ossolineum, (1998), pp. 45–58.
[5] Lebiediewa, S, 'System informatyczny dla wieloetapowego rozpoznawania obiektów'. *Biuletyn Naukowy WWIS. Informatyka,* 2014.
[6] Lebiediewa, S. *Metodologia projektowania problemowo zorientowanych baz danych do systemów wielostopniowego podejmowania decyzji*. Wrocław: Wyd. Pwr., 1998.
[7] Lebiediewa, S., Zarzycki, H., and Dobrosielski, W. T. 'A new approach to the equivalence of relational and object-oriented databases', [in] Novel Developments in Uncertainty Representation and Processing, Springer International Publishing (2016), pp 85-93.

Approximation algorithm supported on minimizing the Kullback-Leibler information divergence in some class of dynamical systems (October 2015)

Algorytm aproksymacyjny w oparciu o informację Kullbacka-Leiblera w pewnej klasie systemów dynamicznych

Jan Owedyk[1], Zdzisław Mathia[2], Hubert Zarzycki[3]

Treść: W pracy przedstawiono algorytm, który umożliwia skonstruowanie przybliżonych rozwiązań dla pewnej klasy systemów dynamiczych opisujących ewolucję w czasie gęstości prawdopodobieństwa. Przybliżone rozwiązania otrzymujemy minimalizując informację Kullbacka-Leiblera przy dodatkowych warunkach.
Wykazano, że pochodna informacji Kullbacka-Leiblera dla dokładnych i przybliżonych rozwiązań jest opisana przez tą samą formułę. W konsekwencji gdy w dynamicznym systemie maleje informacja Kullbacka-Leiblera dla dokładnych rozwiązań to także maleje dla przybliżonych rozwiązań.

Słowa kluczowe: Algorytm aproksymacyjny, Informacja Kullbacka-Leiblera, Metoda minimalizacji, Równania Fokkera-Plancka

Abstract: In this work an algorithm is presented for creating approximate solutions in some class of dynamical systems describing the time evolution probability densities. The approximate solutions are obtained by minimizing Kullback-Leibler divergence under some constrains.
It is shown that the derivatives of the Kullback-Leibler divergence for exact solutions and for approximate solutions are described by the same formula. In consequence if in a dynamical system the Kullback-Leibler divergence decreases in time for exact solutions, it also decreases for approximate solutions.

Keywords: Approximation algorithm, Kullback-Leibler divergence, Minimization methods, Fokker-Planck equation

I. Introduction

In the early eighties of the last century the Kullback-Leibler divergence was used to obtain approximate solutions in dynamical systems represented by *n*-dimentional Fokker-Planck equation [4] also with time dependent drift and diffusion coefficients [6] and in systems represented by master equations [5,7].
In this paper this approach is generalised for some dynamical systems in that the Kullback-Leibler information divergence $K(P,P_0) = \int P \ln(P/P_0)\, d\boldsymbol{x} \geq 0$, [2,3] satisfies inequality $dK/dt \leq 0$ for any two probability density functions $P = P(\boldsymbol{x},t)$, $P_0 = P_0(\boldsymbol{x},t)$ of continuous random variable defined on $\boldsymbol{x} = (x_1,..., x_n)$, which describe time evolution in the system. We have formulated a criterion on choosing such a dynamical system which will be called the Kullback-Leibler system.
In the dynamical system an approximate solution as an exponential probability density is postulated

$$P^* = P_0 \exp\Big(-\lambda_0(t) - \sum_{i=1}^{N} \lambda_i(t) f_i\Big)$$

obtained by minimizing the Kullback-Leibler divergence under some constrains. The functions $\lambda_i(t)$, $i=0,1,...,N$ are the Lagrange multipliers and their time evolution is determined by some system of ordinary differential equations and $f_i = f_i(\boldsymbol{x})$, $i = 1,...,N$ are some lineary independent functions. The probability density $P_0 = P_0(\boldsymbol{x},t)$ is a fixed exact solution of the Kullback-Leibler system.
This paper is organised as follows. In section 2 a definition of the Kullback-Leibler system is formulated and the stability of exact solutions in this system is investigated. In section 3 the approximate solution is obtained for the Kullback-Leibler system by minimizing Kullback-Leibler divergence under some conditions. In section 4 the stability of the approximate solutions is investigated. In section 5 the accompanied differential evolution equations for $\lambda_i(t)$, $i = 1,...,N$ are derived and investigated. Two important properties of solutions to this accompanied differential evolution equations are formulated. An optimization algorithm supported on minimizing the Kullback-Leibler divergence in the Kullback-Leibler systems is also formulated.

1. Department of Informatics, Kujawy and Pomorze University, Bydgoszcz, Poland j.owedyk@kpsw.edu.pl
2. Department of Informatics, Kujawy and Pomorze University, Bydgoszcz, Poland
3. Faculty of Computer Science, Wroclaw School of Information Technology, ul. Wejherowska 28,54-239 Wrocław, Poland, hzarzycki@horyzont.eu

II. The Kullback-Leibler Systems

We take for our considerations some dynamical systems describing by the equation

$$\partial P/\partial t = S_t P, \quad (2.1)$$

where $P = P(\boldsymbol{x},t)$ is the time dependent probability density function of the continuous random variable defined on $\boldsymbol{x} = (x_1,\ldots, x_n)$ $\boldsymbol{E}^n$ ($\boldsymbol{E}^n$ *is the n-dimensional Euclidean space),* S_t is a tin $\in$ dependent system operator. In the case of the Fokker-Planck equation, S_t is given by equation (A.2) (see appendix).

For ensuring the normalization condition $\int P(\boldsymbol{x},t)\, d\boldsymbol{x} = 1$ it must be satisfied

$$\int S_t P\, d\boldsymbol{x} = 0 \quad (2.2)$$

for any probability density function $P(\boldsymbol{x},t)$.

We use the Kullback-Leibler divergence [2,3]

$$K(P,P_0) = \int P \ln(P/P_0)\, d\boldsymbol{x} \geq 0 \quad (2.3)$$

as "a measure of distance" between any two solutions $P = P(\boldsymbol{x},t)$, $P_0 = P_0(\boldsymbol{x},t)$ of equation (2.1).

In order to investigate time dependence of Kullback-Leibler divergence we calculate its time derivative

$$dK/dt = \int ((\partial P/\partial t)\ln(P/P_0) + P\partial(\ln(P/P_0))/\partial t)d\boldsymbol{x} = \int ((\partial P/\partial t)\ln(P/P_0) + (\partial P/\partial t) - (\partial P_0/\partial t)(P/P_0))d\boldsymbol{x} = \int((S_t P)\ln(P/P_0) - (S_t P_0)(P/P_0))d\boldsymbol{x}. \quad (2.4)$$

Above we have used normalisation to the unity i.e. $\int P\, d\boldsymbol{x} = 1$ and equations $\partial P/\partial t = S_t P$, $\partial P_0/\partial t = S_t P_0$ (P and P_0 satisfy (2.1)).

We will restrict ourselves to the system operator S_t so that the last formula in (2.4) is negative for any P and P_0 i.e.

$$\int((S_t P)\ln(P/P_0) - (S_t P_0)(P/P_0))d\boldsymbol{x} \leq 0, \quad (=0 \text{ only if } P = P_0). \quad (2.5)$$

The system described by the system operator S_t which satisfies inequality (2.5) will be called Kullback-Leibler system and the system operator S_t will be called Kullback-Leibler system operator. It is shown in Appendix that systems described by the Fokker-Planck equation are the Kullback-Leibler systems. From now we will consider only Kullback-Leibler systems and Kullback-Leibler system operators S_t.

According to the inequality (2.5) it follows that

$$dK/dt \leq 0, \quad (=0 \text{ only if } P = P_0). \quad (2.6)$$

The inequality (2.6) may be treated as a generalised H-theorem [6]. We can see that the inequality (2.5) is a criterion of choosing dynamical systems in which the generalised H-theorem is satisfied.

Using the above inequality one can investigate an asymptotic behaviour of solutions of the equation (2.1). Because the Kullback-Leibler divergence $K(P,P_0)$ (see (2.3)) is bounded from below and (2.6) is satisfied, one can write

$$\lim_{t\to\infty} dK/dt = 0.$$

If additionally from (2.7) and (2.5) it follows that

$$\lim_{t\to\infty} (P(\boldsymbol{x},t)/P_0(\boldsymbol{x},t)) = 1, \text{ for every } \boldsymbol{x}, \quad (2.8)$$

then because the probability densities are normed to the unity, the difference between two arbitrary solutions $P = P(\boldsymbol{x},t)$, $P_0 = P_0(\boldsymbol{x},t)$ of equation (2.1) vanishes as time goes to infinity, i.e.

$$\lim_{t\to\infty} (P(\boldsymbol{x},t) - P_0(\boldsymbol{x},t)) = 0, \text{ for every } \boldsymbol{x}. \quad (2.9)$$

III. The Minimizing Kullback-Leibler Divergence Solutions

Let $P_0(\boldsymbol{x},t)$ be some fixed solution of equation (2.1). An arbitrary solution $P(\boldsymbol{x},t)$ of the equation (2.1) may be written in the form

$$P = P(\boldsymbol{x},t) = P_0(\boldsymbol{x},t)\, exp(-F(\boldsymbol{x},t)), \quad (3.1)$$

where $F = F(\boldsymbol{x},t)$ is some function.

From (3.1) and (2.1) we obtain

$$\partial P/\partial t = (\partial P_0/\partial t)\, exp(-F) - (\partial F/\partial t)\, P_0 exp(-F) \quad (3.2)$$

and

$$(\partial F/\partial t)\, P = (S_t P_0)\, exp(-F) - S_t P. \quad (3.3)$$

From (3.3) and (3.1) we have

$$(\partial F/\partial t) = (S_t P_0)/P_0 - (S_t P)/P \quad (3.4)$$

and

$$(\partial F/\partial t) = (S_t P_0)/P_0 - (S_t (P_0\, exp(-F)))\, /\, (P_0 exp(-F)). \quad (3.5)$$

Equation (3.5) determines time evolution of the function $F = F(\boldsymbol{x},t)$.

We assume that the datas of the system are represented by mean values

$$\langle f_i \rangle_P = \int f_i(\boldsymbol{x})\, P(\boldsymbol{x},t)\, d\boldsymbol{x}, \quad i = 1,\ldots,N \quad (3.6)$$

of N linearly independent (together with $f_0 = f_0(\boldsymbol{x}) = 1$) functions $f_i = f_i(\boldsymbol{x})$, $i = 1,\ldots,N$. According to the (3.6), (2.1) and (3.1) the evolution equations for the mean values are in the following form

$$d\langle f_i \rangle_P/dt = \int f_i\, \partial P/\partial t\, d\boldsymbol{x} = \int f_i S_t P\, d\boldsymbol{x} = \int f_i S_t (P_0 exp(-F))\, d\boldsymbol{x}, \quad i = 1,\ldots, N, \quad (3.7)$$

finally

$$d\langle f_i \rangle_P/dt = \int f_i S_t (P_0 exp(-F))\, d\boldsymbol{x}, \quad i = 1,\ldots, N. \quad (3.8)$$

Using (3.3) and (3.1) we obtain useful formulas

$$\langle f_i\, \partial F/\partial t \rangle_p = \int f_i\, \partial F/\partial t\, P\, d\boldsymbol{x} = \int f_i (S_t P_0)\, exp(-F)\, d\boldsymbol{x} - \int f_i S_t (P_0 exp(-F))\, d\boldsymbol{x}, \; i = 0,1,\ldots,N. \quad (3.9)$$

For $i = 0$ we remember that $f_0 = f_0(\boldsymbol{x}) = 1$ then from (3.9) we have

$\langle \partial F/\partial t \rangle_p = \int (S_t P_0) \exp(-F)\, d\boldsymbol{x} - \int S_t (P_0 \exp(-F))\, d\boldsymbol{x}.$ (3.10)

From (3.1), (2.1) and normalization condition $\int P(\boldsymbol{x},t)\, d\boldsymbol{x} = 1$ we have

$\int S_t (P_0 \exp(-F))\, d\boldsymbol{x} = \int S_t P\, d\boldsymbol{x} = \int \partial P/\partial t\, d\boldsymbol{x} = d(\int P d\boldsymbol{x})/dt = 0,$ (3.11)

then

$\langle \partial F/\partial t \rangle_p = \int (S_t P_0) \exp(-F)\, d\boldsymbol{x}.$ (3.12)

In general the evolution equations (3.8) are not closed with respect to the mean values $\langle f_i \rangle_P = \int f_i(\boldsymbol{x}) P(\boldsymbol{x},t)\, d\boldsymbol{x}$ of functions $f_i = f_i(\boldsymbol{x})$, $i = 1,\ldots,N$. In order to close and solve the system of evolution equations (3.8) we use the approximate exponential probability density $P^*(\boldsymbol{x},t)$ instead of the exact solution $P(\boldsymbol{x},t)$ i.e.

$P^*(\boldsymbol{x},t) = P_0(\boldsymbol{x},t) \exp(-F^*(\boldsymbol{x},t)),$ (3.13)

where

$F^*(\boldsymbol{x},t) = \lambda_0(t) + \sum_{i=1} \lambda_i(t) f_i(\boldsymbol{x}),$ (3.14)

where $\lambda_i(t)$, $i = 0,1,\ldots, N$ are Lagrange multipliers.

The approximate exponential probability density $P^*(\boldsymbol{x},t)$ is obtained by minimizing Kullback-Leibler information divergence

$K(P(\boldsymbol{x},t),P_0(\boldsymbol{x},t)) = \int P(\boldsymbol{x},t) \ln(P(\boldsymbol{x},t)/P_0(\boldsymbol{x},t))d\boldsymbol{x} \geq 0$ (3.15)

under the constraints

$\int P(\boldsymbol{x},t)\, d\boldsymbol{x} = 1,$ (3.16)

and

$\int f_i(\boldsymbol{x})\, P(\boldsymbol{x},t)\, d\boldsymbol{x} = \langle f_i \rangle_P, \quad i = 1,\ldots, N.$ (3.17)

The method for solving this constrained optimization problem is to use the Lagrange multipliers for each of the constraints and minimize the functional

$$J = \int P \ln(P/P_0) d\boldsymbol{x} + \sum_{i=0}^{N} \lambda_i(t)\left(\int f_i P\, d\boldsymbol{x} - \langle f_i \rangle_P\right) \quad (3.18)$$

with respect to P. Minimizing the functional (3.18) with respect to P leads to the calculation of the derivative of

$$L = P \ln(P/P_0) + \sum_{i=0}^{N} \lambda_i(t) f_i P \quad (3.19)$$

with respect to P and setting it to the zero i.e.

$$\partial L/\partial P = \ln(P/P_0) + 1 + \sum_{i=0}^{N} \lambda_i(t) f_i = 0. \quad (3.20)$$

From (3.20) one obtains the probability density P_{min} which minimizes the functional (3.18)

$$P_{min} = P_0 \exp(-(1+\lambda_0(t)) - \sum_{i=1}^{N} \lambda_i(t) f_i). \quad (3.21)$$

After replacing $1+\lambda_0(t)$ by $\lambda_0(t)$, one obtains from (3.21) the approximate exponential probability density $P^*(\boldsymbol{x},t)$ i.e.

$$P^* = P_0 \exp(-\lambda_0(t) - \sum_{i=1}^{N} \lambda_i(t) f_i). \quad (3.22)$$

The approximate exponential probability density $P^*(\boldsymbol{x},t)$ will be called the minimizing Kullback-Leibler divergence solution or short the Kullback-Leibler solution.

Inserting in (3.8) the Kullback-Leibler solution $P^*(\boldsymbol{x},t)$ instead of $P(\boldsymbol{x},t)$ we obtain

$d\langle f_i \rangle_{P^*}/dt = \int f_i S_t (P_0 \exp(-F^*))\, d\boldsymbol{x}, \quad i = 1,\ldots,N,$ (3.23)

where

$\langle f_i \rangle_{P^*} = \int f_i(\boldsymbol{x})\, P^*(\boldsymbol{x},t)\, d\boldsymbol{x}, \quad i = 1,\ldots,N.$ (3.24)

Eqations (3.23) determine approximate Kullback-Leibler solutions $P^*(\boldsymbol{x},t)$.

In (3.23) we asume that $P(\boldsymbol{x},t)$ and $P^*(\boldsymbol{x},t)$ have the same mean values $\langle f_i \rangle_P$ and $\langle f_i \rangle_{P^*}$ for initial time. From (3.24) we may calculate $\lambda_i(t)$, $i = 1,\ldots,N$ as functions of the mean values $\langle f_i \rangle_{P^*}$, $i = 1,\ldots,N$ and $\lambda_0(t)$ is a function of $\lambda_i(t)$, $i = 1,\ldots,N$ calculated from the normalization condition $\int P^*(\boldsymbol{x},t)\, d\boldsymbol{x} = 1$, then eqations (3.23) constitute a closed system of non autonomous ordinary differential equations for $\langle f_i \rangle_{P^*}$, $i = 1,\ldots,N$. Let us notice that (3.23) together with (3.13), (3.14) and (3.24) determine the differential evolution equations for $\lambda_i(t)$, $i = 1,\ldots,N$ further called the accompanied evolution equations.

Now we present formulas satisfied by the Kullback-Leibler solution $P^*(\boldsymbol{x},t)$ which will be useful in further considerations.

The first. From (2.2) it follows that

$\int S_t P^*\, d\boldsymbol{x} = 0.$ (3.25)

The second. From (3.24), (3.13) and (2.1) one gets

$d\langle f_i \rangle_{P^*}/dt = \int f_i (\partial P_0/\partial t) \exp(-F^*)\, d\boldsymbol{x} - \int f_i P_0 \exp(-F^*)(\partial F^*/\partial t)\, d\boldsymbol{x} = \int f_i (S_t P_0) \exp(-F^*) d\boldsymbol{x} - \int f_i P^* (\partial F^*/\partial t)\, d\boldsymbol{x}, \; i = 1,\ldots, N.$ (3.26)

From (3.26) and (3.23) we finally have

$\langle f_i \partial F^*/\partial t \rangle_{P^*} = \int f_i (S_t P_0) \exp(-F^*)\, d\boldsymbol{x} - \int f_i S_t (P_0 \exp(-F^*))\, d\boldsymbol{x}, \; i = 1,\ldots, N.$ (3.27)

The third. From (3.27) it follows that

$\langle \partial F^*/\partial t \rangle_{p^*} = \int \partial F^*/\partial t\, P^*\, d\boldsymbol{x} = \int (S_t P_0) \exp(-F^*)\, d\boldsymbol{x}.$ (3.28)

One can notice that formulas (3.25), (3.27), (3.28) which

are satisfied for the Kullback-Leibler solution $P^*(\boldsymbol{x}, t)$ are also fulfiled for exact solution $P(\boldsymbol{x}, t)$ see (2.2), (3.9), (3.12).

IV. Stability of The Kullback-Leibler Solutions

We will investigate whether for the Kullback-Leibler solutions $P^*(\boldsymbol{x}, t)$ like for the exact solutions $P(\boldsymbol{x}, t)$, the generalised H-theorem (2.6) and property (2.9), i.e. $\lim_{t\to\infty}(P^*(\boldsymbol{x},t)-P_0(\boldsymbol{x},t))=0$ are satisfied.

For our consideration we take the Kulback-Leibler divergence in the following form

$$K(P^*(\boldsymbol{x},t),P_0(\boldsymbol{x},t)) = \int P^*(\boldsymbol{x},t)\, ln(P^*(\boldsymbol{x},t)/P_0(\boldsymbol{x},t))\, d\boldsymbol{x} \geq 0. \tag{4.1}$$

We calculate its time derivative. According to the (3.13), (3.14), (3.28), (3.23), (3.25) one obtains

$$dK/dt = \int ((\partial P^*/\partial t) ln(P^*/P_0) + P^* \partial ln(P^*/P_0)/\partial t) d\boldsymbol{x} =$$

$$-\int (\partial P^*/\partial t)\, F^*\, d\boldsymbol{x} - \int P^*(\partial F^*/\partial t) d\boldsymbol{x} =$$

$$-\int (\partial P^*/\partial t)\, (\lambda_0(t) + \sum_{i=1}^{N} \lambda_i(t) f_i(\boldsymbol{x})) d\boldsymbol{x} - \langle \partial F^*/\partial t \rangle_{p^*} =$$

$$-\sum_{i=1}^{N} \lambda_i(t) (d\langle f_i \rangle_{P^*}/dt) - \langle \partial F^*/\partial t \rangle_{p^*} =$$

$$-\sum_{i=1}^{N} \lambda_i(t) \int f_i S_t (P_0 exp(-F^*)\, d\boldsymbol{x} - \int (S_t P_0) exp(-F^*)$$

$$d\boldsymbol{x} = -\int F^* S_t P^*\, d\boldsymbol{x} - \int (S_t P_0) exp(-F^*)\, d\boldsymbol{x} =$$

$$\int (S_t P^*)\, ln(P^*/P_0)\, d\boldsymbol{x} - \int (S_t P_0)\, (P^*/P_0)\, d\boldsymbol{x} = \int ((S_t P^*)$$

$$ln(P^*/P_0) - (S_t P_0)\, (P^*/P_0))\, d\boldsymbol{x}. \tag{4.2}$$

The above formula (4.2) for approximate solutions $P^*(\boldsymbol{x},t)$ which do not satisfy equation (2.1) is the same as the formula (2.4) obtained exact solutions $P(\boldsymbol{x},t)$ of equation (2.1).

According to the (2.5) the last formula in (4.2) fulfills the following inequality

$$\int ((S_t P^*)\, ln(P^*/P_0) - (S_t P_0)\, (P^*/P_0))\, d\boldsymbol{x} \leq 0, \quad (=0 \text{ only if } P^* = P_0). \tag{4.3}$$

Then from (4.2) and (4.3) it follows that

$$dK/dt \leq 0, \quad (=0 \text{ only if } P^* = P_0). \tag{4.4}$$

The inequality (4.4) is a generalised H-theorem for the Kullback-Leibler solutions $P^*(\boldsymbol{x},t)$.

Using the inequality (4.4) one can investigate an asymptotic behaviour of the Kullback-Leibler solutions $P^*(\boldsymbol{x},t)$. Because the Kullback-Leibler information divergence $K(P^*,P_0)$ is bounded from below and (4.4) is fulfilled, one can write

$$\lim_{t\to\infty} dK/dt = 0. \tag{4.5}$$

From (4.2), (4.3) and (4.5) it follows

$$\lim_{t\to\infty}(P^*(\boldsymbol{x},t)/P_0(\boldsymbol{x},t)) = 1. \tag{4.6}$$

Because the probability densities are normed to the unity, then according to (4.6) it follows that the difference between the two probability densities $P^* = P^*(\boldsymbol{x},t)$ and $P_0 = P_0(\boldsymbol{x},t)$ vanishes as time goes to infinity, i.e

$$\lim_{t\to\infty} (P^*(\boldsymbol{x},t) - P_0(\boldsymbol{x},t)) = 0. \tag{4.7}$$

Additionally from (4.6) and (3.13)

$$\lim_{t\to\infty} F^*(\boldsymbol{x},t) = 0. \tag{4.8}$$

The above considerations are done under the assumption that the Kullback-Leibler solutions $P^*(\boldsymbol{x},t)$ exist for any time $t > t_0$ (t_0 *is an initial time).*

Remark: It is an important result for the Kullback-Leibler divergence in this paper, that from (4.2) we have $dK/dt=\int((S_t P^*)ln(P^*/P_0)-(S_t P_0)(P^*/P_0))dx$ and from (2.4) $dK/dt=\int((S_t P)ln(P/P_0)-(S_t P_0)(P/P_0))dx$. We can see that the time derivative of Kullback- Leibler divergence for approximate solutions $P^*(x, t)$ and for exact solutions $P = P(x,t)$ are described by the same formula. According to the above, we can conclude that the derivative of the Kullback-Leibler divergence for approximate solutions and the derivative of the Kullback-Leibler divergence for exact solutons, fulfill the same inequalities (4.4) and (2.6). In consequence the approximate solutions $P^*(x, t)$ have the same asymptotic behaviour as the exact solutions $P = P(x,t)$.

V. The Accompanied Evolution Equations and The Optimization Algorithm

In order to derive the accompanied differential evolution equations for $\lambda_i(t)$, $i = 1,...,N$ we start from the equations (3.27) i.e.

$$\langle f_i \partial F^*/\partial t \rangle_{p^*} = \int f_i (S_t P_0) exp(-F^*)\, d\boldsymbol{x} - \int f_i S_t (P_0 exp(-F^*))\, d\boldsymbol{x}, \quad i = 1,...,N. \tag{5.1}$$

Using (3.14) on the left side of (5.1) one obtains

$$\langle f_i \partial F^*/\partial t \rangle_{P^*} = \langle f_i (d\lambda_0/dt + \sum_{j=1}^{N} (d\lambda_j/dt) f_j) \rangle_{P^*} = d\lambda_0/dt$$

$$\langle f_i \rangle_{P^*} + \sum_{j=1}^{N} (d\lambda_j/dt) \langle f_i f_j \rangle_{P^*},$$

$$i = 1,...,N. \tag{5.2}$$

For further calculations we use formula (3.28) i.e.

$$\langle \partial F/\partial t \rangle_{p^*} = \int (S_t P_0) exp(-F^*)\, d\boldsymbol{x}. \tag{5.3}$$

Using (3.14) on the left side of (5.3) one obtains

$$<\partial F/\partial t>_{p^*} = d\lambda_0/dt + \sum_{j=1}^{N} (d\lambda_j/dt) <f_j>_{P^*}. \quad (5.4)$$

From (5.3) and (5.4) we have

$$d\lambda_0/dt = \int (S_t P_0) \exp(-F^*)\, d\boldsymbol{x} - \sum_{j=1}^{N} (d\lambda_j/dt) <f_j>_{P^*}. \quad (5.5)$$

Inserting in (5.2) instead of $d\lambda_0/dt$, the formula (5.5) we obtain

$$<f_i\, \partial F^*/\partial t>_{P^*} = <f_i>_{P^*} \int (S_t P_0) \exp(-F^*) d\boldsymbol{x} - \sum_{j=1}^{N} (d\lambda_j/dt) <f_i>_{P^*} <f_j>_{P^*} + \sum_{j=1}^{N} (d\lambda_j/dt) <f_i f_j>_{P^*} =$$

$$\int <f_i>_{P^*} (S_t P_0) \exp(-F^*) d\boldsymbol{x} + \sum_{j=1}^{N} M_{ij} (d\lambda_j/dt),\ i = 1,\ldots,N, \quad (5.6)$$

where

$$M_{ij} = <f_i f_j>_{P^*} - <f_i>_{P^*} <f_j>_{P^*} \quad (5.7)$$

is a completely positive definite matrix (matrix of correlation of linearly independent functions $f_i = f_i(\boldsymbol{x})$, $i = 1,\ldots, N$).

Finally from (5.1) and (5.6) we obtain a system of non autonomous ordinary differential equations for $\lambda_i(t)$, $i = 1,\ldots, N$ in the followig form

$$-\sum_{j=1}^{N} M_{ij} (d\lambda_j/dt) = \int (f_i S_t (P_0 \exp(-F^*)) - (f_i - <f_i>_{P^*}) (S_t P_0) \exp(-F^*)) d\boldsymbol{x},\ i = 1,\ldots,N, \quad (5.8)$$

where

$$\lambda_0 = \ln\left(\int P_0 \exp\left(-\sum_{j=1}^{N} \lambda_i f_i\, d\boldsymbol{x}\right)\right). \quad (5.9)$$

The equation (5.9) follows from the normalisation condition $\int P^*(\boldsymbol{x},t)\, d\boldsymbol{x} = \int P_0(\boldsymbol{x},t) \exp(-F^*) d\boldsymbol{x} = 1$.

A domain $\boldsymbol{\Lambda}$ of the above equations is a set of such elements $\boldsymbol{\lambda} = (\lambda_1,\ldots,\lambda_N)$ for which all integrals in (5.8) and (5.9) exist. One can check that $\boldsymbol{\lambda} = (0,\ldots,0) = \boldsymbol{0}$ is a stationary point for the system (5.8) and in this case $P^*(\boldsymbol{x},t) = P_0(\boldsymbol{x},t)$.

The system (5.8) will be called completely stable, when its every solution $\boldsymbol{\lambda}(t) = (\lambda_1(t),\ldots,\lambda_N(t))$ may be extended for any time $t > t_0$ (t_0 is an initial time) and $\lim_{t\to\infty} \boldsymbol{\lambda}(t) = \boldsymbol{0}$.

We may formulate two important properties of solutions of the system (5.8).

Property I. *If $\boldsymbol{\lambda}(t) = (\lambda_1(t),\ldots,\lambda_N(t))$ is a solution of the system (5.8) such that $\boldsymbol{\lambda}(t_0) \in \boldsymbol{\Lambda}$ (t_0 is an initial time) and $\boldsymbol{\lambda}(t)$ may be extended into domain $\boldsymbol{\Lambda}$ for any time $t > t_0$, then $\lim_{t\to\infty} \boldsymbol{\lambda}(t) = \boldsymbol{0}$, i.e. $\boldsymbol{\lambda}(t)$ tends to the stationary solution of the system (5.8).*

Let $\boldsymbol{\lambda}(t)$ be a solution of (5.8) which may be extended into domain $\boldsymbol{\Lambda}$ for any time $t > t_0$. For such solution according to (4.8) and (3.14) we have

$$\lim_{t\to\infty} \lambda_0(t) + \sum_{i=1}^{N} \lim_{t\to\infty} \lambda_i(t) f_i(\boldsymbol{x}) = 0. \quad (5.10)$$

Because functions $f_i(\boldsymbol{x})$, $i = 1,\ldots, N$ are linearly independent (together with $f_0(\boldsymbol{x}) = 1$), from (5.10) one gets $\lim_{t\to\infty} \lambda_i(t) = 0$, for $i = 0,1,\ldots, N$. (5.11)

Property II. *In the case when $\boldsymbol{\Lambda} = \boldsymbol{E}^n$ ($\boldsymbol{E}^n$ is the n-dimensional Euclidean space), then the system (5.8) is completely stable.*

In the case when $\boldsymbol{\Lambda} \neq \boldsymbol{E}^n$, the system(5.8) is completely stable if every vector $d\boldsymbol{\lambda}/dt$, which components are given by (5.8) for every point $\boldsymbol{\lambda}$ belonging to the boundary of the set $\boldsymbol{\Lambda}$, is directed into $\boldsymbol{\Lambda}$.

The system (5.8) has the linear approximation in the following form

$$-\sum_{j=1}^{N} M_{ij}^{(0)} (d\lambda_j/dt) = \int (f_i S_t (P_0 \exp(-F^*)) - (f_i - <f_i>_{Po}) (S_t P_0) \exp(-F^*)) d\boldsymbol{x},\ i = 1,\ldots, N, \quad (5.12)$$

where

$$M_{ij}^{(0)} = <f_i f_j>_{Po} - <f_i>_{Po} <f_j>_{Po}. \quad (5.13)$$

Equations (5.12) constitute a system of linear, but in general non autonomous, ordinary differential equations.

One can notice that the function

$$V(\boldsymbol{\lambda},t) = K(P^*(\boldsymbol{x},t), P_0(\boldsymbol{x},t)) = \int P^*(\boldsymbol{x},t) \ln(P^*(\boldsymbol{x},t)/P_0(\boldsymbol{x},t)) d\boldsymbol{x} \geq 0 \quad (5.14)$$

is a Liapunov function for (5.8) [9].

Now we may formulate an approximation algorithm supported on minimizing the Kullback-Leibler information divergence in continuous systems. Approximation algorithm consists of the following steps:

Step 1. We check if a continuous system described by the system operator S_t is a Kullback system i.e. if operator S_t satisfy the inequality (2.5).

Step 2. We choose convenient set of functions $f_i = f_i(\boldsymbol{x})$, $i = 1,\ldots, N$ for our considerations.

Step 3. We calculate integrals on the right side of the accompanied evolution equations (5.8) using equation (5.9).

Step 4. We solve the accompanied evolution equations (5.8), (5.9) and obtain coefficients $\lambda_i(t)$, $i = 0,1,\ldots, N$.

Step 5. The coefficients $\lambda_i(t)$, $i = 0,1,\ldots, N$ are used in (3.13), which is the Kullback-Leibler solution.

VI. Conslusions

We have obtained the important result for the Kullback-Leibler divergence, that the time derivatives of the Kullback-Leibler divergence for approximate solutions $P^*(\boldsymbol{x},t)$ and for exact solutions $P = P(\boldsymbol{x},t)$ have the same shape. As a result, in the Kullback-Leibler systems the inequality $dK/dt \leq 0$ is satisfied for exact and approximate solutions. In consequence, the approximate solutions $P^*(\boldsymbol{x},t)$ have the same asymptotic behaviour as the exact solutions $P = P(\boldsymbol{x},t)$ i.e. they tend to the same solution P_0.

We have created an approximation algorithm supported on minimizing the Kullback-Leibler information divergence in the Kullback-Leibler systems. Practical application of the proposed approach requires knowledge of the probability density P_0. If a Kullback system possesses a stationary solution, this solution may be chosen as P_0 [4]. In the case when the Kullback system possesses a time-dependent periodic solution [6], this periodic solution is an attractor and may be chosen as P_0. Let us notice that the approximation algorithm presented in this paper not only gives a certain approximate scheme of solving the Kullback system but also generalizes the information gain minimizing approach for the Fokker-Planck equation, even with time dependent drift and diffusion coefficients[6].

Problems analogical to the ones presented in this paper were investigated by the author in the case of discrete systems and will be published in a separate paper.

Appendix

Here we will show that the system described by the Fokker-Planck Equation (F.P.E.) is a Kullback system.

We take for our consideration the n-dimensional F.P.E. with time-dependent drift and diffusion coefficients for the probability density function $P(\boldsymbol{x},t)$ of the continuous random variable $\boldsymbol{x} = (x_1, \ldots, x_n)$ in the following form[8]

$$\partial P/\partial t = -\sum_{i=1}^{n} \partial(v_i P)/\partial x_i + \sum_{i,j=1}^{n} \partial(D_{ij}(\partial P/\partial x_j))/\partial x_i , \tag{A.1}$$

were $v_i = v_i(\boldsymbol{x},t)$ is a drift vector and $D_{ij} = D_{ij}(\boldsymbol{x},t)$ is a symmetric and completely positive definite diffusion matrix[1,7]. The system operator S_t in the case of F.P.E. will be denoted as S_t^{FPE} and is defined below

$$S_t^{FPE} P = -\sum_{i=1}^{n} \partial(v_i P)/\partial x_i + \sum_{i,j=1}^{n} \partial(D_{ij}(\partial P/\partial x_j))/\partial x_i . \tag{A.2}$$

We will check that the formula (2.4) is satisfied for the system operator S_t^{FPE} i.e.

$\int((S_t^{FPE} P)\ln(P/P_0) - (S_t^{FPE} P_0)(P/P_0))d\boldsymbol{x} \leq 0$, ($=0$ only if $P = P_0$). (A.3)

Substituting (A.2) on the left side of (A.3) one obtains

$$\int(\ln(P/P_0)(-\sum_{i=1}^{n} \partial(v_i P)/\partial x_i + \sum_{i,j=1}^{n} \partial(D_{ij}(\partial P/\partial x_j))/\partial x_i) - (P/P_0)(-\sum_{i=1}^{n} \partial(v_i P_0)/\partial x_i + \sum_{i,j=1}^{n} \partial(D_{ij}(\partial P_0/\partial x_j))/\partial x_i))d\boldsymbol{x}$$

$$= -\int \sum_{i=1}^{n} \ln(P/P_0)\partial(v_i P)/\partial x_i \, d\boldsymbol{x} + \int \sum_{i,j=1}^{n} \ln(P/P_0)\partial(D_{ij}(\partial P/\partial x_j))/\partial x_i \, d\boldsymbol{x} + \int \sum_{i=1}^{n} (P/P_0)\, \partial(v_i P_0)/\partial x_i \, d\boldsymbol{x}$$

$$- \int \sum_{i,j=1}^{n} (P/P_0)\partial(D_{ij}(\partial P_0/\partial x_j))/\partial x_i \, d\boldsymbol{x} = \int \sum_{i=1}^{n} \partial(\ln(P/P_0))/\partial x_i (v_i P) d\boldsymbol{x} - \int \sum_{i,j=1}^{n} \partial(\ln(P/P_0))/\partial x_i (D_{ij}(\partial P/\partial x_j)) d\boldsymbol{x}$$

$$- \int \sum_{i=1}^{n} \partial(P/P_0)/\partial x_i (v_i P_0) d\boldsymbol{x} + \int \sum_{i,j=1}^{n} \partial(P/P_0)/\partial x_i (D_{ij}(\partial P_0/\partial x_j)) d\boldsymbol{x} = \int \sum_{i=1}^{n} (P_0/P)\partial(P/P_0)/\partial x_i (v_i P) d\boldsymbol{x}$$

$$- \int \sum_{i,j=1}^{n} \partial(\ln(P/P_0))/\partial x_i (D_{ij}(\partial((P/P_0)P_0)/\partial x_j)) d\boldsymbol{x} - \int \sum_{i=1}^{n} \partial(P/P_0)/\partial x_i (v_i P_0) d\boldsymbol{x}$$

$$+ \int \sum_{i,j=1}^{n} \partial(P/P_0)/\partial x_i (D_{ij}(\partial P_0/\partial x_j)) d\boldsymbol{x} = -\int \sum_{i,j=1}^{n} \partial(\ln(P/P_0))/\partial x_i (D_{ij}(\partial(P/P_0)/\partial x_j P_0 + (P/P_0)\partial P_0/\partial x_j) d\boldsymbol{x}$$

$$+ \int \sum_{i,j=1}^{n} \partial(P/P_0)/\partial x_i (D_{ij}(\partial P_0/\partial x_j)) d\boldsymbol{x} = -\int \sum_{i,j=1}^{n} (P_0/P)\partial(P/P_0)/\partial x_i (D_{ij}(\partial(P/P_0)/\partial x_j P_0 + (P/P_0)\partial P_0/\partial x_j) d\boldsymbol{x}$$

$$+ \int \sum_{i,j=1}^{n} \partial(P/P_0)/\partial x_i (D_{ij}(\partial P_0/\partial x_j)) d\boldsymbol{x} = -\int \sum_{i,j=1}^{n} (P_0/P)\partial(P/P_0)/\partial x_i D_{ij}(\partial(P/P_0)/\partial x_j (P_0/P) P \, d\boldsymbol{x}$$

$$- \int \sum_{i,j=1}^{n} \partial(P/P_0)/\partial x_i D_{ij} \partial P_0/\partial x_j) d\boldsymbol{x} + \int \sum_{i,j=1}^{n} \partial(P/P_0)/\partial x_i (D_{ij}(\partial P_0/\partial x_j)) d\boldsymbol{x} =$$

$$- \int \sum_{i,j=1}^{n} \partial \ln(P/P_0)/\partial x_i D_{ij} \partial \ln(P/P_0)/\partial x_j P \, d\boldsymbol{x}. \tag{A.4}$$

Let us notice, in connection with the above calculations in (A.4), that adjoint manipulation associated with spatial operations on probability density function $P(\boldsymbol{x},t)$ is possible only if $P(\boldsymbol{x},t)$ is rapidly decreasing for $|\boldsymbol{x}| \to \infty$ (the natural boundary condition according to Graham [1]).

Because D_{ij} is a completely positive definite matrix, the last formula in (A.4) satisfies the following inequality

$$- \int \sum_{i,j=1}^{n} \partial \ln(P/P_0)/\partial x_i D_{ij} \partial \ln(P/P_0)/\partial x_j P \, d\boldsymbol{x} \leq 0 , \ (=0$$

only if $P = P_0$).

(A.5)

From (A.5), (A.4) it follows that inequality (A.3) is fulfilled for the system operator S_t^{FPE}, so it is the Kullback system operator. Hence our earlier general considerations connected to the Kullback systems may be applied for the systems described by F.P.E..

References

[1] R. Graham, "Springer Tract in Modern Physics", Springer-Verlag, Berlin, vol. 66, 1973.
[2] S. Kullback, R.Leibler, "On information and sufficiency", Ann. Math. Stat., vol. 22, pp 79-86, 1951.
[3] S. Kullback, "Information theory and statistic", Wiley, New York,1959.
[4] J. Owedyk, " Investigation of stability of information gain solutions of a Fokker-Planck equation", Acta Phys. Polon., Vol. A63, no. 3, pp (317-327), 1983.
[5] J. Owedyk, "Stability of a stationary distribution of a Master Equation with respect to information gain solutions", Z. Phys. B, vol. 54, pp (183-186), 1984.
[6] J. Owedyk, "On the Fokker-Plank equation with time-dependent drift diffusion coefficients and its exponential solutions", Z. Phys. B, vol. 59, pp (69-74),1985.
[7] J. Owedyk, "On the master equation with time-dependent transition probabilities", Phys. Lett., vol 109, pp (152-154), issue 20 may 1985.
[8] H. Risken, "The Fokker-Planck equation", Springer-Verlag, Berlin, 1996.
[9] J. la Salle ,S. Lefschetz, "Stability by Liapunov's direct method", New York, London, Academic Press,1961.

Wielkoskalowe symulacje biologicznych sieci neuronowych charakteryzujących się wewnętrzną topologią wielowymiarowych torusów z wykorzystaniem PGENESIS

Large-scale simulations of biological neural networks characterized by internal topology of multi-dimensional torus using PGENESIS

Monika Kwiatkowska[1] i Łukasz Świerczewski[2]

Treść: Praca obejmuje implementacje oraz testy wydajności i skalowania biologicznych sieci neuronowych charakteryzujących się wewnętrzną topologią wielowymiarowych torusów. Do obliczeń wykorzystano równoległą wersję symulatora GENESIS - PGENESIS. Symulacje przeprowadzono w środowisku równoległym na superkomputerze (klaster wydajnościowy o architekturze x86_64) HP BladeSystem/Actina, Hydra dostępnym w Interdyscyplinarnym Centrum Modelowania Matematycznego i Komputerowego Uniwersytetu Warszawskiego. Testy objęły procesory AMD Opteron 2435, AMD Opteron 6174, AMD Opteron 6272 oraz Intel Xeon X5660. Uwzględniono także aspekt wykorzystania interfejsów sieciowych Infiniband QDR, Infiniband DDR oraz 10Gb Ethernet w komunikacji międzywęzłowej. Dodatkowo wykonano analizę uzyskanego zysku wydajności dzięki zastosowaniu wersji PGENESIS skompilowanej pod kątem wybranego procesora. W pracy skupiono się jedynie na części dotyczącej pomiarów wydajności – nie podjęto jakichkolwiek prób analiz aktywności modelowanych biologicznych sieci neuronowych.

Słowa kluczowe: PGENESIS, biologiczne sieci neuronowe, topologia torus

Abstract: This paper includes implementation and performance tests and also scaling of biological neural networks characterized by internal topology of multi-dimensional toruses. For calculations there was used a parallel version of the GENESIS - PGENESIS simulator. Simulations were performed in a supercomputer's parallel environment, (a performance cluster with x86_64 architecture) HP BladeSystem/Actina, Hydra available at the Interdisciplinary Centre for Mathematical and Computational Modeling, Warsaw University. Tests included AMD Opteron 2435, AMD Opteron 6174, AMD Opteron 6272 and Intel Xeon X5660 prosessors. There was also taken into account the aspect of the use of network interfaces such like Infiniband QDR, DDR Infiniband and 10Gb Ethernet in interstitial communication. In addition, there was performed an analysis on the resulting performance gained by using the PGENESIS version compiled for the selected processor. In this paper author focused only on the section of performance measurement - there weren't taken any attempts of activity analysis of the modeled biological neural networks.

Keywords: PGENESIS, biological neural networks, torus topology

Wprowadzenie

GENESIS (the GEneral NEural SImulation System) [1][2] jest platformą ogólnego przeznaczenia, która została opracowana w celu wspierania biologicznie realistycznych symulacji układów nerwowych, od skomplikowanych modeli pojedynczych neuronów do symulacji dużych sieci. GENESIS implementuje język wysokiego poziomu, który pozwala łatwo rozszerzyć możliwości symulatora, wymieniać, modyfikować i wykorzystywać modele lub ich elementy składowe.

Istnieje także równoległa wersja GENESIS – Parallel GENESIS (PGENESIS) [3][4]. Umożliwia ona zrównoleglenie obliczeń i uruchomienie ich na klastrach komputerowych. Wspierany jest zarówno standard PVM [5][6], jak i nowszy MPI [7]. W tej pracy wykorzystano MPI w wersji 1.4.1p1. GENESIS i PGENESIS było kompilowane kompilatorem GCC [8] 4.8.1.

Do realizacji obliczeń wykorzystano klaster komputerowy o architekturze x86_64 HP BladeSystem/Actina, Hydra zainstalowany w Interdyscyplinarnym Centrum Modelowania Matematycznego i Komputerowego Uniwersytetu Warszawskiego. W klastrze tym dostępne są cztery rodzaje węzłów. Dokładną ich specyfikację wraz z ilością zestawiono w Tab. 1.

Tab. 1. Dostępne zasoby sprzętowe na superkomputerze Hydra zainstalowanym w ICM, UW.

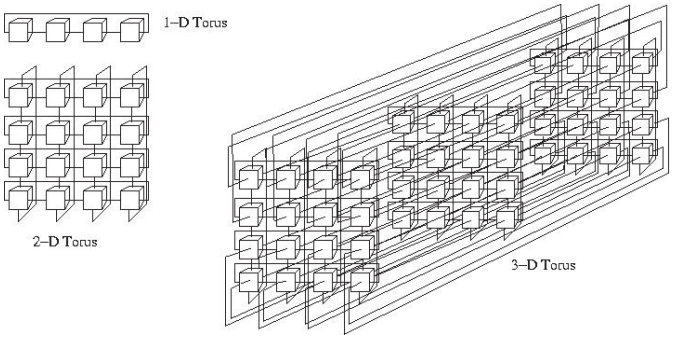
Rodzaj procesora	Taktowanie procesora	Liczba procesorów x liczba rdzeni	RAM	Typ połączenia międzywęzłowego	Liczba węzłów
AMD Opteron 2435	2,6 GHz	2 x 6	32 GB	Infiniband DDR + 1Gb Ethernet	96
Intel Xeon X5660	2,8GHz	2 x 6	24 GB	Infiniband QDR + 1Gb Ethernet	120
AMD Opteron 6174	2,2 GHz	4 x 12	256 GB	10Gb Ethernet	30
AMD Opteron 6272	2,2 GHz	4 x 16	512 GB	10Gb Ethernet	16

1. Uniwersytet Marii Curie-Skłodowskiej w Lublinie, Wydział Matematyki, Fizyki i Informatyki.
2. Państwowa Wyższa Szkołą Informatyki i Przedsiębiorczości w Łomży, Instytut Automatyki i Robotyki.

Jak przedstawiono w powyższej tabeli w systemie dostępnych jest w sumie 262 węzłów obliczeniowych. Największa ich grupa (120 węzłów) posiada po dwa sześciordzeniowe procesory Intel Xeon X5660. Połączone są one w komunikacji międzywęzłowej szybkim interfejsem Infiniband [9][10] QDR oraz 1Gb Ethernet, co daje sumaryczną teoretyczną przepustowość na poziomie 33 Gbit/s. Węzły te posiadają stosunkowo mało pamięci operacyjnej RAM – jedynie 24 GB na węzeł. Drugą pod względem liczebności (96 węzłów) grupą są jednostki wyposażone w procesory AMD Opteron 2435. Posiadają one więcej pamięci operacyjnej RAM (32 GB), jednak charakteryzują się niższą częstotliwością taktowania oraz są połączone znacznie wolniejszą siecią Infiniband typu DDR o przepustowości 16 Gbit/s. Węzłów tych także jest dość dużo – aż 96. Ostatnie dwa typy węzłów obliczeniowych stanowią rozwiązania oparte o procesory AMD Opteron 6174 oraz AMD Opteron 6272. Posiadają one bardzo dużą ilość pamięci RAM (odpowiednio 256 i 512 GB) jednak są połączone jedynie wolną siecią Ethernet o przepustowości 10 Gbit/s.

Kluczową kwestią w przypadku porównywania systemu połączeń węzłów może okazać się nie tylko maksymalna przepustowość ale także opóźnienia. W przypadku urządzeń działających w standardzie Ethernet 10 Gbit/s te mogą wynosić powyżej 40 mikrosekund dla Infiniband DDR czas ten wynosi 2 mikrosekundy, a przy Infiniband QDR spotkamy się z opóźnieniem w granicach jedynie jednej mikrosekundy.

Na potrzeby niniejszej pracy zaimplementowano sieci bazujące na topologiach wielowymiarowych torusów. Przykładowe topologie tego typu dla torusów 1D, 2D i 3D przedstawiono na Ryc. 1.



Ryc. 1. Topologie torus 1D, 2D oraz 3D. Źródło: http://wiki.expertiza.ncsu.edu/.

Jak widać na Ryc. 1. topologia torusów charakteryzuje się tym, że każdy z elementów w tej strukturze posiada odpowiednią ilość sąsiadów. Nawet węzły skrajne w tym przypadku są połączone z przeciwległymi sąsiadami.

W PGENESIS zrealizowano strukturę sieci w następujący sposób:

- na każdym węźle jest alokowana siatka o rozmiarze 32x32 neuronów (razem 1024 neuronów), w ramach węzła realizowane jest 30% połączenia pełnego (314572 połączeń),
- strukturę połączeń międzywęzłowych określa topologia torusa o określonej liczbie wymiarów, w tym przypadku od węzła do węzła realizowane jest 20% połączenia pełnego (209715 połączeń).

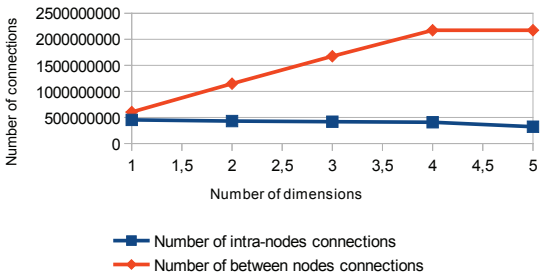
Porównanie charakterystyk największych symulacji jakie można wykonać na jednej partycji procesorów Intel Xeon X5660 na klastrze Hydra przedstawiono w Tab. 2.

Tab. 2. Największe symulacje jakie można zrealizować na jednej partycji systemu Hydra złożonej z 120 węzłów posiadających po dwa sześciordzeniowe procesory Intel Xeon X5660 (razem 1440 rdzeni).

Typ torusa	Rozmiar torusa (ilość procesów)	Całkowita ilość neuronów	Ilość połączeń wewnątrzwęzłowych / pomiędzy węzłami (suma)
1D	1440 (1440)	1474560	452984832 / 603979776 (1056964608)
2D	37x37 (1369)	1401856	430650163 / 1148400435 (1579050598)
3D	11x11x11 (1331)	1362944	418696396 / 1674785587 (2093481983)
4D	6x6x6x6 (1296)	1327104	407686348 / 2174327193 (2582013541)
5D	4x4x4x4x4 (1024)	1048576	322122547 / 2174327193 (2469606195)

Jak widać w tabeli teoretycznie największą ilość neuronów w sieci można uzyskać stosując topologię torusa 1D. Jest to spowodowane tym, że w implementacji skryptu uwzględniono możliwość realizacji jedynie torusów idealnych – tzn. takich gdzie każdy wymiar ma taki sam rozmiar. Dla torusa dwuwymiarowego tego typu największa struktura posiada 37x37 węzłów, co daje wykorzystanie maksymalnie 1369 rdzeni. Im większy wymiar tym większa utrata możliwości zaangażowania w obliczenia określonej ilości procesorów.

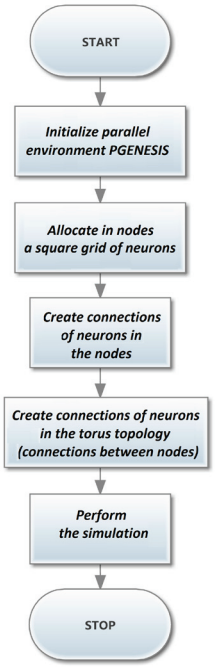
Zmianę ilości połączeń między neuronami przy ustalonej topologii określonego torusa oraz uwzględnieniu maksymalnej liczby dostępnych rdzeni na 1440 przedstawiono na Ryc. 2.



Ryc. 2. Zmiana ilości połączeń między neuronami przy ustalonej topologii określonego torusa oraz uwzględnieniu maksymalnej liczby dostępnych rdzeni na 1440.

Implementacja

Prosty schemat blokowy szablonu zaimplementowanych skryptów przedstawiono na Ryc. 3.



Ryc. 3. Prosty schemat blokowy szablonu zaimplementowanych skryptów.

Najbardziej zaawansowanym krokiem podczas tworzenia sieci neuronowej jest operacja oznaczona na schemacie (Ryc. 3) jako „*Create connections of neurons in the torus topology* (*connections beetwen nodes*)". Listingi przedstawiające realizację tej procedury w PGENESIS przedstawiono na Listingu 1 (dla torusa 2D) oraz Listingu 2 (dla torusa 3D).

```
if ({nodes} == 2)
        if ({mynode} == 1)
                pconnect_two_maps /net1 1 /net2 2 {dim} {dim} {connect_probability_beetwen_node}
                make_synapse /input /net1/cell[0]/dend/Ex_channel 10 0 0
        end
else
        for (i=1;i<nodes+1;i=i+2)
                if ({mynode} == {i})
                        if({{i+1}%{nodes}} == 0)
                                pconnect_two_maps /net{i} {i} /net{nodes} {nodes} {dim} {dim} \
                                  {connect_probability_beetwen_node}
                        else
                                pconnect_two_maps /net{i} {i} /net{{i+1}%{nodes}} {{i+1}%{nodes}} {dim} \
                                  {dim} {connect_probability_beetwen_node}
                        end
                end
        end

        if ({mynode} == 1)
                make_synapse /input /net1/cell[0]/dend/Ex_channel 10 0 0
        end
end
```

Listing 1. Struktura kod odpowiedzialna za tworzenie połączeń pomiędzy siatkami neuronów o strukturze torusa 1D

```
for (i=1;i<grid_dim+1;i=i+1)
        for (j=1;j<grid_dim+1;j=j+1)
                for (k=1;k<grid_dim+1;k=k+1)
                        if ({mynode} == {((i-1)*grid_dim*grid_dim)+((j-1)*grid_dim)+k})
                                if ( {{i+1}%{grid_dim}} == 0)
                                        pconnect_two_maps /net{i}-{j}-{k}
                                          {((i-1)*grid_dim*grid_dim)+((j-1)*grid_dim)+k} \
                                          /net{grid_dim}-{j}-{k} \
                                          {((grid_dim-1)*grid_dim*grid_dim)+((j-1)*grid_dim)+k} \
                                          {dim} {dim} {connect_probability_beetwen_node}
                                else
                                        pconnect_two_maps /net{i}-{j}-{k}
                                          {((i-1)*grid_dim*grid_dim)+((j-1)*grid_dim)+k} \
                                          /net{{i+1}%{grid_dim}}-{j}-{k} \
                                          {(({i+1}%{grid_dim}-1)*grid_dim*grid_dim)+((j-1)*grid_dim)+k} \
                                          {dim} {dim} {connect_probability_beetwen_node}
                                end
                        end
                end
        end
end
```

Listing 2. Struktura kod odpowiedzialna za tworzenie połączeń (jedynie po jednej osi) pomiędzy siatkami neuronów o strukturze torusa 3D

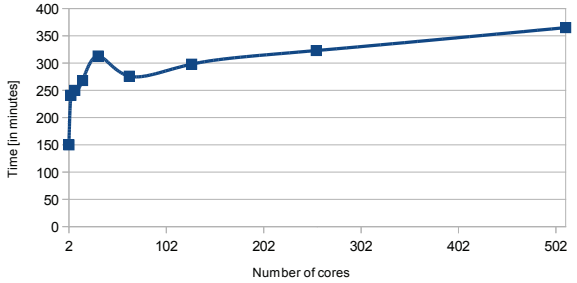
Zrealizowane symulacje

Na potrzeby pracy badawczej zrealizowano szereg symulacji – zostały one przedstawione w Tab. 3 (torus 1D), Tab. 4 (torus 2D), Tab. 5 (torus 3D), Tab. 6 (torus 4D) oraz Tab. 7 (torus 5D).

Tab. 3. Zestawienie symulacji wykonanych z wykorzystaniem topologii wewnętrznej torus 1D na czterech różnych typach węzłów.

Rozmiar torusa (ilość procesów)	Czas realizacji symulacji na AMD Opteron 6272	Czas realizacji symulacji na AMD Opteron 6174	Czas realizacji symulacji na AMD Opteron 2435	Czas realizacji symulacji na Intel Xeon X5660
2 (2)	271 min 25,199 s	386 min 43,978 s	369 min 22,822 s	150 min 21,573 s
4 (4)	322 min 3,664 s	303 min 57,682 s	375 min 19,659 s	241 min 12,390 s
8 (8)	374 min 26.470 s	592 min 5,022 s	319 min 12,117 s	250 min 46,845 s
16 (16)	486 min 8,700 s	463 min 29,277 s	352 min 49,962 s	268 min 8,244 s
32 (32)	475 min 12,651 s	599 min 14,141 s	346 min 52,060 s	313 min 45,030 s
64 (64)	526 min 33,446 s	509 min 55,929 s	342 min 39,755 s	276 min 12,626 s
128 (128)	523 min 31,221 s	577 min 54,899 s	400 min 20,326 s	298 min 19,998 s
256 (256)	?	530 min 27,452 s	?	323 min 42,148 s
512 (512)	?	530 min 7,629 s	?	365 min 5,279 s
720 (720)	?	?	?	478 min 17,941 s **ERROR NR 25 / 6**

Jak już widać w Tab. 3 dla torusa 1D wraz ze wzrostem rozmiaru symulowanej czasu wcale niekoniecznie łączy się także wzrostowy trend czasu jej symulacji. Spowodowane jest to tym, że zarówno dwa najistotniejsze parametry określające rozmiar sieci (ilość neuronów i ilość połączeń między nimi), jak i ilość realizujących symulację procesorów rośnie w sposób liniowy. Przykładowy wykres obrazujący zależność czasu od rozmiaru symulacji na torusie 1D (i zastosowanej ilości procesorów Intel Xeon X5660) przedstawiono na Ryc. 4.



Ryc. 4. Wykres obrazujący zależność czasu od rozmiaru symulacji na torusie 1D oraz zastosowanej różnej ilości procesorów Intel Xeon X5660.

Na Ryc. 4 widać zauważalne maksimum czasu realizacji w przedziale zastosowania od 2 do 128 rdzeni – dokładnie wystąpiło ono podczas symulacji określonej sieci na 32 rdzeniach. Widać, że dla większych symulacji (> 128 rdzeni) czas w niewielkim stopniu proporcjonalnie rośnie. Jak zaznaczono w Tab. 3 podczas wykonywania największych symulacji (wykorzystanych 720 rdzeni) wystąpił błąd. Objawiał się on głównie zwróceniem na ekran poniższej informacji i zakończeniem symulacji:

Parlib error 25: barrierall hung in local zone barrier (timed out)

==========

= BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES

= EXIT CODE: 6

= CLEANING UP REMAINING PROCESSES

= YOU CAN IGNORE THE BELOW CLEANUP MESSAGES

==========

APPLICATION TERMINATED WITH THE EXIT STRING: Aborted (signal 6)

Z powodu tego błędu nie wykonywano już większych symulacji. Zarzucono także próby wykorzystania m.in. całej partycji procesorów Intel Xeon X5660 (120 węzłów) co początkowo planowano zrobić. Dodatkowo w Tab. 3 w niektórych polach wstawiono znak „?" - oznacza on, że pomimo blisko miesiąca oczekiwania w systemie kolejkowym zadanie to nie zostało zrealizowane (brak wolnych zasobów sprzętowych). Poniżej zestawiono wyniki dla torusów 2-5D.

Tab. 4. Zestawienie symulacji wykonanych z wykorzystaniem topologii wewnętrznej torus 2D na węzłach posiadających procesory Intel Xeon X5660.

Rozmiar torusa (ilość procesów)	Czas realizacji symulacji
2x2 (4)	391 min 54,872 s
3x3 (9)	515 min 23,696 s
6x6 (36)	453 min 35,542 s
8x8 (64)	452 min 35,541 s
12x12 (144)	425 min 26,538 s
16x16 (256)	461 min 35,235 s
24x24 (576)	484 min 9,090 s
26x26 (676)	481 min 37,494 s

Tab. 5. Zestawienie symulacji wykonanych z wykorzystaniem topologii wewnętrznej torus 3D na węzłach posiadających procesory Intel Xeon X5660.

Rozmiar torusa (ilość procesów)	Czas realizacji symulacji
2x2x2 (8)	499 min 29,652 s
3x3x3 (27)	572 min 39,844 s
4x4x4 (64)	590 min 36,026 s
6x6x6 (216)	796 min 28,919 s **ERROR NR 25 / 6**
8x8x8 (512)	600 min 29,887 s

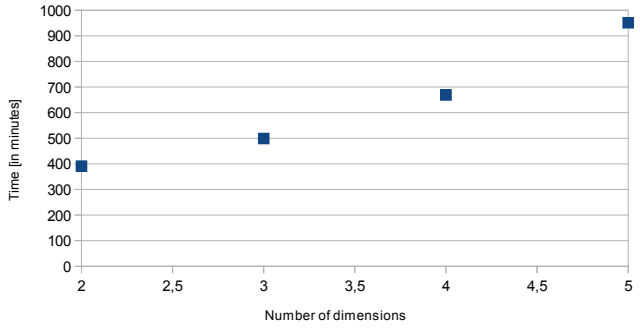
Tab. 6. Zestawienie symulacji wykonanych z wykorzystaniem topologii wewnętrznej torus 4D na węzłach posiadających procesory Intel Xeon X5660.

Rozmiar torusa (ilość procesów)	Czas realizacji symulacji
2x2x2x2 (16)	669 min 36,610 s
3x3x3x3 (81)	658 min 28,172 s
4x4x4x4 (256)	727 min 9,920 s
5x5x5x5 (625)	790 min 4,390 s

Tab. 7. Zestawienie symulacji wykonanych z wykorzystaniem topologii wewnętrznej torus 5D na węzłach posiadających procesory Intel Xeon X5660.

Rozmiar torusa (ilość procesów)	Czas realizacji symulacji
2x2x2x2x2 (32)	951 min 58,543 s
3x3x3x3x3 (243)	856 min 40,842 s

Jak widać utrzymywany jest w miarę stały czas realizacji w ramach danego torusa. Dla torusa 2D waha się on w przedziale od 391 do 515 minut, dla torusa 3D od 499 do 796 minut, dla torusa 4D od 669 do 790 minut i wreszcie dla torusa 5D ze względu na największą ilość połączeń także czas realizacji jest najwyższy i wynosi od 851 do 951 minut (dla jedynie dwóch symulacji).
Zmianę czasu realizacji symulacji w odniesieniu do ilości wymiarów torusa podczas wykonywania na Intel Xeon X5660 przedstawiono na Ryc. 5.



Ryc. 5. Czas realizacji symulacji w odniesieniu do ilości wymiarów torusa podczas wykonywania na Intel Xeon X5660

Analiza zysku uzyskanego dzięki kompilacji PGENESIS pod kątem określonego typu procesora

Podczas badań podjęto także próbę porównania wydajności uzyskanej dzięki zastosowaniu PGENESIS skompilowanego z optymalizacją pod kątem określonego typu procesora z kompilacją bez takiej optymalizacji (kompilacja generic). Czasy wykonywania przez niezoptymalizowane PGENESIS dla sieci o topologii wewnętrznej torusa 1D ujęto w Tab. 8. Ograniczono się w tym przypadku do wykonania symulacji na konfiguracjach jedno-węzłowych (tak aby pominąć dodatkowy narzut czasu spowodowany dodatkową komunikacją za pomocą znacznie wolniejszych interfejsów sieciowych).

Tab. 8. Czasy realizacji symulacji sieci neuronowych o topologii wewnętrznej torusa 1D z wykorzystaniem niezoptymalizowanej pod kątem procesora wersji PGENESIS.

Rozmiar torusa (ilość procesów)	Czas realizacji symulacji na AMD Opteron 6272	Czas realizacji symulacji na AMD Opteron 6174	Czas realizacji symulacji na AMD Opteron 2435	Czas realizacji symulacji na Intel Xeon X5660
2 (2)	272 min 53,974 s	297 min 34,104s	423 min 12,055 s	302 min 58,905 s
4 (4)	215 min 5,974 s	353 min 51,872 s	322 min 0,303 s	239 min 42,939 s
8 (8)	373 min 38,573 s	517 min 29,013 s	405 min 29,657 s	316 min 28,257 s
16 (16)	409 min 57,412 s	681 min 13,005 s	-	-
32 (32)	667 min 50,090 s	469 min 5,884 s	-	-
64 (64)	528 min 17,632 s	-	-	-

Jak widać w Tab. 8 wyniki nie są jednoznaczne – raz szybsza jest wersja zoptymalizowana pod kątem procesora, a raz jest wręcz przeciwnie. Bardziej czytelne porównanie przedstawiono w Tab. 9. W tej tabeli przykładowy zapis + 0,36% oznacza, że podczas realizacji danej symulacji wersja skompilowana z optymalizacją pod kątem danego procesora okazała się o ***0,36%*** szybsza od zastosowania bez optymalizacji. Analogicznie w przypadku gdy wartość procentową poprzedza minus mamy do czynienia ze spowolnieniem wykonywania symulacji.

Tab. 9. Procentowy zysk lub strata uzyskana dzięki zastosowaniu optymalizacji PGENESIS pod kątem procesora.

Rozmiar torusa (ilość procesów)	Czas realizacji symulacji na AMD Opteron 6272	Czas realizacji symulacji na AMD Opteron 6174	Czas realizacji symulacji na AMD Opteron 2435	Czas realizacji symulacji na Intel Xeon X5660
2 (2)	+ 0,36%	**- 29,96%**	+ 14,63%	+ 101,33%
4 (4)	**- 49,76%**	+ 16,50%	**- 16,45%**	**- 0,83%**
8 (8)	**- 0,26%**	**- 14,50%**	+ 26,95%	+ 26,40%
16 (16)	+ 18,82%	+ 47,08%	-	-
32 (32)	+ 40,42%	**- 27,71%**	-	-
64 (64)	+ 0,38%	-	-	-

W Tab. 9 możemy zaobserwować bardzo dużą rozbieżność w uzyskanych pomiarach. Według danych empirycznych uruchamiając zoptymalizowane pod kątem procesora PGENESIS w najgorszym przypadku utracimy 49,76% wydajności (AMD Opteron 6272 i rozmiar torusa równy 4) lub zyskamy 101,33% na szybkości realizacji symulacji (Intel Xeon X5660 i rozmiar torusa równy 2). Trudno jest zdecydować, które z rozwiązań daje lepsze rezultaty.
Uśredniony zysk (lub strata) wydajności (suma po kolumnach z Tab. 9) dzięki zastosowaniu optymalizacji pod kątem procesora został przedstawiony w Tab. 10.

Tab. 10. Uśredniony zysk/strata wydajności dzięki zastosowaniu optymalizacji na czterech typach węzłów.

Uśredniony zysk/starta wydajności dzięki zastosowaniu optymalizacji na:			
AMD Opteron 6272	AMD Opteron 6174	AMD Opteron 2435	Intel Xeon X5660
+ 9,96%	**- 8,59%**	+ 25,13%	+ 126,90%

Jak widać w przypadku interpretowanych danych tak jak w Tab. 10 najlepiej pod tym względem wypadają procesory Intel Xeon X5660, a najgorzej AMD Opteron 6174. Tego typu porównanie może być jednak nieco oderwane od faktycznego problemu (zbytnie uproszczenie).

Analiza spowolnienia obliczeń wynikającego z komunikacji międzywęzłowej

Teoretycznie konieczność zastosowania interfejsów sieciowych w komunikacji międzywęzłowej powinna znacznie spowolnić wykonywanie symulacji realizowanych za pomocą GENESIS. Analizie poddano prosty model oparty o dwuwymiarowy torus o rozmiarze jednego wymiaru równym 2 (2x2 - 4 główne procesy obliczeniowe). Na pojedynczym węźle standardowo alokowano siatkę 32x32 (1024) neuronów. W ramach węzła jak dotychczas realizowano 30% połączenia pełnego, natomiast zmieniano ilość połączeń przy połączeniach międzywęzłowych (od 10% do 100% w kroku co 10%). Wyniki analiz dla procesorów Intel Xeon X5660 oraz Infiniband QDR przedstawiono w Tab. 11. Rezultaty dla AMD Opteron 2435 oraz Infiniband DDR zestawiono w Tab. 12, a dane dla AMD Opteron 6174 ze wsparciem najwolniejszego w zestawieniu protokołu Ethernet 10 Gb można zobaczyć w Tab. 13. Podczas analiz pominięto węzły posiadające procesory AMD Opteron 6272 gdyż posiadają one taki sam standard komunikacji międzywęzłowej co AMD Opterony 6174.

Tab. 11. Porównanie różnych koncepcji przydziału miejsca wykonywania procesów (na jednym węźle lub z podziałem na dwa) z wykorzystaniem procesorów Intel Xeon X5660 oraz interfejsu komunikacji Infiniband QDR.

Wartość procentowa w odniesieniu do połączenia pełnego realizowanego między węzłami	Czas realizacji w przypadku zastosowania jednego węzła Architektura: 1 (węzeł) x 4 procesy	Czas realizacji w przypadku zastosowania dwóch węzłów Architektura: 2 (węzły) x 2 procesy
10%	285 min 1,974 s	303 min 45,377 s
20%	397 min 52,157 s	409 min 14,064 s
30%	489 min 32,580 s	494 min 13,924 s
40%	597 min 42,026 s	602 min 0,824 s
50%	710 min 57,143 s	702 min 44,341 s
60%	823 min 30,265 s	792 min 19,056 s
70%	919 min 49,225 s	909 min 20,841 s
80%	1010 min 34,762 s	1017 min 47,569 s
90%	1149 min 23,754 s	1111 min 44,418 s
100%	1239 min 58,651 s	1512 min 40,198 s

Tab. 12. Porównanie różnych koncepcji przydziału miejsca wykonywania procesów (na jednym węźle lub z podziałem na dwa) z wykorzystaniem procesorów AMD Opteron 2435 oraz interfejsu komunikacji Infiniband DDR.

Wartość procentowa w odniesieniu do połączenia pełnego realizowanego między węzłami	Czas realizacji w przypadku zastosowania jednego węzła Architektura: 1 (węzeł) x 4 procesy	Czas realizacji w przypadku zastosowania dwóch węzłów Architektura: 2 (węzły) x 2 procesy
10%	321 min 46,090 s	329 min 32,030 s
20%	409 min 46,183 s	521 min 24,899 s
30%	549 min 13,830 s	608 min 7,272 s
40%	739 min 56,930 s	743 min 10,726 s
50%	884 min 52,342 s	953 min 10,848 s
60%	840 min 49,289 s	962 min 16,840 s
70%	1154 min 24,513 s	1054 min 41,357 s
80%	1208 min 15,929 s	1270 min 16,345 s
90%	1301 min 9,254 s	1289 min 37,833 s
100%	1613 min 25,423 s	1583 min 44,740 s

Tab. 13. Porównanie różnych koncepcji przydziału miejsca wykonywania procesów (na jednym węźle lub z podziałem na dwa) z wykorzystaniem procesorów AMD Opteron 6174 oraz interfejsu komunikacji Ethernet 10 Gb.

Wartość procentowa w odniesieniu do połączenia pełnego realizowanego między węzłami	Czas realizacji w przypadku zastosowania jednego węzła Architektura: 1 (węzeł) x 4 procesy	Czas realizacji w przypadku zastosowania dwóch węzłów Architektura: 2 (węzły) x 2 procesy
10%	321 min 46,090 s	329 min 32,030 s
20%	409 min 46,183 s	521 min 24,899 s
30%	549 min 13,830 s	608 min 7,272 s
40%	739 min 56,930 s	743 min 10,726 s
50%	884 min 52,342 s	953 min 10,848 s
60%	840 min 49,289 s	962 min 16,840 s
70%	1154 min 24,513 s	1054 min 41,357 s
80%	1208 min 15,929 s	1270 min 16,345 s
90%	1301 min 9,254 s	1289 min 37,833 s
100%	1613 min 25,423 s	1583 min 44,740 s

Jak widać w powyższych tabelach nawet dla teoretycznie najwolniejszej konfiguracji wyposażonej w procesory AMD Opteron 6174 i Ethernet 10 Gb i nie widać zasadniczej różnicy wydajności podczas porównywania dwóch różnych rodzajów przydziałów zasobów sprzętowych. Sumaryczne zestawienie wyników z tabel (suma po kolumnach) Tab. 11, Tab. 12 oraz Tab. 13 przedstawiono w Tab. 14.

Tab. 14. Porównanie różnych koncepcji przydziału miejsca wykonywania procesów - sumaryczny czas realizacji symulacji na trzech typach węzłów i różnych rodzajach połączeń między nimi.

Sumaryczny czas realizacji symulacji					
Intel Xeon X5660 + Infiniband QDR		AMD Opteron 2435 + Infiniband DDR		AMD Opteron 6174 + Ethernet 10 Gb	
Sumaryczny czas realizacji w przypadku zastosowania jednego węzła Architektura: 1 (węzeł) x 4 procesy	Sumaryczny czas realizacji w przypadku zastosowania dwóch węzłów Architektura: 2 (węzły) x 2 procesy	Sumaryczny czas realizacji w przypadku zastosowania jednego węzła Architektura: 1 (węzeł) x 4 procesy	Sumaryczny czas realizacji w przypadku zastosowania dwóch węzłów Architektura: 2 (węzły) x 2 procesy	Sumaryczny czas realizacji w przypadku zastosowania jednego węzła Architektura: 1 (węzeł) x 4 procesy	Sumaryczny czas realizacji w przypadku zastosowania dwóch węzłów Architektura: 2 (węzły) x 2 procesy
7618 min	7851 min	9018 min	9312 min	10748 min	10301 min

W Tab. 14 widzimy, że zgodnie z oczekiwaniami dla konfiguracji posiadających szybki interfejs Infiniband sumaryczny czas realizacji w przypadku zastosowania jednego węzła jest niższy niż w przypadku zastosowania dwóch. Różnice te są jednak dla tej skali czasowej niewielkie i wynoszą ok. od 200 do 300 minut. Natomiast dla konfiguracji AMD Opteron 6174 + Ethernet 10 Gb wyniki czasowe są całkowicie odwrotne – rozwiązanie wykorzystujące jeden węzeł okazało się o ponad 400 minut wolniejsze.
Aby mieć pewność co do wniosków końcowych przeprowadzono także symulację sieci o znacznie mniejszej gęstości połączeń w ramach węzła (wykorzystano jedynie 0.003% połączenia pełnego oraz węzły połączone interfejsem Ethernet 10 Gb). Wyniki dla takiego podejścia zostały zaprezentowane w Tab. 15.

Tab. 15. Porównanie różnych koncepcji przydziału miejsca wykonywania procesów (na jednym węźle lub z podziałem na dwa) z wykorzystaniem procesorów AMD Opteron 6174 oraz interfejsu komunikacji Ethernet 10 Gb dla sieci gdzie zrealizowano 0.003% połączenia pełnego w ramach węzła.

Wartość procentowa w odniesieniu do połączenia pełnego realizowanego między węzłami	Czas realizacji w przypadku zastosowania jednego węzła Architektura: 1 (węzeł) x 4 procesy	Czas realizacji w przypadku zastosowania dwóch węzłów Architektura: 2 (węzły) x 2 procesy
10%	197 min 48,125 s	183 min 26,121 s
20%	355 min 0,894 s	366 min 48,142 s
30%	506 min 56,474 s	503 min 46,749 s
40%	614 min 54,221 s	617 min 43,608 s
50%	858 min 21,759 s	720 min 3,312 s
60%	924 min 13,016 s	792 min 44,519 s
70%	1039 min 2,005 s	914 min 55,103 s
80%	1294 min 9,567 s	983 min 38,876 s
90%	1432 min 28,770 s	1161 min 26,895 s
100%	1206 min 25,549 s	1258 min 32,250 s

Powyższe pomiary jedynie potwierdziły, że jeżeli posiadamy w miarę nowoczesny interfejs sieciowy to nie jest on jakimkolwiek ograniczeniem dla symulacji w GENESIS.

Wnioski i perspektywy dalszych badań

Początkowo w założeniach pracy był test możliwości uruchomienia symulacji charakteryzujących się topologią wielowymiarowych torusów na komputerach, które taką topologię połączeń międzywęzłowych realizują w sposób sprzętowy – Blue Gene/P [11] oraz Blue Gene/Q[12][13]. Z przyczyn czysto technicznych okazało się to jednak nie do zrealizowania. Na komputerach klasy Blue Gene występował problem z prawidłową kompilacją PGENESIS. Nie było problemów z uruchomieniem samego GENESIS – wersja sekwencyjna jednak w przypadku analiz potrzebnych do wykonania na potrzeby artykułu jest niewystarczająca. Oficjalnie do wersji 2.3.1 PGENESIS dodano plik makefile współpracujący jedynie z pierwszą generacją Blue Gene/L [14] – bez modyfikacji nie działa on jednak poprawnie na nowszych maszynach. Z powodu tych komplikacji zostaliśmy zmuszeni do wykorzystania klastra komputerowego x86_64 HP BladeSystem/Actina, Hydra. Być może w przyszłości uda się poprawnie skompilować PGENESIS na nowszych Blue Gene dzięki czemu możliwy będzie powrót do tematu.
Podczas badań okazało się, że rozmiar symulowanej sieci w przypadku torusa o określonej liczbie wymiarów nie wpływa w zauważalny sposób na czas symulacji. Dla przykładu symulacja sieci o topologii torusa 2D i wymiarze 2x2 wykonywała się ok. 391 minut. Dla wymiaru 24x24 operacja symulacji zajęła już ok. 484 minut, ale dla 26x26 zaledwie ok. 481 minut.
W trakcie formowania założeń do artykułu zakładano, że PGENESIS będzie działało prawidłowo na dowolnej liczbie procesorów – przynajmniej do rozmiarów jednej partycji złożonej ze 120 węzłów posiadających po dwa procesory Intel Xeon X5660, co daje łącznie 1440 rdzeni obliczeniowych. W praktyce okazało się jednak, że już przy wykorzystaniu 60 węzłów tego typu (720 rdzeni obliczeniowych) symulacja kończy się błędem synchronizacji procesów i przedwczesnego zakończenia przynajmniej jednego z nich. Największą symulację prawidłowo udało się zakończyć na 676 rdzeniach (57 węzłów). Nie zdiagnozowano co dokładnie było problemem – samo GENESIS, środowisko programowe na klastrze Hydra czy też sprzęt. Nie wykluczone, że na innym klastrze komputerowym symulacje o większych rozmiarach przebiegłyby prawidłowo. Autorom nie udało się odnaleźć informacji o symulacjach cechujących się rozmiarem sieci ponad 1,5 miliarda neuronów (i kilka bilionów połączeń między nimi) wykonanych za pomocą symulatora PGENESIS.
Najciekawszym efektem pracy jest jednak uzasadnienie, że PGENESIS nie limituje się na możliwościach sieci łączącej węzły (operacje I/O), a możliwościach obliczeniowych współczesnych procesorów. W badaniach uwzględniono jedynie interfejsy Infiniband QDR/DDR oraz Ethernet 10 Gb. Nie wiadomo jak sprawa wyglądałaby podczas zastosowania znacznie wolniejszych standardów Ethernet 1 Gb, które nadal często stosowane w tani klastrach typu Beowulf [15].
Warto także uściślić ułomną naturę pomiarów, które wy-

konano. W przypadku gdy alokowano całe węzły to oczywiście symulacja wykorzystywała w pełni ich możliwości obliczeniowe. Jednak gdy rezerwowano jedynie część węzła/węzłów to na pozostałej części mogły (ale nie musiały) wykonywać się zadania innych użytkowników. Oczywiście w pewnym stopniu wpłynęły one na czas przeprowadzonych symulacji. W przypadku gdy na węźle rezydowałyby tylko nieliczne procesy to przy procesorach Intel Xeon X5660 istniałaby dodatkowo możliwość automatycznego wykorzystania technologii Turbo Boost [16], która zwiększa częstotliwość taktowania wykorzystywanych rdzeni.

Podziękowania

Obliczenia wykonano w Interdyscyplinarnym Centrum Modelowania Matematycznego i Komputerowego (ICM) Uniwersytetu Warszawskiego w ramach grantu obliczeniowego nr G55-11.

Literatura

1. Bower, James M., David Beeman. The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System. Electronic Library of Science, The, 1995.
2. Bower, James M., David Beeman. "Constructing realistic neural simulations with GENESIS." Neuroinformatics. Humana Press, (2007), 103-125.
3. Goddard, Nigel H., Greg Hood. "Large-scale simulation using parallel GENESIS." The Book of GENESIS. Springer New York, (1998), 349-379.
4. Hood, Greg. "Using P-GENESIS for Parallel Simulation of GENESIS Models." Brains, Minds and Media 1.2 2005.
5. Geist, Al, ed. PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing. MIT press, 1994.
6. Sunderam, Vaidy S. "PVM: A framework for parallel distributed computing." Concurrency: practice and experience 2.4 (1990), 315-339.
7. Gropp, William, Ewing Lusk, and Anthony Skjellum. Using MPI: portable parallel programming with the message-passing interface. Vol. 1. MIT press, 1999.
8. de Hoon, Michiel JL, et al. "Open source clustering software." Bioinformatics 20.9 (2004), 1453-1454.
9. Pfister, Gregory F. "An introduction to the InfiniBand architecture." High Performance Mass Storage and Parallel I/O 42 (2001), 617-632.
10. Liu, Jiuxing, Jiesheng Wu, and Dhabaleswar K. Panda. "High performance RDMA-based MPI implementation over InfiniBand." International Journal of Parallel Programming 32.3 (2004), 167-198.
11. Almasi, Gheorghe, et al. "Overview of the IBM Blue Gene/P project." IBM Journal of Research and Development 52.1-2 (2008), 199-220.
12. Haring, Ruud A., et al. "The ibm blue gene/q compute chip." Micro, IEEE 32.2 (2012), 48-60.
13. Chen, Dong, et al. "The IBM Blue Gene/Q interconnection network and message unit." High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for. IEEE, 2011.
14. Gara, Alan, et al. "Overview of the Blue Gene/L system architecture." IBM Journal of Research and Development 49.2.3 (2005), 195-212.
15. Becker, Donald J., et al. "BEOWULF: A parallel workstation for scientific computation." Proceedings, International Conference on Parallel Processing. Vol. 95. 1995.
16. Charles, James, et al. "Evaluation of the Intel® Core™ i7 Turbo Boost feature." Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on. IEEE, 2009.

Acknowledgment

Interdisciplinary Centre for Mathematical and Computational Modeling (ICM), Warsaw University, Poland is acknowledged for providing the computer facilities under the Grant No. G55-11.

Testowanie przypuszczenia Beal'a z wykorzystaniem klasycznych procesorów

Testing Beal conjecture using classical processors

Monika Kwiatkowska[1] i Łukasz Świerczewski[2]

Treść: Praca obejmuje testowanie przypuszczenia Beal'a z wykorzystaniem klasycznych procesorów. Dodatkowo w wybranych funkcjach oprogramowania wykorzystano standard OpenMP, co umożliwiło zrównoleglenie obliczeń. Do obliczeń wykorzystano jednostki obliczeniowe wchodzące w skład komputerów IBM Blue Gene/P, IBM Blue Gene/Q oraz IBM Power 775. Testy wykonano także na superkomputerze HP BladeSystem/Actina, Hydra dostępnym w ICM UW - użyto tam węzła obliczeniowego posiadającego dwa procesory Intel Xeon X5660. Porównano wydajność własnych rozwiązań napisanych w języku C z możliwościami oprogramowania napisanego w języku Python przez Peter'a Novig'a.

Słowa kluczowe: przypuszczenie Beal'a, Blue Gene/P, Blue Gene/Q, Power 775

Abstarct: This paper includes the testing of Beal's conjecture using classical processors. Additionally some features of OpenMP standard were used in software what allowed to parallel the calculation. Calculations were based on computational units included in the computers IBM Blue Gene/L, IBM Blue Gene/Q, and the IBM Power 775 tests have been performed on the supercomputer HP BladeSystem / Actina, Hydra available in the ICM UW - computing nodes with Intel Xeon processors X5660 were used there. The performance of own solutions written in C was compared with the capabilities of software written in Python by Peter Novig.

Keywords: Beal conjecture, Blue Gene/P, Blue Gene/Q, Power 775

Acknowledgment

Interdisciplinary Centre for Mathematical and Computational Modeling (ICM), Warsaw University, Poland is acknowledged for providing the computer facilities under the Grant No. G55-11.

Wprowadzenie

Przypuszczenie Beal'a jest nieudowodnionym twierdzeniem matematycznym z teorii liczb. Mówi ono, że jeśli:

$x^m + y^n = z^r$

gdzie x, y, z, m, n oraz r są dodatnimi liczbami całkowitymi, oraz m, n, $r > 2$, to x, y, z mają wspólny dzielnik będący liczbą pierwszą. Z powyższego wynika, że nie znajdziemy rozwiązania powyższego równania dla wartości x, y, z, które są parami względnie pierwsze.
Przypuszczenie zostało sformułowane w roku 1993 przez Andrew Beal'a podczas jego prac nad uogólnieniami twierdzenia Fermata. Ufundował on w roku 1997 nagrodę w wysokości $5000 za dostarczenie dowodu lub kontrprzykładu dla swojej teorii. Na przestrzeni lat nagroda była kilkukrotnie podnoszona i w tej chwili (rok 2015) wynosi $1000000.

Rozwiązanie Peter'a Norvig'a

Peter Norvig na swojej stronie [4] zaprezentował rozwiązanie w języku Python analizujące przypuszczenie Beal'a. Jeden z dwóch kodów źródłowych (ten bardziej zoptymalizowany) zaprezentowano na Listingu 1.

```
1  def beal(max_base, max_power):
2      bases, powers, table, pow = initial_data(max_base, max_power)
3      for x in bases:
4          powx = pow[x]
5          for y in bases:
6              if y > x or gcd(x,y) > 1: continue
7              powy = pow[y]
8              for m in powers:
9                  xm = powx[m]
10                 for n in powers:
11                     sum = xm + powy[n]
12                     r = table.get(sum)
13                     if r: report(x, m, y, n, nth_root(sum, r), r)
```

Listing. 1. Rozwiązanie w języku Python zaproponowane przez Peter'a Norvig'a.
Listing 1. The solution in Python programming language proposed by Peter Norvig.

W Tab. 1. zaprezentowano główne wyniki jakie uzyskał Peter Norvig [4] realizując kod przedstawiony na Listingu 1. Podczas obliczeń wykorzystano interpreter języka Python w wersji 1.5 oraz procesor o częstotliwości taktowania 400 MHz.

1. Uniwersytet Marii Curie-Skłodowskiej w Lublinie, Wydział Matematyki, Fizyki i Informatyki
2. Państwowa Wyższa Szkołą Informatyki i Przedsiębiorczości w Łomży, Instytut Automatyki i Robotyki

Tab. 1. Czasy realizacji rozwiązania Peter'a Norvig'a przedstawione w [4]. Czasy mierzone w godzinach.
Tab. 1. Execution times of Peter Norvig solution presented in [4]. Times measured in hours.

	max_ power=7	max_ power=10	max_ power=30	max_ power=100	max_ power=1000
max_base=100	-	-	-	0,6	19,0
max_base = 1000	-	-	-	6,2	?
max_ base=10000	-	-	52,0	993,0	?
max_ base=100000	-	109,0	?	?	?
max_ base=250000	1323,0	?	?	?	?

Własne wyniki czasowe dla tego samego kodu realizowanego jednak na procesorze Intel Xeon X5660 z wykorzystaniem interpretera Python w wersji 2.6.6 zaprezentowano w Tab. 2.

Tab. 2. Czasy realizacji rozwiązania Peter'a Norvig'a z wykorzystaniem procesora Intel Xeon X5660 i Pythona w wersji 2.6.6. Czasy mierzone w godzinach.
Tab. 2. Execution times of Peter Norvig solution with the use of Intel Xeon X5600 processor and 2.6.6. version of Python. Times measured in hours.

	max_ power=7	max_ power=10	max_ power=30	max_ power=100	max_ power=1000
max_ base=100	-	-	-	0,003	0,527
max_ base = 1000	-	-	-	0,968	?
max_ base=10000	-	-	42,257	> 168,000	?
max_ base=100000	-	> 168,00	?	?	?
max_ base=250000	> 168,00	?	?	?	?

Jak widać uzyskane przyśpieszenie jest w dużym stopniu zależne od parametrów z jakimi był uruchamiany program. Na procesorze 400 MHz i Pythonie 1.5 dla parametrów max_power=100 oraz max_base=100 czas wykonywania wyniósł 0.6 godziny. Po zastosowaniu nowszej wersji Pythona i procesora Intel Xeon X5660 czas ten spadł do 12.86 sekundy (0.003 godziny). Daje to więc przyśpieszenie równe aż równo 200 razy. Jeżeli jednak uwzględnimy większy przedział dla podstaw (max_base=1000) to wartość przyśpieszenia spadnie do 6.404.

Wersja pierwsza własnego rozwiązania

Zaimplementowano funkcję o poniższym nagłówku:

```
unsigned int beal_conjecture_test(
        unsigned long long int min_base,
        unsigned long long int max_base,
        unsigned long long int min_power,
        unsigned long long int max_power,
        beal_test_result_t** results);
```

Gdzie *beal_test_result_t* jest strukturą, w której będą ewentualnie przechowywane odnalezione kontrprzykłady. Struktura ta zdefiniowana jest następująco:

```
typedef struct beal_test_result {
        unsigned long long int x;
        unsigned long long int y;
        unsigned long long int z;
        unsigned long long int m;
        unsigned long long int n;
        unsigned long long int r;
} beal_test_result_t;
```

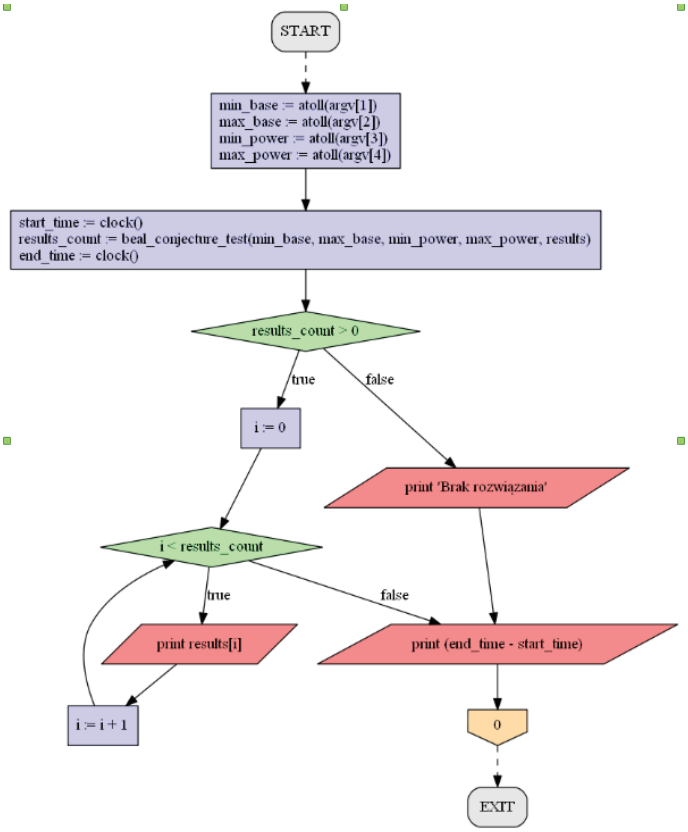
Grupuje ona wartości wszystkich podstaw i potęg równania $x^m + y^n = z^r$.

Funkcja *beal_conjecture_test* wykonuje zasadniczą część analizy Przypuszczenia Beala. Dla podanych przedziałów wartości podstaw (*min_base, max_base*) i potęg *(min_power, max_power*) generowane są wszystkie możliwe kombinacje wartości x, y, z, m, n, r z równania $x^m + y^n = z^r$. Jeśli po podstawieniu (z uwzględnieniem, że x, y, z muszą być parami względnie pierwsze) wartości równanie jest spełnione, oznacza to, że został znaleziony kontrprzykład. Wartości wszystkich zmiennych równania zapisywane są wtedy w strukturze *beal_test_result_t* i umieszczane w tablicy takich struktur. Po zapisaniu danych znalezionego rozwiązania, funkcja kontynuuje poszukiwania kolejnych rozwiązań. Po zbadaniu wszystkich kombinacji, tablica struktur jest zwracana przez jeden z parametrów funkcji, a wartość będąca wynikiem funkcji określa ilość znalezionych kontrprzykładów.
W pierwszej kolejności rozpatrywane są wszystkie kombinacje wartości podstaw x i y (dwie najbardziej zewnętrzne pętle *for*). Dla tych par, które są względnie pierwsze ($gcd(x,y)==1$), sprawdzane są następnie wszystkie możliwe wartości podstawy z. Jeśli z jest względnie pierwsze zarówno z x jak i z y, dla trójki x, y, z, sprawdzane są wszystkie możliwe kombinacje potęg z zadanego przedziału (trzy najbardziej wewnętrzne pętle *for*).
Program operuje na 64-bitowych dodatnich liczbach całkowitych. Wszelkie obliczenia są wykonywane modulo 2^{64}, co stwarza ryzyko fałszywych trafień. Takie przypadki powinny być wyeliminowane już ręcznie (poza programem).

Tab. 3. Czasy realizacji pierwszego rozwiązania napisanego w języku ANSI C na procesorze Intel Xeon X5660 (kompilacja kompilatorem GCC 4.4.7 z optymalizacją trzeciego stopnia O3). Min_base = 2, Min_ power = 3. Czasy mierzone w godzinach.
Tab. 3. Execution times of the first solution written in ANSI C programming language on Intel Xeon X5660 processor (compilation with GCC 4.4.7 compiler, with 3rd level optimisation O3). Min_base = 2, Min_power = 3. Times measured in hours.

	max_ power=7	max_ power=10	max_ power=30	max_ power=100	max_ power=1000
max_base=100	0,000	0,000	0,000	0,032	38,223
max_base = 1000	0,016	0,030	0,654	34,780	> 168,000
max_ base=10000	19,356	33,189	> 168,000		
max_ base=100000	> 168,000				
max_ base=250000	> 168,000				

Podstawowy schemat blokowy przedstawiający działanie programu przedstawiono na Ryc. 1.



Ryc. 1. Uproszczony schemat blokowy przedstawiający działanie programu.

Fig. 1. Simplified block diagram showing how the programme works.

Wersja druga własnego rozwiązania

W celu przyspieszenia wykonywania obliczeń, zoptymalizowana funkcja wykorzystuje dodatkową pamięć (tablicę), w której zapisywane są wszystkie możliwe wyniki potęgowania dla podstaw z przedziału [*min_base*, *max_base*] i potęg z przedziału [*min_power*, *max_power*]. Pierwszy indeks utworzonej tablicy odpowiada wartościom podstawym, spośród tych analizowanych, a drugi indeks – wartościom wykładnika. Dla przykładu, w komórce o indeksach [0][0], znajdzie się wartość $min\_base^{min\_power}$, w komórce o indeksach [0][1] – $min\_base^{(min\_power+1)}$, [1][0] – $(min\_base+1)^{min\_power}$, itd. Dzięki takiemu podejściu, rozmiar wymaganej tablicy zależy jedynie od szerokości przedziałów wartości podstaw i potęg.

Tab. 4. Czasy realizacji drugiego rozwiązania napisanego w języku ANSI C na procesorze Intel Xeon X5660 (kompilacja kompilatorem GCC 4.4.7 z optymalizacją trzeciego stopnia O3). Czasy mierzone w godzinach.

Tab. 4. Execution times of the second solution written in ANSI C programming language on Intel Xeon X5660 processor (compilation with GCC 4.4.7 compiler, with third level optimisation O3). Times measured in hours.

	max_ power=7	max_ power=10	max_ power=30	max_ power=100	max_ power=1000
max_ base=100	0,000	0,000	0,000	0,026	25,798
max_ base = 1000	0,016	0,030	0,662	27,136	> 168,000
max_ base=10000	19,851	33,693	> 168,000		
max_ base=100000	> 168,000				
max_ base=250000	> 168,000				

Wersja trzecia własnego rozwiązania

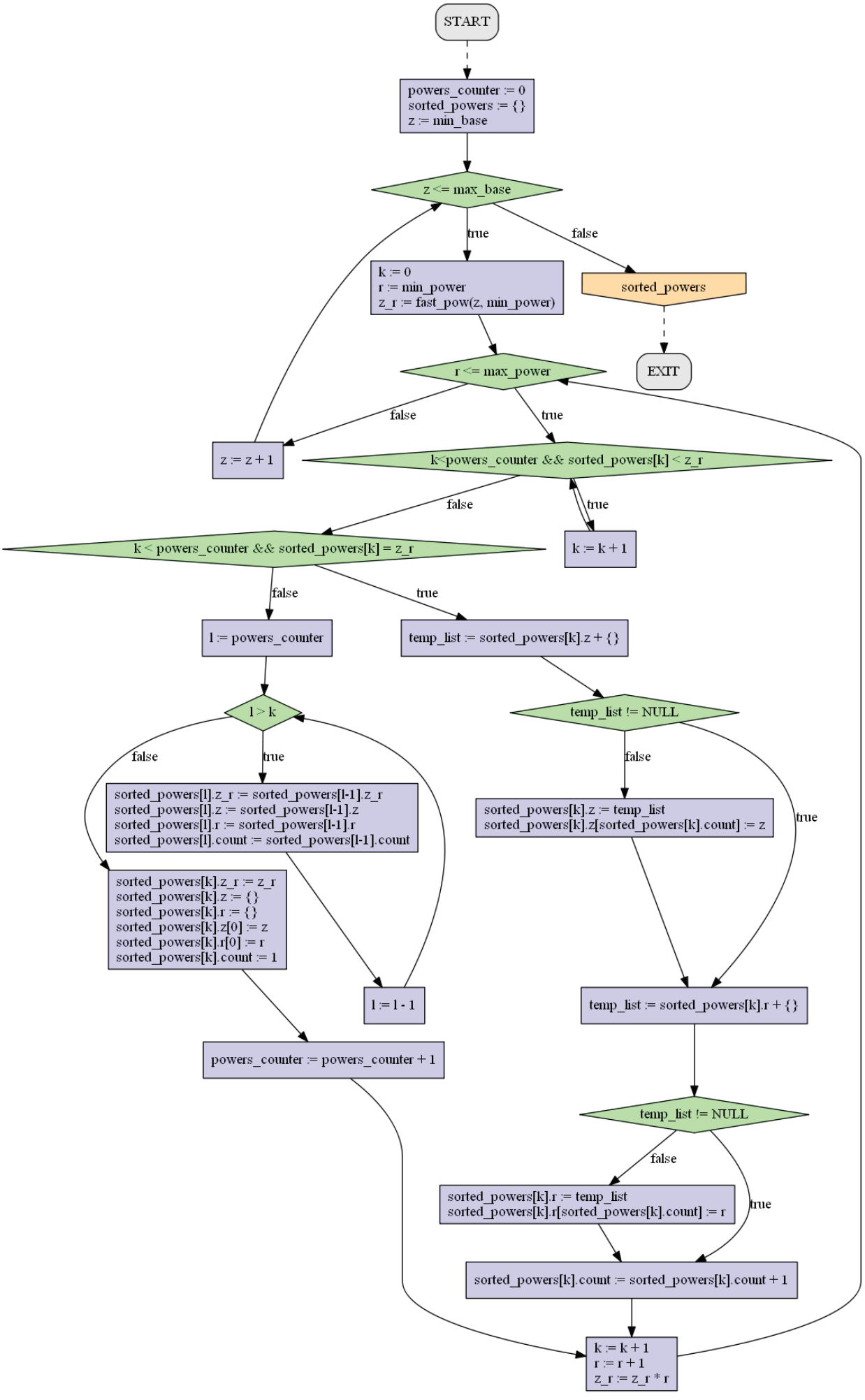
W trzecim algorytmie zastosowanie znajduje również druga tablica (*sorted_powers*), przechowująca struktury *powers_list_t*. Dla każdego możliwego wyniku potęgowania (końcowej wartości), gdzie podstawa jest z przedziału [*min_base, max_base*], a potęga z przedziału [*min_power, max_power*], w tablicy znajduje się dokładnie jeden rekord. Oprócz wyniku potęgowania, przechowywane są również podstawa i wykładnik (potęga). W przypadku, gdy więcej niż jedna kombinacja podstawy i wykładnika daje taki sam wynik (np. $10^4 = 100^2$), w strukturze na danej pozycji zapisane będą wszystkie wartości podstaw i wykładników dających taki wynik.

W tablicy *sorted_powers* będziemy szukać wyników potęgowania po wartości. Aby można było robić to efektywnie, tablica musi być posortowana. Tablica jest sortowana „w locie" podczas tworzenia. Wykorzystujemy fakt, że kolejne potęgi zadanej podstawy tworzą ciąg rosnący, dzięki czemu dodawanie elementów z zachowaniem sortowania jest szybsze, niż gdybyśmy sortowali tablicę po wypełnieniu.

```
for (xi=0; xi<=max_base_index; ++xi)
{
    x = min_base + xi;

    for (yi=xi+1; yi<=max_base_index; ++yi)
    {
        y = min_base + yi;

        if (gcd(x, y) == 1ULL)
        {
            for (mi=0; mi<=max_power_index; ++mi)
            {
                x_m = powers[xi][mi];

                for (ni=0; ni<=max_power_index; ++ni)
                {
                    y_n = powers[yi][ni];
                    i = binary_search(sorted_powers, 0, powers_counter-1, x_m + y_n);

                    if (i != -1)
                    {
                        for (j=0; j<sorted_powers[i].count; ++j)
                        {
                            z = sorted_powers[i].z[j];

                            if (z != x && z != y && gcd(z, x) == 1ULL && gcd(z, y) == 1ULL)
                            {
                                ++results_count;

                                temp_results = (beal_test_result_t*)
                                    realloc(*results, results_count * sizeof(beal_test_result_t));

                                if (temp_results)
                                {
                                    *results = temp_results;
                                    (*results)[results_count-1].x = x;
                                    (*results)[results_count-1].y = y;
                                    (*results)[results_count-1].z = z;
                                    (*results)[results_count-1].m = min_power+mi;
                                    (*results)[results_count-1].n = min_power+ni;
                                    (*results)[results_count-1].r = sorted_powers[i].r[j];
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Listing 2. Część główna kodu maksymalnie zoptymalizowanej funkcji beal_conjecture_test odpowiedzialna za poszukiwanie kontrprzykładu. Źródło: Opracowanie własne.

Listing 2. The part of maximally optimised function's code beal_conjecture_test responsible for creating the sorted_powers table.



build_sorted_powers(min_base, max_base, min_power, max_power, <out>powers_counter)

Ryc. 2. Schemat blokowy procedury build_sorted_powers.
Fig. 2. The block diagram of build_sorted_powers procedure.

```
for (xi=0; xi<=max_base_index; ++xi) {
    x = min_base + xi;
    for (yi=xi+1; yi<=max_base_index; ++yi) {
        y = min_base + yi;
        if (gcd(x, y) == 1ULL) {
            for (mi=0; mi<=max_power_index; ++mi) {
                x_m = powers[xi][mi];
                for (ni=0; ni<=max_power_index; ++ni) {
                    y_n = powers[yi][ni];
                    i = binary_search(sorted_powers, 0, powers_counter-1, x_m + y_n);
                    if (i != -1) {
                        for (j=0; j<sorted_powers[i].count; ++j) {
                            z = sorted_powers[i].z[j];
                            if (z != x && z != y && gcd(z, x) == 1ULL && gcd(z, y) == 1ULL) {
                                ++results_count;
                                temp_results = (beal_test_result_t*)
                                    realloc(*results, results_count * sizeof(beal_test_result_t));
                                if (temp_results) {
                                    *results = temp_results;
                                    (*results)[results_count-1].x = x;
                                    (*results)[results_count-1].y = y;
                                    (*results)[results_count-1].z = z;
                                    (*results)[results_count-1].m = min_power+mi;
                                    (*results)[results_count-1].n = min_power+ni;
                                    (*results)[results_count-1].r = sorted_powers[i].r[j];
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Listing 3. Część główna kodu maksymalnie zoptymalizowanej funkcji beal_conjecture_test odpowiedzialna za poszukiwanie kontrprzykładu.
Listing 3. The main part of maximally optimised function's code beal_conjecture_test responsible for counterexample searching.

Tab. 5. Czasy realizacji trzeciego rozwiązania napisanego w języku ANSI C na procesorze Intel Xeon X5660 (kompilacja kompilatorem GCC 4.4.7 z optymalizacją trzeciego stopnia O3). Czasy mierzone w godzinach.
Tab. 5. Execution times of the third solution written in ANSI C programming language on Intel Xeon X5660 processor (compilation with GCC 4.4.7 compiler, with third level optimisation O3). Times measured in hours.

	max_ power=7	max_ power=10	max_ power=30	max_ power=100	max_ power=1000
max_ base=100	0,000	0,000	0,000	0,000	0,050
max_ base = 1000	0,000	0,000	0,003	0,042	5,921
max_ base=10000	0,018	0,046	0,561	5,780	> 168,000
max_ base=100000	1,906	5,348	79,925	> 168,000	
max_ base=250000	17,543	43,025	> 168,000		

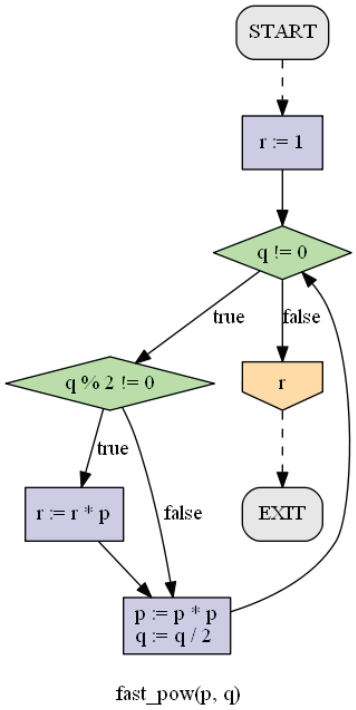
Wykorzystane algorytmy elementarne

W funkcji głównej *beal_conjecture_test* byliśmy zmuszeni do zastosowania kilku elementarnych algorytmów. W tym miejscu warto jest poświęcić chwilę na ich analizę.

Szybkie potęgowanie

```
unsigned long long int fast_pow(unsigned long long int p, unsigned long long int q):
```

Funkcja implementuje algorytm szybkiego obliczania potęgi o wykładniku naturalnym. Potęgowanie odbywa się poprzez serię operacji mnożenia. Dzięki wykorzystaniu właściwości binarnej reprezentacji wykładnika, liczba potrzebnych operacji mnożenia jest rzędu $log_2 q$ (metoda naiwna potrzebuje q operacji mnożenia).
Wykorzystujemy fakt, że podnoszenie do potęgi, która sama jest potęgą dwójki, można wykonać przy pomocy serii operacji podniesienia do kwadratu (podnosimy do kwadratu wynik poprzedniej operacji). Jedynki w binarnej reprezentacji liczby naturalnej określają, jakie potęgi dwójki wchodzą w jej skład. Każde potęgowanie możemy rozbić na iloczyn potęg, z których każda ma wykładnik będący potęgą dwójki.
Schemat blokowy algorytmu szybkiego potęgowania został przedstawiony na Ryc. 3.



Ryc. 3. Schemat blokowy algorytmu szybkiego potęgowania.
Fig. 3. The block diagram of fast raising to a power algorithm.

Największy wspólny dzielnik

```
unsigned long long int gcd(unsigned long long int a, unsigned long long int b);
```

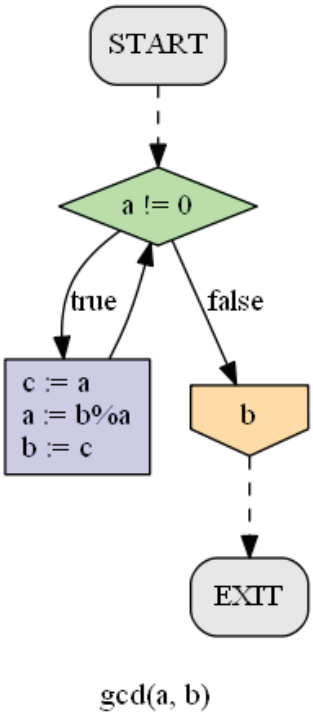
Funkcja oblicza największy wspólnego dzielnik (*nwd*) dwóch liczb naturalnych, wykorzystując algorytm Euklidesa. Algorytm oparty jest na następujących zależnościach:

$nwd(b, 0) = b$,
$nwd(b, a) = nwd(a, b \bmod a)$.

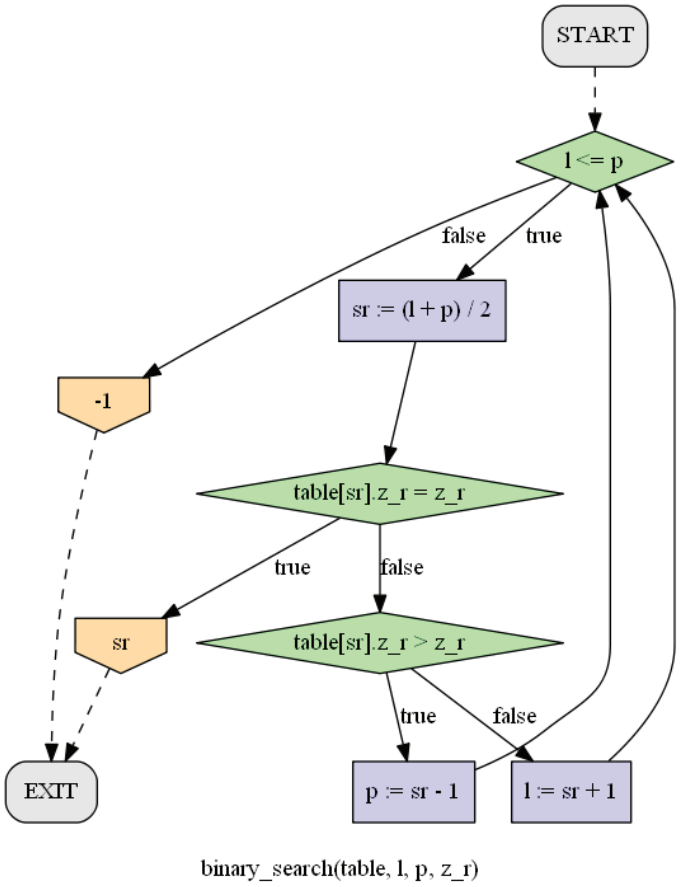
W naszej implementacji, wartość zmiennych a i b w kolejnych iteracjach wyliczane są jako:

$a_{t+1} = b_t \bmod a_t$,
$b_{t+1} = a_t$.

Zatrzymujemy się w momencie, gdy nowa wartość a jest równa 0. Wynikiem, czyli największym wspólnym dzielnikiem, jest wartość b.
Schemat blokowy algorytmu NWD został przedstawiony na Ryc. 4.



Ryc. 4. Schemat blokowy algorytmu NWD.
Fig. 4. The block diagram of NWD algorithm.



Ryc. 5. Schemat blokowy algorytmu przeszukiwania tablicy metodą podziałów.
Fig. 5. The block diagram of table searching with divisions method.

Przeszukiwanie tablicy metodą podziałów

```
int binary_search(powers_list_t* table, int l, int
p, unsigned long long int z_r);
```

Funkcja sprawdza, czy podany element znajduje się w przekazanej tablicy. Tablica musi być posortowana, aby algorytm działał poprawnie. Na początku sprawdzany jest środkowy element tablicy. Jeśli jest taki sam jak szukany, algorytm się kończy. Jeśli środkowy element jest większy od szukanego, powtarzamy procedurę dla lewej połówki tablicy. Jeśli jest mniejszy – powtarzamy procedurę dla prawej połówki tablicy.
Jeśli uda się znaleźć szukany element, funkcja zwraca jego indeks w tablicy. Jeśli elementu nie ma, funkcja zwraca wartość -1.
Schemat blokowy algorytmu przeszukiwania tablicy metodą podziałów został przedstawiony na Ryc. 5.

Zrównoleglenie kodu w OpenMP

Program można bardzo prosto zrównoleglić z wykorzystaniem środowiska OpenMP [5] [6]. Do trzeciej wersji kodu wystarczy dodać dwie pragmy – jedną odpowiedzialną za zrównoleglenie najbardziej zewnętrznej pętli for i drugiej odpowiedzialnej za wyodrębnienie sekcji krytycznej.

Pierwsza dyrektywa *#pragma* powinna wyglądać następująco:

#pragma omp parallel for private(*xi, x, yi, y, mi, x_m, ni, y_n, i, j, z*) *num_threads*(*number_of_threads*)

Druga dyrektywa *#pragma* to zwykłe ujęcie bloku instrukcji warunkowej:

if (*z != x && z != y && gcd*(*z, x*) == *1ULL && gcd*(*z, y*) == *1ULL*)

w

#pragma omp critical

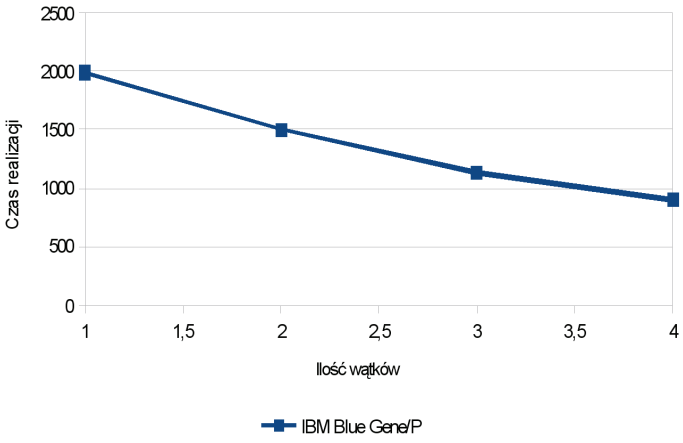
Wyniki czasowe zrównoleglenia uzyskane na czterech różnych platformach sprzętowych (IBM Blue Gene/P, IBM

Blue Gene/Q, IBM Power 775 oraz 2x Intel Xeon X5660) przedstawiono w Tab. 6.

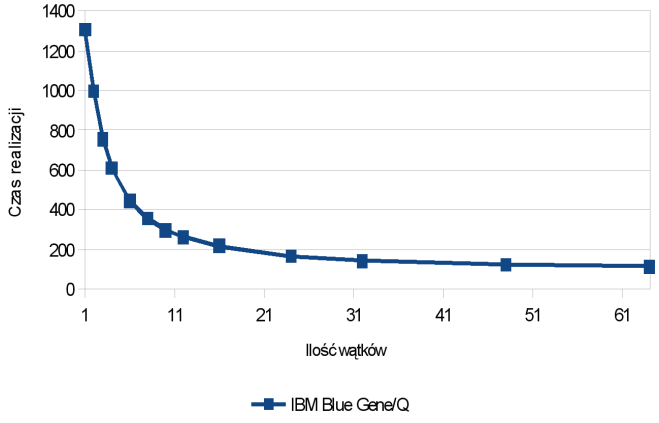
Tab. 6. Czasy realizacji rozwiązania zrównoleglonego w środowisku OpenMP wykonywanego z parametrami max_base = 500, max_power = 500. Czasy mierzone w sekundach.
Tab. 6. Execution times of parallel solution in OpenMP environment with parameters: max_base=500, max power = 500. Times measured in seconds.

Ilość wątków	IBM Blue Gene/P	IBM Blue Gene/Q	IBM Power 775	2x Intel Xeon X5660
1	1991	1305	1734	1612
2	1505	996	1309	1229
3	1133	755	986	924
4	901	609	784	729
6	-	445	562	533
8	-	356	443	421
10	-	298	364	349
12	-	262	314	302
16	-	218	253	-
24	-	166	181	-
32	-	143	147	-
48	-	125	-	-
64	-	115	-	-

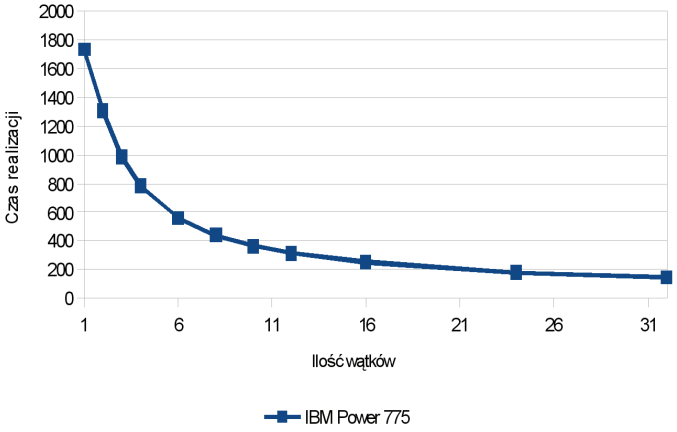
Wykresy przedstawiające spadek czasu realizacji programu dzięki zrównolegleniu dla badanych platform sprzętowych zaprezentowano na Ryc. 6 (IBM Blue Gene/P), Ryc. 7 (IBM Blue Gene/Q), Ryc. 8 (IBM Power 775) oraz Ryc. 9 (2x Intel Xeon X5660).



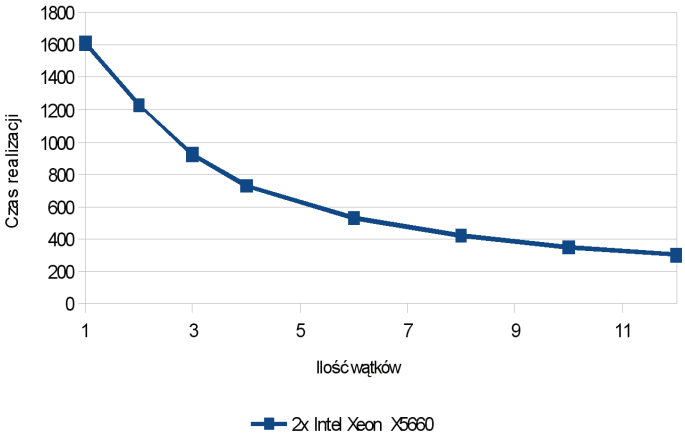
Ryc. 6. Redukcja czasu wykonywania programu na Blue Gene/P dzięki zrównolegleniu w środowisku OpenMP. Czasy mierzone w sekundach.
Fig. 6. Time reduction of program performing on Blue Gene/P thanks to parallelization in OpenMP environment. Times measured in seconds.



Ryc. 7. Redukcja czasu wykonywania programu na Blue Gene/Q dzięki zrównolegleniu w środowisku OpenMP. Czasy mierzone w sekundach.
Fig. 7. Time reduction of program performing on Blue Gene/Q thanks to parallelization in OpenMP environment. Times measured in seconds.



Ryc. 8. Redukcja czasu wykonywania programu na IBM Power 775 dzięki zrównolegleniu w środowisku OpenMP. Czasy mierzone w sekundach.
Fig. 8. Time reduction of program performing on IBM Power 775 thanks to parallelization in OpenMP environment. Times measured in seconds.



Ryc. 9. Redukcja czasu wykonywania programu na 2x Intel Xeon X5660 dzięki zrównolegleniu w środowisku OpenMP. Czasy mierzone w sekundach.
Fig. 9. Time reduction of program performing on 2x Intel Xeon X5660 thanks to parallelization in OpenMP environment. Times measured in seconds.

Zestawienie maksymalnych przyśpieszeń jakie udało się uzyskać na badanych czterech platformach sprzętowych przedstawiono w Tab. 6.

Tab. 6. Uzyskane maksymalne przyśpieszenie dzięki zastosowaniu rozwiązania zrównoleglonego w środowisku OpenMP na różnych platformach sprzętowych.
Tab. 6. Maximal acceleration obtained thanks to the use of parallel solution in OpenMP environment on various hardware platforms.

	IBM Blue Gene/P	IBM Blue Gene/Q	IBM Power 775	2x Intel Xeon X5660
Maksymalne uzyskane przyśpieszenie:	2.209	11.347	11.795	5,337

Porównanie wydajności najszybszego z zaimplementowanych rozwiązań z implementacją CUDA GPU zrealizowaną przez Jeet'a Chauhan'a

Porównano wydajność własnych rozwiązań z implementacją zaprezentowaną przez Jeet'a Chauhan'a [8]. Jak się okazało oprogramowanie Chauhan'a, pomimo że wykorzystywało GPU rozwiązywało problem w sposób wysoce niezadowalający pod względem wydajnościowym. Nie realne w tym przypadku okazały się testy dla *max_base = 500* oraz *max_powers = 500* – zajęłyby relatywnie dużo czasu. Dlatego też w tym przypadku dla celów porównawczych zmniejszono badany przedział do *max_base = 100* oraz *max_powers = 100*.
Wyniki zaprezentowano w Tab. 8.

Tab. 8. Porównanie wydajności programu GPGPU Jeet'a Chauhan'a z oprogramowaniem własnym na kilku różnych platformach sprzętowych. Źródło: Opracowanie własne.

	Procesor CPU	RAM	Procesor GPU	Kompilator	Czas realizacji
Jeet Chauhan Implementation	Intel i7 920 @ 2.80 GHz	24 GB DDR3	nVidia Tesla C2050	Cuda compilation tools, release 3.2, V0.2.1221	953,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K10	Cuda compilation tools, release 4.2, V0.2.1221	1116,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K10	Cuda compilation tools, release 5.0, V0.2.1221	1160,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K10	Cuda compilation tools, release 6.0, V6.0.1	1160,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K20	Cuda compilation tools, release 4.2, V0.2.1221	585,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K20	Cuda compilation tools, release 5.0, V0.2.1221	585,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K20	Cuda compilation tools, release 6.0, V6.0.1	588,000 s
Łukasz Świerczewski Implementation	Intel i7 920 @ 2.80 GHz – 1 wątek	24 GB DDR3	Nie dotyczy	gcc version 4.1.2 20080704 (Red Hat 4.1.2-52)	1,790 s
Łukasz Świerczewski Implementation	Intel i7 920 @ 2.80 GHz – 8 wątków	24 GB DDR3	Nie dotyczy	gcc version 4.1.2 20080704 (Red Hat 4.1.2-52)	0,472 s
Łukasz Świerczewski Implementation	Intel Xeon E5-2670 @ 2.60GHz – 1 wątek	512 GB RAM	Nie dotyczy	gcc version 4.4.7 20120313 (Red Hat 4.4.7-11) (GCC)	1,100 s
Łukasz Świerczewski Implementation	Intel Xeon E5-2670 @ 2.60GHz – 16 wątków	512 GB RAM	Nie dotyczy	gcc version 4.4.7 20120313 (Red Hat 4.4.7-11) (GCC)	0,298 s

Wnioski i perspektywy dalszych badań

Pomimo wielu prób i doświadczeń przeprowadzonych w ramach pisania tego artykułu nie udało się odnaleźć kontrprzykładu dla przypuszczenia Beal'a.
Podczas obliczeń ograniczono się jedynie do klasycznych procesorów CPU. Można jednak dodatkowo wykorzystać akceleratory graficzne co byłoby bardzo ciekawym podejściem do problemu. Dzięki takiemu rozwiązaniu najprawdopodobniej możliwe byłoby uzyskanie znacznie lepszych wyników końcowych. Oczywiście istnieją już zaprogramowane implementacje w CUDA [7] – należy do nich m.in. rozwiązanie zaprezentowane w pracy magisterskiej Jeet'a Chauhan'a [8]. Istnieje także implementacja w języku Brook+ [9]. Pomysły te mają jednak swoje ograniczenia i nie nadają się do zastosowania na wielką skalę. Przykładowo w pracy Jeet'a Chauhan'a [8] rozważono algorytm działający na GPU ale jedynie dla bardzo niewielkich liczb. Przeprowadzone analizy na potrzeby tej pracy wykazały także, że oprogramowanie Jeet'a Chauhan'a jest niesamowicie wolne. Połączeniem możliwości jakie daje środowisko MPI [10] z wykorzystaniem akceleratorów graficznych zajął się także Naveen Kumar Reddy Nandipati [11]. W tym miejscu należy także wspomnieć o standardzie OpenCL [12], który jest jednolitym środowiskiem umożliwiającym programowanie kart graficznych (zarówno AMD jak i nVidia) oraz np. procesorów IBM Cell [13]. Do obliczeń można także wykorzystać platformę BOINC [14] [15] (Berkeley Open Infrastructure for Network Computing), która umożliwiłaby połączenie mocy obliczeniowych tysięcy komputerów połączonych za pomocą sieci Internet. Inne aspekty badawcze, które można wykorzystać w dalszej parcy zaprezentowali polscy badacze [16] [17].

Podziękowania

Obliczenia wykonano w Interdyscyplinarnym Centrum Modelowania Matematycznego i Komputerowego (ICM) Uniwersytetu Warszawskiego w ramach grantu obliczeniowego nr G55-11.

Literatura

1. Mauldin, R. Daniel. "A generalization of Fermat's Last Theorem: the Beal conjecture and prize problem." *Notices of the American Mathematical Society* 44.11 (1997), 1436-1437.
2. Beal Prize - American Mathematical Society, URL: http://www.ams.org/profession/prizes-awards/ams-supported/beal-prize, Ostatni dostęp: 18.04.2014
3. Beal Conjecture, URL: http://www.bealconjecture.com/, Ostatni dostęp: 18.04.2014
4. Beal's Conjecture: A Search for Counterexamples, URL: http://norvig.com/beal.html, Ostatni dostęp: 18.04.2014
5. Dagum, Leonardo, and Ramesh Menon. "OpenMP: an

industry standard API for shared-memory programming." *Computational Science & Engineering*, IEEE 5.1 (1998), 46-55.
6. Chapman, Barbara, Gabriele Jost, and Ruud Van Der Pas. Using OpenMP: *portable shared memory parallel programming*. Vol. 10. MIT press, 2008.
7. Sanders, Jason, and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.
8. Chauhan, Jeet. *Nvidia GeForce 8400 GPGPU implementation of a CUDA C parallel algorithm to search for counterexamples to Beal's Conjecture*. Diss. San Diego State University, 2010.
9. Shah, Nirav. *AMID Firestream 9170 GPGPU implementation of a Brook+ parallel algorithm to search for counterexamples to Beal's Conjecture*. Diss. San Diego State University, 2010.
10. Gropp, William, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press, 1999.
11. Nandipati, Naveen Kumar Reddy. "Heterogeneous NPACI-ROCKS/MPI/CUDA distributed multi-GPGPU application for seeking counterexamples to Beal's Conjecture; ROCKS/CUDA integration component.", 2011.
12. Stone, John E., David Gohara, and Guochun Shi. "OpenCL: A parallel programming standard for heterogeneous computing systems." *Computing in science & engineering* 12.3 (2010): 66.
13. Chen, Thomas, et al. "Cell broadband engine architecture and its first implementation—a performance view." *IBM Journal of Research and Development* 51.5 (2007), 559-572.
14. Anderson, David P. "Boinc: A system for public-resource computing and storage." *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*. IEEE, 2004.
15. Estrada, Trilce, Michela Taufer, and David P. Anderson. "Performance prediction and analysis of BOINC projects: An empirical study with EmBOINC." *Journal of Grid Computing* 7.4 (2009), 537-554.
16. H. Zarzycki H, J. Czerniak, *Development and Code Management of Large Software Systems*, PSZW, nr 27, s. 327-339, Bydgoszcz 2010.
17. H. Zarzycki, Application of the finite difference CN method to value derivatives, PSZW, nr 42, s. 267-277, Bydgoszcz 2011.

Śledzenie obiektów z wykorzystaniem obrazowania spektralnego

Object tracking with spectral imagery

Krzysztof Tutak[1], Mateusz Pieszko[1]

Treść. Niniejsza praca poświęcona jest analizie skuteczności śledzenia obiektów przy pomocy obrazowania spektralnego wykonywanego za pomocą 16-kanałowej kamery spektralnej rejestrującej dane w trybie wideo w zakresie 400-1000 nm. Wykorzystano algorytm Lucas-Kanade, wyznaczający przepływ optyczny w charakterystycznych punktach obrazu, określonych metodą Shi-Tomasi. Śledzenie inicjowane jest ręcznie poprzez wskazanie prostokątnego okna zawierającego obiekt. Do przetwarzania wybierany jest monochromatyczny obraz odpowiadający długości fali, dla której liczba punktów leżących w tym oknie jest największa. Zastosowano reprezentację obrazu w formie piramidy, dzięki czemu zmniejszono zależności od zmian skali obserwowanego obiektu. Otrzymane w każdym kroku śledzenia nowe pozycje punktów charakterystycznych były analizowane w celu odrzucenia obserwacji odstających. Wykonano szereg eksperymentów polegających na próbie śledzenia makiety samochodu wojskowego w trudnych warunkach oświetlenia i przy niejednorodnym tle o kolorystyce zbliżonej do barw maskujących pojazdu. Otrzymane rezultaty potwierdziły zasadność stosowania obrazowania spektralnego do śledzenia obiektów.

Słowa kluczowe: obrazowanie spektralne, przetwarzanie obrazów, śledzenie obiektów, przepływ optyczny, spektralny system wizyjny

Abstract. This paper is devoted to the analysis of the effectiveness of object tracking with spectral imagery performed with a 16-channel spectral video camera operating in the 400-1000 nm range. We used the Lucas-Kanade algorithm which computes the optical flow at characteristic points of the image which were determined by the Shi-Tomasi method. The tracking is initialized manually by pointing to a rectangular window containing the object. Monochrome image corresponding to the wavelength for which the number of points lying in this window is the greatest is selected for processing. We used a representation of an image in the form of a pyramid, so that dependence on scale changes of the observed object was reduced. New positions of characteristic points received in each step of tracking were analyzed in order to reject outliers. We performed a series of experiments that tries to track military vehicle model under difficult lighting conditions and heterogeneous background of a color similar to the vehicle masking colors. Obtained results confirmed the advisability of applying spectral imagery for object tracking.

Keywords: hyperspectral imaging, image processing, object tracking, optical flow, spectral vision system

1. Wprowadzenie

Algorytmy detekcji i śledzenia obiektów są obecnie przedmiotem intensywnego zainteresowania z uwagi na liczne zastosowania w obronności, przemyśle oraz w systemach wspomagających utrzymanie porządku publicznego. Najczęstsze problemy w istniejących rozwiązaniach związane są z niejednorodnością tła, które może dodatkowo mieć zbliżoną barwę do obiektu, niewielkim kontrastem obiektu w stosunku do pozostałych elementów obserwowanej sceny, zmiennymi warunkami oświetlenia i koniecznością śledzenia w nocy, zmiennym kształtem śledzonego celu, częściowymi lub całkowitymi przysłonięciami, obecnością w kadrze innych podobnych obiektów oraz niejednostajnym charakterem ruchu [1].

Obrazowanie spektralne posiada szereg zalet, które powodują, że jego zastosowanie do śledzenia obiektów wydaje się być bardzo obiecujące oraz pozwala na wyeliminowanie przynajmniej części ze wspomnianych ograniczeń. Umożliwia ono mierzenie tzw. sygnatur spektralnych, które zawierają więcej informacji aniżeli trzy podstawowe składowe barwy oferowane przez tradycyjne kamery. Do detekcji obiektu można zatem wykorzystać właściwości spektralne materiału, z którego jest on wykonany zamiast koloru, jasności lub informacji o gradiencie (krawędziach). Dotychczas główną przeszkodą był czas akwizycji obrazów spektralnych pozwalający jedynie na obserwację scen statycznych, jednak na rynku dostępne od niedawna są również kamery spektralne działające w trybie wideo.

2. Przegląd literatury

Dobry przegląd nowoczesnych metod śledzenia z wykorzystaniem tradycyjnych kamer można znaleźć w [2]. Metody te nie są jednak uniwersalne i wciąż obarczone są ograniczeniami takimi jak szybkość, obrót czy częściowe przesłonięcia śledzonych obiektów, a także problemami z segmentacją obiektu od tła. Część z tych ograniczeń

1.Enformatic Sp. z o.o., ul. Mieszka I 73A, 35-303 Rzeszów, e-mail:krzysztof.tutak@enformatic.eu, mpieszko@enformatic.eu

może być wyeliminowana dzięki zastosowaniu kamer spektralnych.

W literaturze dostępne są również publikacje z zakresu śledzenia obiektów przy pomocy obrazowania spektralnego. W pracy [3] autorzy wykorzystali algorytm *mean shift* do śledzenia obiektów na podstawie widma ich reflektancji, predykcji położenia i redukcji wymiarowości metodą *random projection*. W artykule [4] wykorzystano filtr cząsteczkowy do śledzenia osób na podstawie hiperspektralnych właściwości skóry w zakresie światła widzialnego i bliskiej podczerwieni. W pracy [5] wykorzystano połączenie analizy cech spektralnych oraz zachowania śledzonego obiektu w czasie przy wykorzystaniu znanych metod wykrywania celu i algorytmu *variance-filter*. Śledzenie pojazdów jest przedmiotem pracy [6], w której autorzy opisali własną wersję algorytmu *feautre aided tracking*. W celu śledzenia obiektów o niewielkim rozmiarze poruszających się z dużą prędkością, w artykule [7] zastosowano metodę opartą o algorytm Mean-Shift oraz filtr Kalmana w celu zbudowania estymatora jądrowego gęstości obiektu. Z kolei w publikacji [8] zaproponowano system złożony z komponentu umożliwiającego panoramiczne widzenie peryferyjne oraz z modułu hiperspektralnego o wąskim polu widzenia.

Dostępne ogólnie pozycje literaturowe odnoszą się do wykorzystania kamer rejestrujących dużą liczbę pasm kosztem długiego czasu akwizycji. Dzięki rozwojowi technologii na rynku pojawiły się ostatnio kamery spektralne rejestrujące mniej pasm, lecz działające w czasie rzeczywistym. Tego typu urządzenia są obecnie przedmiotem wielu badań, zaś ich skuteczność nie została jeszcze opisana w literaturze w wystarczającym stopniu. Wśród nielicznych dostępnych publikacji z tego zakresu należy wymienić pracę [9], w której wykorzystano sensor kontrolowany przy pomocy metodologii DDDAS (Dynamic Data Driven Applications Systems) w celu szybkiej rejestracji wybranych danych spektralnych.

3. Opis metody

3.1. Pojęcie przepływu optycznego

Zastosowana w pracy metoda śledzenia wykorzystuje tzw. przepływ optyczny, który można interpretować jako pole wektorowe określające przesunięcia zawartości dwóch kolejnych obrazów [10]. Niech $I(x,y,t)$ oznacza jasność piksela obrazu o współrzędnych x, y w chwili t. Piksel ten przemieszcza się w kolejnym obrazie o $\Delta x, \Delta y$ i Δt przy czym zakłada się spełnienie następującego warunku:

$$I(x, y, t)=I(x+\Delta x, y+\Delta y, t+\Delta t) \tag{1}$$

Dla niewielkich wartości przesunięć można zastosować rozwinięcie w szereg Taylora:

$$I(x+\Delta x, y+\Delta y, t+\Delta t)= =I(x, y, t)+\frac{\partial I}{\partial x}\Delta x+\frac{\partial I}{\partial y}\Delta y+\frac{\partial I}{\partial t}\Delta t+reszta \tag{2}$$

Z równań (1) i (2) przy pominięciu reszty wynika, że:

$$\frac{\partial I}{\partial x}\Delta x+\frac{\partial I}{\partial y}\Delta y+\frac{\partial I}{\partial t}\Delta t=0 \tag{3}$$

Po obustronnym podzieleniu przez Δt :

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t}+\frac{\partial I}{\partial y}\frac{\Delta x}{\Delta t}+\frac{\partial I}{\partial t}=0 \tag{4}$$

Oznaczając pochodne cząstkowe obrazu po x, y, i t odpowiednio I_x, I_y, i I_t zaś składowe przepływu optycznego wzdłuż osi x, y odpowiednio przez V_x i V_y można zapisać równanie przepływu optycznego:

$$I_xV_x+I_yV_y=-I_t \tag{5}$$

Równanie (5) zawiera dwie niewiadome. Do ich wyznaczenia potrzebny jest zatem dodatkowy warunek.

3.2. Wyznaczanie przepływu metodą Lucas-Kanade

Do wyznaczenia przepływu optycznego może być wykorzystana metoda Lucas-Kanade [11]. Zakłada się, że przepływ optyczny jest niezmienny w pewnym niewielkim otoczeniu rozpatrywanego punktu. W związku z tym można zapisać następujący układ równań:

$$\begin{cases} I_x(p_1)V_x+I_y(p_1)V_y=-I_t(p_1) \\ I_x(p_1)V_x+I_y(p_1)V_y=-I_t(p_2) \\ \vdots \\ I_x(p_n)V_x+I_y(p_n)V_y=-I_t(p_n) \end{cases} \tag{6}$$

gdzie $p_1, p_2, \ldots, p_n$ oznaczają kolejne piksele w rozpatrywanym fragmencie obrazu.

Układ równań (6) można zapisać w formie macierzowej:

$$AV=B$$

gdzie:

$$A=\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix}, V=\begin{bmatrix} V_x \\ V_y \end{bmatrix}, B=\begin{bmatrix} -I_t(p_1) \\ -I_y(p_2) \\ \vdots \\ -I_y(p_n) \end{bmatrix} \tag{7}$$

Układ równań (7) można rozwiązać metodą najmniejszych kwadratów:

$$V = \left(A^T A \right)^{-1} A^T B \tag{8}$$

Znajomość składowych wektora V_x, V_y oraz czasu Δt pomiędzy dwiema kolejnymi ramkami obrazu pozwala na wyznaczenie nowej pozycji śledzonego punktu charakterystycznego.

3.3. Wykorzystanie piramidy obrazów

W podstawowej wersji algorytmu Lucas-Kanade rozważane jest niewielkie sąsiedztwo piksela, w którym wyznaczany jest przepływ optyczny. Oznacza to, że większe przesunięcia obiektów mogą być niezauważone przez metodę. W związku z tym w pracy wykorzystano wersję opartą na tzw. piramidzie obrazów [12]. Jest to sposób reprezentacji obrazu w wielu skalach. Powstaje on w wyniku wielokrotnego wygładzania a następnie zmniejszania obrazu. Powstałe w trakcie obrazy, nałożone jeden na drugi, tworzą piramidę, stąd nazwa tej reprezentacji.
W zmodyfikowanej wersji metody Lucas-Kanade rozmiar sąsiedztwa jest stały niezależnie od skali obrazu, a wyznaczanie przepływu optycznego odbywa się rekurencyjnie, zaczynając od najwyższego poziomu. Oznacza to, że początkowo w sposób zgrubny przeszukiwany jest większy fragment obserwowanej sceny, a następnie informacja jest uszczegółowiana na niższych poziomach piramidy.

3.4. Procedura wyboru kanału

W opisywanym rozwiązaniu algorytm Lucas-Kanade został wykorzystany do wyznaczenia tzw. rzadkiego przepływu optycznego, wyliczanego tylko w pewnych charakterystycznych punktach obrazu. Przyjęto, że dobrymi cechami do śledzenia będą narożniki wykryte za pomocą metody Shi – Tomasi [13]. Wykorzystywana kamera spektralna umożliwia akwizycję 16 monochromatycznych obrazów odpowiadających poszczególnym długością fali, nazywanych dalej kanałami. Śledzenie rozpoczyna się od ręcznego wskazania prostokątnego obszaru zawierającego obiekt. W obrębie tego obszaru znajdowane są punkty charakterystyczne we wszystkich 16 kanałach (ryc. 1).

Ryc. 1. Wyniki detekcji punktów charakterystycznych w 16 kanałach pozyskiwanych za pomocą kamery spektralnej (w nawiasie podano liczby znalezionych punktów).
Fig. 1. Results of the detection of characteristic points in 16 bands acquired by spectral camera (the number of found points is indicated in brackets).

Do dalszego przetwarzania wybrano długość fali, dla której liczba punktów charakterystycznych jest największa. W rozpatrywanym przypadku był to kanał nr 6. Procedura wyboru kanału powtarzana jest za każdym razem, gdy śledzenie jest inicjowane. Dzięki temu do dalszego przetwarzania wybierany jest kanał, w którym w warunkach panujących w danej chwili znaleziono najwięcej punktów charakterystycznych.

3.5. Przetwarzanie końcowe

W każdym kroku śledzenia, po wyznaczeniu nowych położeń punktów charakterystycznych wykonywana jest procedura mająca na celu wyeliminowanie obserwacji odstających. W tym celu wyznaczany jest środek ciężkości : $P_c = (x_c, y_c)$:

$$x_c = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad y_c = \frac{1}{m}\sum_{i=1}^{m} y_i \tag{9}$$

gdzie x_i, y_i to współrzędne, zaś m liczba punktów charakterystycznych wyznaczonych w danej iteracji śledzenia. Następnie wyznaczana jest średnia odległość punktów od P_c :

$$d = \frac{1}{m}\sum_{i=1}^{m} d_i \tag{10}$$

gdzie:

$$d_i = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} \tag{11}$$

Dany punkt jest odrzucany jeżeli spełnia warunek:

$$d_i > K \cdot d \tag{12}$$

gdzie K jest parametrem metody. W rozpatrywanych przykładach przyjęto $K = 3$.

4. Eksperymenty

4.1. Opis stanowiska badawczego

W trakcie badań wykorzystano kamerę spektralną „SOC716-VNIR" wyprodukowaną przez firmę Surface Optics Corportation [14]. Przyrząd ten rejestruje sygnatury spektralne badanych obiektów w 16 kanałach z maksymalną szybkością 30 klatek na sekundę. Na potrzeby badań stworzone zostało otoczenie pomiarowe, w którym rozmieszczono elementy tła zbliżone do faktury oraz kolorów śledzonego obiektu.

4.2. Opis bazy sekwencji wideo

Algorytm testowano dla scen na zewnątrz w warunkach naturalnego oświetlenia. Przedmiotem śledzenia była makieta samochodu wojskowego. W celu oceny skuteczności metody opracowano 9 scenariuszy śledzenia. Dla każdego ze scenariuszy zarejestrowano 20 sekwencji testowych w rozdzielczości 520×520 zawierających śledzony obiekt poruszający się ze zmienną prędkością. Poszczególne sekwencje były rejestrowane o różnych porach dnia i przy rozmaitych warunkach pogodowych. Zmieniano odległość kamery od celu oraz konfigurację tła. Obiekt poruszał się pod różnym kątem w stosunku do osi widzenia kamery. Zarejestrowane sekwencje trwały kilkanaście sekund, co przy zastosowanym czasie integracji odpowiada średnio 130 klatkom. Wybrane klatki przykładowej sekwencji testowej pokazano na ryc. 2.

Ryc. 2. Wybrane klatki pokazujące wynik śledzenia dla scenariusza 2.
Fig. 2. Selected frames showing the result of tracking for scenario 2.

4.2.1. Opis scenariuszy śledzenia.

Scenariusz 1: Wykonana w skali 1:20 makieta samochodu wojskowego z nadrukiem w barwach ochronnych zapewniających kamuflaż poruszała się ruchem niejednostajnym ze zmienną prędkością oraz przerwami w jeździe. Obiekt przemieszczał się równolegle to tła, które stanowi zróżnicowana roślinność naturalna. Obserwacja dokonywana była z dużej odległości. Jednocześnie w niewielkim zakresie zmieniał się kąt patrzenia kamery. Śledzonym elementem była naczepa makiety pojazdu.
Scenariusz 2: Makieta samochodu poruszała się w sposób analogiczny do scenariusza 1. W scenariuszu tym tło zostało rozszerzone o dodatkowe elementy niewystępujące w sekwencji 1. Obserwacja odbywała się z bliskiej odległości, a kąt patrzenia kamery zmieniany był w większym zakresie niż w scenariuszu 1. Śledzonym elementem była naczepa makiety pojazdu.
Scenariusz 3: Makieta samochodu poruszała się ruchem niejednostajnym z przerwami w jeździe, pod kątem około 18 stopni w stosunku do tła (zróżnicowana roślinność naturalna) w taki sposób, że obiekt przybliżał się do kamery w trakcie ruchu. Kąt patrzenia kamery zmieniał się w trakcie pomiaru. Rejestracja sekwencji odbywała się z dużej odległości. Śledzonym elementem była naczepa pojazdu.
Pozostałe scenariusze: Scenariusze 4-9 opierały się na scenariuszach 1-3. Jedyna modyfikacja polegała na zmianie śledzonego fragmentu makiety (zgodnie z tab. 1.). Scenariusze 4 i 5 zostały oparte na scenariuszu 1, scenariusze 6 i 7 na scenariuszu 2, zaś scenariusze 8 i 9 zostały oparte na scenariuszu 3.

4.3. Opis eksperymentu

Dla każdej z zarejestrowanych sekwencji eksperyment rozpoczynał się od ręcznego zaznaczenia śledzonego obiektu. Oceniano przez jak długi okres czasu, rozumiany jako liczba klatek obrazu, algorytm jest w stanie poprawnie śledzić cel. Przyjęto, że obiekt przestał być śledzony w sytuacji, gdy nie znaleziono w ogóle punktów w następnej klatce lub rozrzut znalezionych punktów jest na tyle duży, że pole powierzchni obejmującego je prostokąta jest ponad dwa razy większe od zakreskowanej na ryc. 3 na niebiesko części wspólnej tego prostokąta i rzeczywistego położenia śledzonego obszaru.

Ryc. 3. Przykładowa klatka pokazująca moment przerwania śledzenia
Fig. 3. Sample frame showing the moment of tracking interruption

4.4. Ocena wyników

Średnie wyniki skuteczności otrzymane dla poszczególnych scenariuszy zostały porównane z wynikami uzyskanymi dla metody Lukas-Kanade [15] zastosowanej na sekwencjach obrazów przedstawiających te same sceny rejestrowane za pomocą kamery RGB (tab. 1). Skuteczność śledzenia zdefiniowano jako wyrażony w procentach stosunek liczby klatek, w których śledzenie jest poprawne do całkowitej liczby klatek zawierających obiekt.
Wyniki z tab. 1. pokazują, że badany algorytm jest szczególnie skuteczny w przypadku gdy zaznaczonym do śledzenia fragmentem obiektu jest kabina makiety (w przypadku dwóch scenariuszy zawsze uzyskiwano wynik w wysokości 100%). Wyniki uzyskane dla innych fragmentów obiektu są zróżnicowane, jednakże nie zaobserwowano dramatycznego pogorszenia. Biorąc pod uwagę to, że przedmiotem eksperymentu było śledzenie celu bardzo podobnego do tła w różnych warunkach oświetlenia, wyniki należy uznać za satysfakcjonujące. Dla rozpatrywanego obiektu opracowana metoda okazała się skuteczniejsza niż metoda Lukas-Kanade zastosowana dla sekwencji obrazów RGB. Ponadto, przebadano również skuteczność metody Cam-Shift [15] dla tej samej sekwencji obrazów. Okazała się ona jednakże nieskuteczna dla tego obiektu w przyjętych warunkach oświetleniowych.

Tab. 1. Porównanie średniej skuteczność śledzenia dla rozpatrywanych 9 scenariuszy
Tab. 1. Comparison of the average effectiveness of tracLukas-Kanade dla kamery RGB [%]

Nr scenariusza	1	2	3	4	6	8	5	7	9
Śledzony element	naczepa			tylne koło			kabina		
Badana metoda [%]	78,80	93,20	89,86	82,61	97,28	78,26	100,00	100,00	95,65
Lukas-Kanade dla kamery RGB [%]	75,61	85,62	62,35	82,61	97,28	55,30	98,45	73,70	92,65

5. Podsumowanie

Wyniki przeprowadzonych eksperymentów pokazują, że możliwe jest skuteczne śledzenie obiektów w oparciu o technologię obrazowania spektralnego oraz metodę Lucas-Kanade nawet w przypadku obiektów bardzo zbliżonych do tła, w odniesieniu do których tradycyjne metody okazują się zazwyczaj niewystarczające. W ramach przeprowadzonych eksperymentów, w których śledzona była makieta samochodu wojskowego w barwach ochronnych na tle bardzo podobnego do obiektu tła, osiągnięto łącznie średnią skuteczność śledzenia w wysokości 90,63%. Przeprowadzone eksperymenty pokazują, że skuteczność metody zależy między innymi od tego, która część obiektu oraz jak duży jej fragment jest wykorzystywany do jego śledzenia.
Wśród zalet zaproponowanej metody należy wymienić możliwość śledzenia obiektów, których kolor jest zbliżony do tła. Stwierdzono także, że śledzenie jest skuteczne przy dużych zmianach rozmiarów obserwowanego obiektu, co jest szczególnie ważne w przypadku gdy obiekt porusza się w kierunku kamery. Stwierdzono eksperymentalnie, że algorytm działa poprawnie w słabszych warunkach oświetlenia, np. wieczorem. Jako wadę należy wymienić to, że śledzenie nie działa w przypadku, gdy warunki oświetlenia zmieniają się w trakcie procesu śledzenia.

6. Literatura

[1] A. Yilmaz , O. Javed , M. Shah, Object tracking: A survey, ACM Computing Surveys vol. 38, no. 4 (2006), 38.
[2] H. Yang, L. Shao, F. Zheng, L. Wang, Z. Song, Recent advances and trends in visual tracking: A review, Neurocomputing 74 (2011), 3823-3831.
[3] H. V. Nguyen, A. Banerjee, R. Chellappa, Tracking via Object Reflectance using a Hyperspectral Video Camera, IEEE Computer Society Conference (2010), 44 - 51.
[4] H. V. Nguyen, A. Banerjee, P. Burlina, J. Broadwater, R. Chellappa, Tracking and Identification via Object Reflectance Using a Hyperspectral Video Camera, Machine Vision Beyond Visible Spectrum (2011), 201-219.
[5] B. Aminov, O. Nichtern, S. R. Rotman, Spatial and temporal point tracking in real hyperspectral images, EURASIP Journal on Advances in Signal Processing no. 1 (2011).
[6] J. Blackburn, M. Mendenhall, A. Rice, P. Shelnutt, N. Soliman, J. Vasquez, Feature Aided Tracking with Hyperspectral Imagery, Signal and Data Processing of Small Targets (2007).
[7] H. Sheng, Y. Ouyang , C. Li, W. Rong, Z. Xiong, Tracking Dim–Small Object Based on the Hyperspectral Features, Arabian Journal for Science and Engineering, vol. 39 (2014), 1725-1736.
[8] T. Wang, Z. Zhu, E. Blasch, Bio-Inspired Adaptive Hyperspectral Imaging for Real-Time Target Tracking, Sensors Journal IEEE vol. 10, no. 3 (2010), 647-654.
[9] B. Uzkent, M. Hoffman, A. Vodacek, Spectral Validation of Measurements in a Vehicle Tracking DDDAS, Procedia Computer Science vol. 51 (2015), 2493–2502.
[10] S. S. Beauchemin, J. L. Barron, The computation of optical flow, ACM Computer Surveys vol. 27, no. 3 (1995), 433-466.
[11] B. D. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, Proceedings of the 7th joint conference on Artificial intelligence vol. 2 (1981), 674-679.
[12] T. Lindeberg, Scale-Space Theory in Computer Vision, Kluwer Academic Publishers 1994.
[13] J. Shi, C. Tomasi, Good Features to Track. Technical

Report, Cornell University 1993.
[14] Strona internetowa Surface Optics Corporation. (2015-05-18). Hyperspectral and Multispectral Imagers [Online]. Dostępne pod adresem: http://surfaceoptics.com/products/hyperspectral-imaging
[15] G. Bradski, Learning OpenCV, O'Reilly 2008.

Badania realizowane w ramach Projektu "Zbadanie możliwości śledzenia, identyfikacji i kontroli jakości przy użyciu 16-32 kanałowej kamery do obrazowania spektralnego w zakresie 400-1000 nm", Nr WNDRPPK.01.03.00-18-044/13 współfinansowanego ze środków Unii Europejskiej z Europejskiego Funduszu Rozwoju Regionalnego oraz Budżetu Państwa w ramach Regionalnego Programu Operacyjnego Województwa Podkarpackiego. Inwestujemy w rozwój województwa podkarpackiego.