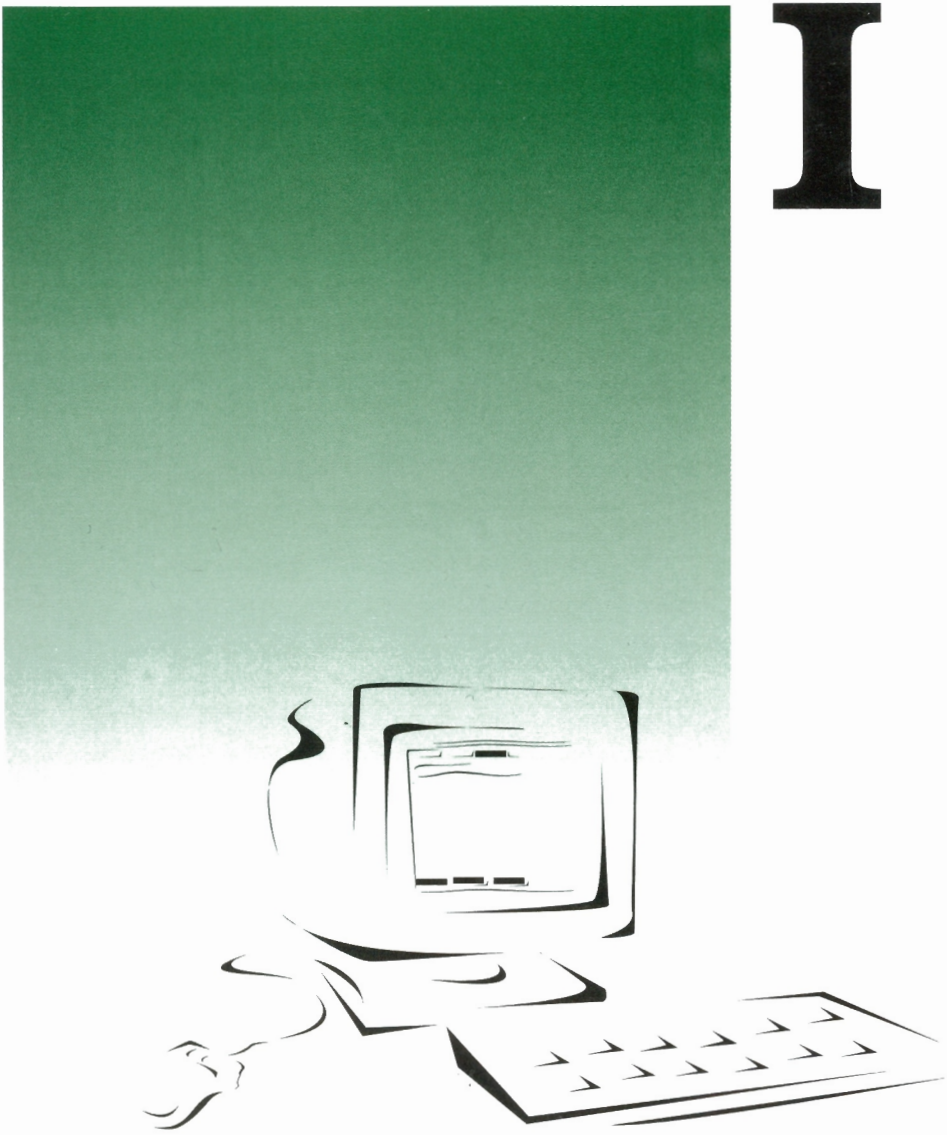
POLITECHNIKA BIAŁOSTOCKA

INFORMATYKA

zeszyty naukowe

ISSN 1644-0331

I



3
2008

Zeszyty Naukowe Politechniki Białostockiej

INFORMATYKA

Zeszyt 3

Wydawnictwo Politechniki Białostockiej
Białystok 2008

Redaktor naukowy:
dr inż. Marek Krętowski

Recenzenci:
dr hab. inż. Jarosław Arabas
dr hab. Anna Bartkowiak, prof. UWr
dr hab. inż. Stanisław Deniziak, prof. PK
dr hab. inż. Mieczysław A. Kłopotek, prof. AP
dr hab. Grażyna Mirkowska-Salwicka, prof. PJWSTK
prof. dr hab. inż. Evgeny Ochim
dr hab. Jerzy Stefanowski
dr hab. inż. Michał Strzelecki, prof. PŁ
dr hab. inż. Piotr J. Suchomski
prof. dr hab. Jerzy Weres
dr hab. inż. Aleksander Zgrzywa, prof. PWr
dr hab. inż. Eugeniusz Zieniuk, prof. UwB

Opracowanie redakcyjne:
Jadwiga Żukowska

Sekretarz redakcji:
mgr inż. Tomasz Łukaszuk

ISSN 1644-0331

Druk:
Dział Wydawnictw i Poligrafii Politechniki Białostockiej

Nakład:100 egz.

SPIS TREŚCI

CONTENTS

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Leon Bobrowski[1,2], Tomasz Łukaszuk[1]

SZEREGOWANIE ZADAŃ OBLICZENIOWYCH Z ZASTOSOWANIEM MODELU RANGOWEGO

Streszczenie: Zagadnienia szeregowania zadań pojawiają się między innymi w kontekście problemów realizowalności dużych procesów obliczeniowych i ich optymalizacji. Przy rozstrzyganiu tego typu problemów można wykorzystywać metody regresji rangowej. Do celów konstrukcji modeli regresji rangowej poszczególne zadania obliczeniowe charakteryzowane są poprzez wielowymiarowe wektory zależności. Wektory zależności pozwalają stwierdzić czy określone zadanie może być zrealizowane tylko wtedy, gdy zostaną wcześniej zrealizowane pewne inne zadania. Regresja rangowa obejmuje konstrukcję takich odwzorowań liniowych z wielowymiarowej przestrzeni zależności na przestrzeń jednowymiarową (linię czasu), która odzwierciedla w możliwie dużym stopniu zależności pomiędzy zadaniami.

Słowa kluczowe: szeregowanie zadań obliczeniowych, model rangowy, wypukła i odcinkowo-liniowa funkcja kryterialna (CPL)

1. Wstęp

Realizację dużych zadań obliczeniowych wiąże się na ogół z dążeniem do maksymalnego skracania czasu obliczeń przy spełnieniu wszystkich warunków ograniczających [4], [7], [6]. Dla skrócenia czasu obliczeń, procesy obliczeniowe dekomponuje się na podzadania O_j, które następnie realizuje się szeregowo poprzez jeden procesor lub w sposób równoległy przy o większej liczbie procesorów. Modele szeregowania zadań pomocne są zarówno przy weryfikacji warunków realizowalności zdekomponowanych programów, jak również przy skracaniu czasu obliczeń poprzez odpowiednie rozdzielanie zadań na poszczególne procesory. W przedstawionym artykule analizowana jest metoda badania niesprzeczności, szeregowania oraz optymalizacji zadań obliczeniowych O_j, oparta na koncepcjach regresji rangowej [2].

Modele regresji rangowej mają postać takich odwzorowań liniowych z wielowymiarowej przestrzeni cech na linię prostą, które w możliwie dużym stopniu zachowują zadany porządek w wybranych parach zdarzeń lub obiektów [2]. Porządek

[1] Wydział Informatyki, Politechnika Białostocka, Białystok
[2] Instytut Biocybernetyki i Inżynierii Biomedycznej, PAN, Warszawa

realizacji dwu zadań O_j i O_k oznacza, że do realizacji zadania O_k potrzebna jest znajomość pewnych wyników realizacji zadania O_j. W rezultacie wcześniejsza realizacja zadania O_j jest warunkiem koniecznym realizacji zadania O_k. Wymagany porządek realizacji zadań obliczeniowych można reprezentować w postaci wielowymiarowych wektorów zależności. Wektor zależności zadania O_k identyfikuje takie zadania O_j $(j \neq k)$, których wcześniejsza realizacja jest warunkiem koniecznym realizacji zadania O_k [1].

Konstrukcja rangowych odwzorowań liniowych została oparta na wyznaczaniu hiperłaszczyzny możliwie najlepiej rozdzielającej dwa zbiory w przestrzeni zależności. Wyznaczanie hiperłaszczyzny rozdzielającej można oprzeć na minimalizacji wypukłej i odcinkowo liniowej (*CPL*) funkcji kryterialnej [2]. Funkcja kryterialna definiowana jest w tym przypadku na podstawie wektorów zależności charakteryzujących poszczególne zadania obliczeniowe O_k. Algorytmy wymiany rozwiązań bazowych, podobne do programowania liniowego, pozwalają wyznaczać minimum funkcji typu *CPL* w sposób efektywny nawet w przypadku dużej liczby wektorów zależności.

2. Charakterystyka zadań obliczeniowych przez wektory zależności

Załóżmy, że realizowany proces obliczeniowy został podzielony (zdekomponowany) na m zadań O_k $(k = 1,...,m)$. Każde z zadań O_k zostało scharakteryzowane przez czas realizacji τ_k oraz przez wektor zależności $\rho_{\mathbf{k}} = [\rho_{k1},...,\rho_{km}]^T$ o m składowych binarnych ρ_{kj} ($\rho_{kj} = 1$ lub $\rho_{kj} = 0$). Wartość $\rho_{kj} = 1$ oznacza, że k-te zadanie O_k może być realizowane tylko wtedy, gdy wcześniej będzie zrealizowane zadanie O_j. Wartość $\rho_{kj} = 0$ oznacza, że k-te zadanie O_k może być realizowane niezależnie od stanu realizacji zadania O_j. Zakładamy przy tym, że zadania O_k i O_j nie mogą blokować się wzajemnie, tj.:

$$(\forall k, j \in \{1,...,m\})\ \rho_{kk} = 0 \quad oraz \quad (\rho_{kj} = 1) \Rightarrow (\rho_{jk} = 0) \tag{1}$$

Na podstawie wektorów zależności $\rho_{\mathbf{k}}$ wprowadzimy rangową *relację poprzedzania* $O_j \prec O_k$ pomiędzy zadaniami O_j i O_k, która oznacza, że zadanie O_j poprzedza O_k:

$$(O_j \prec O_k) \Leftrightarrow (\rho_{kj} = 1) \tag{2}$$

Przyjmujemy, że zarówno podział procesu obliczeniowego na zadania O_k jak również charakterystyka tych zadań za pomocą wektorów zależności ρ_k oraz czasów

obliczeń τ_k wynikają z wiedzy ekspertów o realizowanym procesie. Np. w hierarchicznej strukturze warstwowej realizacja każdego zadania $O_{k(l)}$ w l-tej warstwie jest możliwa tylko wtedy, gdy wszystkie zadania $O_{k(l-1)}$ w warstwie wcześniejszej zastaną zrealizowane. Przy realizacji złożonych procesów obliczeniowych liczba m zadań O_k może być bardzo duża. W takich okolicznościach pojawia się zagrożenie, że proponowany przez ekspertów podział procesu na zadania O_k, scharakteryzowane za pomocą wektorów zależności $\rho_{\mathbf{k}}$ oraz czasów obliczeń τ_k, spowoduje pojawienie się trudności w realizacji procesu obliczeniowego. Realizacja procesu może być niemożliwa z powodu pojawienia się ciągu zależności sprzecznych, wzajemnie blokujących się. Blokowanie się procesu obliczeniowego może nastąpić na przykład wówczas, gdy zadanie O_k odwołuje się do zadania O_l, a zadanie O_l odwołuje się do zadania O_m, które może być zrealizowane po wykonaniu zadania O_k. W tym przypadku podzbiór rangowych relacji poprzedzania (2) ma poniższą postać:

$$\{(O_l \prec O_k), (O_m \prec O_l), (O_k \prec O_m)\} \tag{3}$$

Relacja $\prec$ jest przechodnia (ang. *transient relation*) wtedy i tylko wtedy, gdy dla dowolnych O_k, O_l, O_m spełniona jest poniższa implikacja:

$$\textit{jezeli} \quad (O_l \prec O_k) \quad \textit{oraz} \quad (O_k \prec O_m), \quad \textit{to} \quad (O_m \prec O_l) \tag{4}$$

Zauważmy, że pary ze zbioru (3) nie tworzą relacji przechodniej. Reprezentacja zbioru (3) za pomocą grafu skierowanego z wierzchołkami utworzonymi przez zadania O_k pozwala uwypuklić istnienie pętli, która jest źródłem braku przechodniości i zablokowania się procesu.

Blokujące się łańcuchy zależności mogą mieć oczywiście długość większą niż trzy. Łatwo weryfikowalne warunki (1) zabezpieczają przed blokowaniem się wzajemnym w parach zadań. Weryfikacja zagrożenia sprzecznościami i blokowaniem się w długich sekwencjach zadania O_k wymaga wprowadzenia innych sposobów analizy. Istnieje możliwość wykorzystania do tego celu regresji rangowej [2]. Innym problemem, który może być również analizowany za pomocą regresji rangowej jest rozpatrywanie możliwości skrócenia czasu realizacji procesu obliczeniowego, poprzez równoległą realizację pewnych zadań O_k. Model regresji rangowej pozwala w pewnych przypadkach wskazać takie zadania O_k, których równoległa realizacja skróci czas przebiegu procesu.

3. Liniowe odwzorowania rangowe

Rozważmy liniowe odwzorowanie $t(\mathbf{w}) = \mathbf{w}^T\rho$, gdzie $\mathbf{w} = [w_1, ..., w_m]^T$ $(\mathbf{w} \in R^m)$ jest wektorem parametrów, a $\rho = [\rho_{\mathbf{1}}, ..., \rho_{\mathbf{m}}]$ macierzą utworzoną z m-wymiarowych

wektorów zależności $\rho_{\mathbf{k}}$ (1). Prosta $t(\mathbf{w}) = \mathbf{w}^T\rho$ może reprezentować kolejność rozpoczęcia realizacji t_k poszczególnych zadań O_k:

$$(\forall k \in \{1,...,m\}) \quad t_k = \mathbf{w}^T\rho_{\mathbf{k}} \tag{5}$$

Projektując rangowy model zależności poszukujemy takiego optymalnego wektora parametrów $\mathbf{w}^*$, dla którego odwzorowanie liniowe $t(\mathbf{w}) = \mathbf{w}^T\rho$ w najlepszym stopniu zachowuje poniższe *implikacje rangowe*:

$$\begin{array}{l} O_j \prec O_k \Rightarrow (\mathbf{w}^*)^T\rho_{\mathbf{j}} < (\mathbf{w}^*)^T\rho_{\mathbf{k}} \quad oraz \\ O_k \prec O_j \Rightarrow (\mathbf{w}^*)^T\rho_{\mathbf{j}} > (\mathbf{w}^*)^T\rho_{\mathbf{k}}, \quad gdzie\ j < k \end{array} \tag{6}$$

Metodę poszukiwania optymalnego wektora parametrów $\mathbf{w}^*$ (6) można oprzeć na badaniu *liniowej rozdzielności* za pomocą hiperpłaszczyzny przechodzącej przez początek układu współrzędnych przestrzeni cech poniższych dwu zbiorów *wektorów różnicowych* $\mathbf{r_{jk}} = \rho_{\mathbf{k}} - \rho_{\mathbf{j}}$, gdzie $j < k$:

$$\begin{array}{l} R^+ = \{\mathbf{r}_{jk} = \rho_{\mathbf{k}} - \rho_{\mathbf{j}} :\ O_j \prec O_k\} \\ R^- = \{\mathbf{r}_{jk} = \rho_{\mathbf{k}} - \rho_{\mathbf{j}} :\ O_k \prec O_j\} \end{array} \tag{7}$$

Definicja 1. *Zbiory R^+ i R^- są liniowo separowalne za pomocą hiperpłaszczyzny przechodzącej przez początek układu współrzędnych wtedy i tylko wtedy, gdy istnieje taki wektor parametrów $\mathbf{w}'$, że zachodzą poniższe nierówności:*

$$\begin{array}{r} (\forall \mathbf{r}_{jk} \in R^+)\ \mathbf{w}'\mathbf{r}_{jk} > 0 \\ oraz\ (\forall \mathbf{r}_{jk} \in R^-)\ \mathbf{w}'\mathbf{r}_{jk} < 0 \end{array} \tag{8}$$

Wektor parametrów $\mathbf{w}'$ wyznacza hiperpłaszczyznę $H(\mathbf{w}')$ w przestrzeni cech (przestrzeni zależności) przechodzącą przez początek układu współrzędnych tej przestrzeni:

$$H(\mathbf{w}') = \{\mathbf{x} :\ \mathbf{w}'\mathbf{x} = 0,\ \mathbf{x} \in R^m\} \tag{9}$$

O hiperpłaszczyznie $H(\mathbf{w}')$ mówimy, że separuje zbiory R^+ i R^- (7). Wszystkie elementy $\mathbf{r}_{jk}$ zbioru R^+ leżą po dodatniej stronie hiperpłaszczyzny $H(\mathbf{w}')$, a wszystkie elementy $\mathbf{r}_{jk}$ zbioru R^- leżą po jej stronie ujemnej.

Można wykazać słuszność poniższego Lematu [2]:

Lemat 1. *Odwzorowanie liniowe $t = t(\mathbf{w}') = (\mathbf{w}')^T\rho$ zachowuje wszystkie implikacje (6), wtedy i tylko wtedy, gdy hiperpłaszczyzna $H(\mathbf{w}')$ (9) wyznaczona przez wektor $\mathbf{w}'$ separuje zbiory R^+ i R^- (7).*

4. Wypukła i odcinkowo liniowa (*CPL*) funkcja kryterialna $\Phi(\mathbf{w})$

Projektowanie hiperpłaszczyzny $H(\mathbf{w}^*)$ (9) optymalnie rozdzielającej zbiór dodatni R^+ od zbioru ujemnego R^- (7) może być oparte na minimalizacji wypukłej i odcinkowo-liniowej (*CPL*) funkcji kryterialnej $\Phi(\mathbf{w})$, która jest podobna do perceptronowej funkcji kryterialnej [6]. Wprowadźmy w tym celu pozytywne funkcje kary $\varphi^+_{jk}(\mathbf{w})$ określone na elementach $\mathbf{r}_{jk}$ zbioru R^+ oraz negatywne funkcje kary $\varphi^-_{jk}(\mathbf{w})$ określone na elementach $\mathbf{r}_{jk}$ zbioru R^- (7):

$$(\forall \mathbf{r}_{jk} \in R^+) \quad \varphi^+_{jk}(\mathbf{w}) = \begin{cases} 1 - \mathbf{w}^T \mathbf{r}_{jk} & jezeli \quad \mathbf{w}^T \mathbf{r}_{jk} \leq 1 \\ 0 & jezeli \quad \mathbf{w}^T \mathbf{r}_{jk} > 1 \end{cases} \tag{10}$$

oraz

$$(\forall \mathbf{r}_{jk} \in R^-) \quad \varphi^-_{jk}(\mathbf{w}) = \begin{cases} 1 + \mathbf{w}^T \mathbf{r}_{jk} & jezeli \quad \mathbf{w}^T \mathbf{r}_{jk} \geq -1 \\ 0 & jezeli \quad \mathbf{w}^T \mathbf{r}_{jk} < -1 \end{cases} \tag{11}$$

Funkcja kryterialna $\Phi(\mathbf{w})$ jest dodatnio ważoną sumą funkcji kary $\varphi^+_{jk}(\mathbf{w})$ i $\varphi^-_{jk}(\mathbf{w})$, zdefiniowaną następująco:

$$\Phi(\mathbf{w}) = \sum_{(j,k) \in J^+} \gamma_{jk} \varphi^+_{jk}(\mathbf{w}) + \sum_{(j,k) \in J^-} \gamma_{jk} \varphi^-_{jk}(\mathbf{w}) \tag{12}$$

gdzie γ_{jk} jest dodatnim parametrem (*ceną*) związanym z *wektorem różnicowym* $\mathbf{r}_{jk} = \rho_k - \rho_j$, J^+ jest zbiorem par indeksów (j,k) wektorów ze zbioru dodatniego R^+ ($(j,k) \in J^+ \Leftrightarrow \mathbf{r}_{jk} \in R^+$), J^- jest zbiorem par indeksów (j,k) wektorów ze zbioru ujemnego R^- ($(j,k) \in J^- \Leftrightarrow \mathbf{r}_{jk} \in R^-$) (7).

Perceptronowa funkcja kryterialna używana w teorii sieci neuropodobnych i rozpoznawania obrazów ma postać podobną do $\Phi(\mathbf{w})$ (12) [2]. Funkcja kryterialna $\Phi(\mathbf{w})$ (12) jest funkcją wypukłą i odcinkowo-liniową (ang. *convex and piecewise linear - CPL*) jako suma tego typu funkcji kary $\varphi^+_{jk}(\mathbf{w})$ (10) i $\varphi^+_{jk}(\mathbf{w})$ (11). Algorytmy wymiany rozwiązań bazowych, zbliżone do algorytmów programowania liniowego, pozwalają znaleźć minimum funkcji $\Phi(\mathbf{w})$ w sposób efektywny nawet w przypadku dużych, wielowymiarowych zbiorów R^+ i R^- (7) [3]:

$$\Phi^* = \Phi(\mathbf{w}^*) = \min_{\mathbf{w}} \Phi(\mathbf{w}) \geq 0 \tag{13}$$

Optymalny wektor parametrów $\mathbf{w}^*$ oraz wartość minimalna $\Phi(\mathbf{w})^*$ funkcji kryterialnej $\Phi(\mathbf{w})$ (12) mogą być stosowane w rozwiązywaniu wielu problemów eksploracyjnej analizy danych. W szczególności wektor $\mathbf{w}^*$ pozwala wyznaczyć optymalne odwzorowanie rangowe $t = t(\mathbf{w}^*) = (\mathbf{w}^*)^T \rho$ zachowujące wszystkie implikacje rangowe (6) lub ich większość. Można wykazać słuszność poniższego Lematu [2]:

Lemat 2. *Wartość minimalna (13) funkcji kryterialnej (12) jest równa zeru,* $\Phi^* = 0$ *wtedy i tylko wtedy, gdy istnieje taki wektor parametrów* $\mathbf{w}'$*, że wszystkie implikacje rangowe (6) zachowane są przez odwzorowanie liniowe* $t = t(\mathbf{w}') = (\mathbf{w}')^T \rho$ *(5).*

Można zauważyć też, że $\Phi(\mathbf{w}^*) = 0$ wtedy i tylko wtedy, gdy istnieje taka hiperpłaszczyzna $H(\mathbf{w}')$ (9), która w pełni separuje zbiór dodatni R^+ od zbioru ujemnego R^- (7). Jeżeli hiperpłaszczyzna $H(\mathbf{w}^*)$ (9) wyznaczona przez wektor optymalny $\mathbf{w}^*$ (13) nie separuje zbiorów R^+ i R^-, to zbiory te nie są saparowalne przez żadną inną hiperpłaszczyznę $H(\mathbf{w})$ (9). Wartość minimalna $\Phi(\mathbf{w}^*)$ (13) jest wtedy większa od zera ($\Phi(\mathbf{w}^*) > 0$).

5. Wybrane problemy szeregowania zadań obliczeniowych

Wiele problemów szeregowania zadań obliczeniowych O_j może być analizowanych z wykorzystaniem zbioru wektorów zależności ρ_j (1). Załóżmy, że realizowany proces obliczeniowy podzielono (zdekomponowano) na m zadań O_j scharakteryzowanych przez wektory zależności ρ_j (1) oraz czasy obliczeń τ_j:

$$C_0 = \{O_j\}, \qquad gdzie \quad j = 1,...,m \tag{14}$$

Jednym z podstawowych pytań jest to, czy zbiór C_0 zadań obliczeniowych reprezentowanych przez wektory zależności jest niesprzeczny i może być zrealizowany w sposób sekwencyjny? Innymi słowy, czy dany proces obliczeniowy może zostać zrealizowany jako pewna sekwencja zadań O_j (dla $j = 1,...,m$)?

Szukając odpowiedzi na to pytanie, posłużymy się konstrukcją tzw. *warstw funkcjonalnych*. Zerowa warstwa funkcjonalna F_0 jest utworzona przez m_0 takich zadań obliczeniowych O_k ze zbioru C_0 (14), które nie zależą od żadnych innych zadań O_j, tj.:

$$F_0 = \{O_k : \ \rho_k = [0,...,0]^T\} \tag{15}$$

Zakładamy tu, że $m_0 > 0$.

W celu tworzenia kolejnych warstw funkcjonalnych F_n ($n > 0$) zbiór C_0 (14) zostaje zredukowany do zbioru C_1 poprzez pominięcie takich m_0 zadań, które wchodzą do zerowej warstwy funkcjonalnej F_0 (15):

$$C_1 = C_0 / F_0 \tag{16}$$

Redukcja zbioru C_0 do zbioru C_1 oznacza też redukcję tych składowych ρ_{kj} w wektorach zależności ρ_k (1), które odpowiadają pomijanym zadaniom O_j ($O_j \in F_0$). Jeżeli zbiór F_0 zawiera m_0 zadań, to zbiór C_1 zawiera $m - m_0$ zadań scharakteryzowanych

wektorami zależności $\rho[m-m_0]$ o wymiarze $m-m_0$. Jeżeli $m-m_0>0$, to pierwsza warstwa funkcjonalna F_1 zostaje utworzona przez takich m_1 ($m_1>0$) zadań obliczeniowych O_k ze zbioru C_1 (16), które nie zależą od innych zadań O_j z tego zbioru, tj.:

$$F_1=\{O_k:\ O_k\in C_1,\ oraz\ \rho_k[m-m_0]=[0,...,0]^T\} \tag{17}$$

W kolejnym etapie zbiór C_1 (16) zostaje zredukowany do zbioru C_2 poprzez pominięcie takich m_1 zadań O_k, które wchodzą do pierwszej warstwy funkcjonalnej F_1 (17):

$$C_2=C_1/F_1 \tag{18}$$

Podobnie jak poprzednio, redukcji zbioru C_1 zadań O_j (18) do zbioru C_2 towarzyszy redukcja składowych ρ_{kj} odpowiadających pomijanym zadaniom oraz powstanie zredukowanych wektorów zależności $\rho_k[m-m_0-m_1]$ o wymiarze $m-m_0-m_1$. Jeżeli $m-m_0-m_1>0$, to podjęta zostaje próba utworzenia drugiej warstwy funkcjonalnej F_2 w sposób podobny do tworzenia warstwy F_1. W podobny sposób tworzone są następne warstwy funkcjonalne F_n ($n>2$) aż do osiągnięcia etapu, w którym kolejna warstwa funkcjonalna nie może być utworzona.

Dwie przyczyny mogą spowodować, że kolejna warstwa funkcjonalna F_n nie może być utworzona. Po pierwsze, warstwa funkcjonalna F_n nie może być utworzona w sytuacji, gdy zbiór C_n typu (18) jest zbiorem pustym ($C_n=\phi$), ponieważ wszystkie zadania obliczeniowe O_k zostały już zredukowane we wcześniejszych warstwach $F_{n'}$ ($n'<n$). W tym przypadku mamy równość:

$$m-m_0-...-m_{n-1}=0 \tag{19}$$

Jest to naturalne zakończenie procesu tworzenia $n-1$ warstw funkcjonalnych F_i ($i=1,...,n-1$).

Drugim powodem uniemożliwiającym utworzenie warstwy funkcjonalnej F_n może być brak takiego zadania O_k w niepustym zbiorze $C_n=C_{n-1}/F_{n-1}$ (18), dla którego zredukowany wektor zależności $\rho_k[m-m_0-...-m_{n-1}]$ jest równy wektorowi zerowemu $[0,...,0]^T$:

$$(\forall k\in I_n)\quad \rho_k[m-m_0-...-m_{n-1}]\neq[0,...,0]^T \tag{20}$$

gdzie $k\in I_n\Leftrightarrow O_k\in C_n$.

Warunek stopu (20) oznacza, że wszystkie zadania O_k należące do zredukowanego zbioru C_n są ze sobą powiązane. Powstaje pętla, która uniemożliwia wydzielenie zadania odpowiedniego do realizacji. Nie można zrealizować żadnego zadania O_k ze zbioru C_n, ponieważ realizacja każdego z tych zadań wymaga znajomości

wyników pewnego innego zadania z tego zbioru. Podobny problem może się pojawić już w momencie tworzenia zerowej warstwy funkcjonalnej F_0 (15). Warunek uniemożliwiający utworzenie warstwy F_0 ma postać:

$$(\forall O_k \in C_0) \quad \rho_k[m] \neq [0,...,0]^T \tag{21}$$

Warunki typu (20) lub (21) oznaczają pojawienie się pętli w grafie zależności. Pojawienie się pętli w zbiorze C_n (18) oznacza, że zbiór zadań O_k z tego zbioru, reprezentowanych za pomocą zredukowanych wektorów zależności $\rho_k[m-m_0-...-m_{n-1}]$, jest sprzeczny i nie może być zrealizowany. Realizacja każdego zadania O_k tworzącego pętlę jest niemożliwa, ponieważ wymaga znajomości wyników pewnego innego zadania O_j ze zbioru C_n.

Algorytm 1 Algorytm pozwalający stwierdzić sprzeczność lub niesprzeczność zbioru zadań obliczenowych

1: $C_0 := \{0_j\}\ j = 1,...,m$
2: $I_0 := \{j\}\ j = 1,...,m$
3: $n := 0$
4: **if** $C_n = \phi$ **then**
5: KONIEC: zbiór zadań jest niesprzeczny
6: **end if**
7: **if** $(\forall k \in I_n)\ \rho_{\mathbf{k}} \neq [0,...,0]^T$ **then**
8: KONIEC: zbiór zadań jest sprzeczny
9: **end if**
10: $F_n := \{O_k : O_k \in C_n\ oraz\ \rho_{\mathbf{k}} = [0,...,0]^T$
11: $C_{n+1} := C_n / F_n$
12: $I_{n+1} := I_n / \{k : O_k \in F_n\}$
13: zredukuj składowe ρ_{kj} w wektorach zależności $\rho_{\mathbf{k}}$ $(O_k \in C_{n+1})$ odpowiadające pomijanym zadaniom O_j $(O_j \in F_n)$
14: $n := n+1$
15: GOTO 4

Zauważmy, że przynależność zadań O_j i O_k do tej samej warstwy funkcjonalnej F_n wyklucza istnienie zależności $O_j \prec O_k$ (2) lub $O_k \prec O_j$ pomiędzy tymi zadaniami. Równoległa realizacja zadań obliczeniowych należących O_j do tej samej warstwy funkcjonalnej F_n daje możliwość skrócenia czasu realizacji całego procesu obliczeniowego.

Lemat 3. *Jeżeli zadania O_j i O_k należą do tej samej warstwy funkcjonalnej F_n, to nie może istnieć zależność $O_j \prec O_k$ (2) pomiędzy tymi zadaniami.*

Słuszność lematu 3 wynika bezpośrednio z opisanej powyżej konstrukcji warstw funkcjonalnych F_n, gdzie każde zadanie O_k danej warstwy jest scharakteryzowane zredukowanym wektorem zależności ρ_k o wartości zerowej ($\rho_k = [0,...,0]^T$. Podobnie można uzasadnić poniższą własność:

$$(\forall O_{k'} \in F_{n'})\,(\forall O_k \in F_n) \quad (O_{k'} \prec O_k \Rightarrow n' < n) \tag{22}$$

Zgodnie z powyższą własnością, istnienie zależności $O_{k'} \prec O_k$ w strukturze warstw funkcjonalnych $F_{n'}$ i F_n wyznacza taki porządek, że warstwa F_n znajduje się wyżej niż warstwa $F_{n'}$. Zadanie O_k z warstwy wyższej nie może poprzedzać zadania $O_{k'}$ z warstwy niższej.

Lemat 4. *Zadania obliczeniowe ze zbioru C_0 (14) mogą być zrealizowane wtedy i tylko wtedy, gdy podczas wydzielania warstw funkcjonalnych F_n nie zachodzą warunki typu (20) lub (21) wskazujące na „zapętlenie się" procesu obliczeniowego.*

Istnienie pętli w zbiorze C_0 (14) m zadań obliczeniowych ma związek z minimalną wartością $\Phi(\mathbf{w}^*[m])$ (13) funkcji kryterialnej $\Phi(\mathbf{w}[m])$ (12):

Twierdzenie 1. *Zbiór C_0 opisuje niesprzeczny zestaw m zadań obliczeniowych O_k wtedy i tylko wtedy, gdy wartość minimalna $\Phi(\mathbf{w}^*[m])$ funkcji kryterialnej $\Phi(\mathbf{w}[m])$ określonej na bazie wektorów zależności $\rho_k[m]$ (1) reprezentujących te zadania jest równa zeru ($\Phi(\mathbf{w}^*[m]) = 0$).*

Dowód: W dowodzie posłużymy się opisaną powyżej konstrukcją warstw funkcjonalnych F_n. Załóżmy, że w trakcie konstrukcji nie są spełnone warunki typu (20) lub (21). W tym przypadku wszystkie zadania obliczeniowe ze zbioru C_0 (14) mogą być rozłożone w warstwy funkcjonalne F_n. W rezultacie wszystkie zadania mogą być zrealizowane w sposób sekwencyjny, zgodny z podziałem na warstwy F_n.

Dla przeprowadzenia dowodu dokonamy indeksowania zadań obliczeniowych O_j ze zbioru C_0 zgodnie z podziałem na warstwy funkcjonalne F_n (*indeksowanie kanoniczne*). Przyjmiemy, że m_0 zadań O_j z warstwy F_0 ma najniższe indeksy j ($1 \le j \le m_0$), m_1 zadań O_j z warstwy F_1 ma indeksy j z przedziału ($m_0 + 1 \le j \le m_0 + m_1$), itd. Przy takim sposobie indeksowania zadań obliczeniowych O_j wektory zależności $\rho_j = [\rho_{j1},...,\rho_{jm}]^T$ o m składowych binarnych ρ_{ji} mają następującą strukturę: m_0 pierwszych wektorów ρ_j ($1 \le j \le m_0$) ma wszystkie m składowych ρ_{ji} równe zeru ($(\forall j \in \{1,...,m_0\})\,(\forall i \in \{1,...,m\})\,\rho_{ji} = 0$). Kolejne m_1 wektorów ρ_j, odpowiadających zadaniom O_j z warstwy F_1, ma składowe ρ_{ji} o indeksach i większych niż m_0 równe zeru ($(\forall j \in \{m_0+1,...,m_0+m_1\})\,(\forall i \in \{m_0+1,...,m\})\,\rho_{ji} = 0$). Podobnie m_2 wektorów ρ_j odpowiadających zadaniom O_j

z warstwy F_2 ma składowe ρ_{ji} o indeksach i większych niż m_0+m_1 równe zeru $((\forall j \in \{m_0+m_1+1,...,m_0+m_1+m_2\})\,(\forall i \in \{m_0+m_1+1,...,m\})\,\rho_{ji}=0)$. Podobną strukturę mają wektory zależności ρ_j z kolejnych wyższych warstw. Rozumowanie to jest oparte na właściwości określonej w relacji (22).

Weźmy wektor parametrów $\mathbf{w}'[m]$ o poniższej strukturze:

$$\mathbf{w}'[m] = [c_1,...,c_1,c_2,...,c_2,...,c_n,...,c_n]^T \tag{23}$$

Pierwsze m_0 składowych wektora $\mathbf{w}'[m]$ ma wartość c_1 ($c_1 > 0$, np. $c_1 = 1$) i odpowiada zadaniom z warstwy F_0 (15). Kolejne m_1 składowych wektora $\mathbf{w}'[m]$ ma wartość c_2 ($c_2 > c_1$) itd. Ostatnie m_n składowych wektora $\mathbf{w}'[m]$ ma wartość c_n ($c_n > c_{n-1}$).

Parametry c_i (23) można wybrać tak, by wszystkie implikacje rangowe (6) były zachowane przez odwzorowanie liniowe $t_j = (\mathbf{w}'[m])^T \rho_j[m]$. Np. odpowiednio duża wartość parametru c_k odpowiadającego warstwie F_k zapewnia, że wszystkie zależności $O_{k'} \prec O_k$, gdzie $k' < k$, są reprezentowane przez nierówności (6): $(\mathbf{w}'[m])^T \rho_{k'}[m] < (\mathbf{w}'[m])^T \rho_k[m]$.

Jak wynika z lematu 2, wartość $\Phi(\mathbf{w}^*[m])$ (13) jest równa zeru ($\Phi(\mathbf{w}^*[m] = 0)$ wtedy i tylko wtedy, gdy wszystkie implikacje rangowe (6) są zachowane przez odwzorowanie liniowe $t_j = (\mathbf{w}'[m])^T \rho_j[m]$.

Twierdzenie 1 wskazuje możliwość badania niesprzeczności zestawu zadań obliczeniowych O_j (15) schrakteryzowanych przez wektory zależności $\rho_j[m]$ (1) na podstawie sprawdzenia warunku, czy wartość minimalna $\Phi(\mathbf{w}^*[m])$ (13) funkcji kryterialnej $\Phi(\mathbf{w}[m])$ (12) jest równa zeru.

6. Możliwości wykorzystania modeli rangowych w modyfikacji zadań obliczeniowych

Znajomość zestawu m zadań obliczeniowych (15) scharakteryzowanych przez wektory zależności $\rho_j[m]$ (1) pozwala zdefiniować zbiory $R^+[m]$ i $R^-[m]$ (7) złożone z wektorów różnicowych $\mathbf{r}_{jk}[m] = \rho_k[m] - \rho_j[m]$, a następnie funkcję kryterialną $\Phi(\mathbf{w}[m])$ (16). Minimalizacja funkcji kryterialnej $\Phi(\mathbf{w}[m])$ daje odpowiedź na pytanie czy zestaw zadań jest niesprzeczny (Tw. 1). Zestaw m zadań obliczeniowych scharakteryzowanych przez wektory zależności $\rho_j[m]$ (1) jest sprzeczny wtedy i tylko wtedy, gdy wartość minimalna $\Phi(\mathbf{w}^*[m])$ (13) jest większa od zera ($\Phi(\mathbf{w}^*[m]) > 0$). Ma to miejsce w sytuacji, gdy zbiory $R^+[m]$ i $R^-[m]$ (7) nie są liniowo separowalne (8). W takim przypadku część wektorów różnicowych $\mathbf{r}_{jk}[m] = \rho_k[m] - \rho_j[m]$ usytuowana jest po niewłaściwej stronie hiperpłaszczyzny $H(\mathbf{w}^*[m])$ (9) wyznaczonej w m-wymiarowej przestrzeni zależności przez wektor optymalny $\mathbf{w}^*[m]$ (13).

Część wektorów $\mathbf{r}_{jk}[m]$ należących do zbioru $R^+[m]$ usytuowana jest niewłaściwie po ujemnej stronie hiperpłaszczyzny $H(\mathbf{w}^*[m])$ ($\mathbf{w}^*[m]\mathbf{r}_{jk} < 0$). Istnieją także takie wektory $\mathbf{r}_{jk}[m]$ ze zbioru $R^-[m]$, które są niewłaściwie usytuowane po dodatniej stronie $H(\mathbf{w}^*[m])$ ($\mathbf{w}^*[m]\mathbf{r}_{jk} > 0$). W problemach rozpoznawania obrazów znany jest fakt, że dwa zbiory liniowo nieseparowalne można doprowadzić do liniowej separowalności poprzez redukcję przynajmniej części elemetów niewłaściwie usytuowanych względem hiperpłaszczyzny optymalnej $H(\mathbf{w}^*[m])$ [5]. Podobne techniki redukcji można stosować w szeregowaniu zadań obliczeniowych.

Zgodnie z opisaną wcześniej wieloetapową procedurą wydzielana jest warstwa funkcjonalna F_0 (15) a następnie kolejne warstwy F_n. Wydzielenie kolejnej warstwy F_n wiąże się z redukcją $C_n = C_{n-1}/F_{n-1}$ (18) zbioru zadań i sprawdzaniu czy zachodzi warunek stopu (20). Warunek stopu w warstwie zerowej ma podobną postać (21). Warunek stopu wskazuje, że żadne z m zadań O_k ze zbioru C_n nie może być zrealizowane, ponieważ wymaga to znajomości wyników pewnego innego zadania ze zbioru C_n. Sposobem na przerwanie tego typu wzjemnych zależności może być taka modyfikacja jednego z zadań O_k ze zbioru C_n, by jego wykonanie nie zależało od żadnego innego zadania O_j ze zbioru C_n. Zredukowany wektor zależności $\rho_k[m]$ zmodyfikowanego zadania O'_k powinien być równy wektorowi zerowemu $[0,...,0]^T$. Modyfikacja zadania O_k prowadząca do zerowania się wektora zależności $\rho_k[m']$ jest na ogół możliwa, ale może wymagać przedefiniowania nie tylko jednego, lecz większej liczby zadań obliczeniowych, co może być obarczone dużymi kosztami.

W poszukiwaniu najbardziej efektywnego sposobu modyfikacji zadań O_k ze sprzecznego zbioru C_n można posłużyć się minimalną wartością $\Phi(\mathbf{w}^*[m])$ (13) funkcji kryterialnej $\Phi(\mathbf{w}[m])$ (12), określonej na bazie wektorów zależności $\rho_k[m]$ (1) reprezentujących zadania O_k ze zbioru C_n. Bierzemy tu pod uwagę taką modyfikację zadań $O_k \rightarrow O'_k$, którym towarzyszy zerowanie się zredukowanych wektorów zależności $\rho_k[m]$:

$$\rho_k[m] \rightarrow [0,...,0]^T \tag{24}$$

Zauważmy, że jeżeli modyfikowanie zadania O_k ze sprzecznego zbioru C_n następuje po wcześniejszym wydzieleniu pewnych warstw funkcjonalnych F_n, to warunek (24) nie ogranicza zależności zadania O_k od innych zadań z tych warswtw.

Wybór zadania O_k do modyfikacji można oprzeć na kryterium maksymalnego spadku $\triangle_k\Phi(\mathbf{w}^*[m])$ minimalnej wartości $\Phi(\mathbf{w}^*[m])$ (13) funkcji kryterialnej $\Phi(\mathbf{w}[m])$ (12) w wyniku modyfikacji (24) wektora zależności $\rho_k[m]$ (1) zadania O_k:

$$\triangle_k\Phi(\mathbf{w}^*[m]) = \max_{j\in I_n} \triangle_j\Phi(\mathbf{w}^*[m]) \tag{25}$$

gdzie $j \in I_n \Leftrightarrow O_j \in C_n$

Zgodnie z powyższym kryterium modyfikowania, do modyfikacji wybieramy takie zadanie, które daje największy spadek minimalnej wartości funkcji kryterialnej określonej na zredukowanym zbiorze C_n.

Można założyć, że obliczanie potencjalnych spadków $\triangle_j\Phi(\mathbf{w}^*[m])$ minimalnej wartości $\Phi(\mathbf{w}^*[m])$ (13) funkcji kryterialnej $\Phi(\mathbf{w}[m])$ (12) jest stosunkowo tanie w stosunku do kosztów δ_j ($\delta_j > 0$) rzeczywistej modyfikacji poszczególnych zadań obliczeniowych O_j.

Jeżeli znane są koszty δ_j modyfikacji poszczególnych zadań obliczeniowych O_j, to kryterium wyboru zadania O_k do modyfikacji może uwzględniać także wielkości δ_j:

$$\triangle_k\Phi(\mathbf{w}^*[m])/\delta_k = \max_{O_j \in C_n} \triangle_j\Phi(\mathbf{w}^*[m])/\delta_j \tag{26}$$

Opisana powyżej procedura pozwala usuwać sprzeczności z zestawów zadań obliczeniowych O_j poprzez modyfikację wybranych zadań O_k.

Innym ważnym w praktyce problemem jest zagadnienie redukcji niepotrzebych zadań obliczeniowych O_j tworzących zbiór C_0 (14). W pewnych przypadkach możemy być zainteresowani w jak najszybszym uzyskaniu wyniku określonego zadania O_k. Uzyskanie wyniku danego zadania O_k wymaga wykonania pewnych zadań O_j, lecz wykonywanie innych zadań ze zbioru C_0 może nie być konieczne. Zidentyfikowanie takich niekoniecznych zadań O_j pozwala je pominąć i w rezultacie skrócić czas obliczeń.

Załóżmy, że zbiór C_0 zadań O_j scharakteryzowych przez wektory zależności ρ_j (1) i czasy realizacji τ_j jest niesprzeczny. W tym przypadku wszystkie zadania O_j mogą być przyporządkowane poszczególnym warstwom funkcjonalnym F_n (19). Dogodnie jest posługiwać się wtedy indeksowaniem kanonicznym opisanym w dowodzie Twierdzenia 1. Redukcja zadań O_j oznacza też zmniejszanie wymiaru wektorów zależności $\rho_j[m]$ (1).

Jeżeli zadania O_j zostały przyporządkowane poszczególnym warstwom funkcjonalnym F_n i wybrane zadanie O_k znalazło się w warstwie $F_{n'}$, to wszystkie zadania z warstw wyższych F_n ($n > n'$), jak również różne od O_k zadania z warstwy $F_{n'}$ mogą zostać zredukowane (pominięte). W rezultacie takiej redukcji powstaje zbiór zredukowany C_0'. Zbiór C_0' może być w pewnych okolicznościach dodatkowo zredukowany. W tym celu sprawdzamy czy w zbiorze C_0' znajdzie się takie zadanie $O_{j'}$, które nie poprzedza żadnego innego zadania z tego zbioru. Jeżeli takie zadanie zostanie znalezione, to zadanie to usuwamy ze zbioru C_0' i powtórnie poszukujemy w ostatnio zredukowanym zbiorze C_0' zadania $O_{j''}$, które nie poprzedza żadnego innego zadania z tego zbioru. Jeżeli zadanie $O_{j''}$ o takiej właściowości bdzie znalezione, to usuwamy

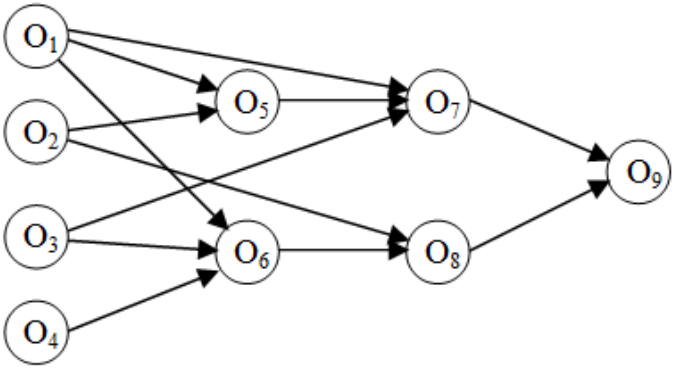
je ze zbioru C_0'. Poszukiwania niekoniecznych zadań O_j i ich redukcje powtarzamy do momentu, gdy w zbiorze C_0' nie będzie elementów nadających się do redukcji.

7. Wyniki eksperymentów

Przy zastosowaniu opisanych wczesniej metod wykonane zostały dwa eksperymenty, sprawdzające w praktyce słuszność zaproponowanych rozwiązań teoretycznych. W obu eksperymentach użyto sztucznie wygenerowanych sekwencji zadań obliczeniowych.

7.1 Eksperyment 1 - Sekwencja niesprzeczna

W pierwszym eksperymencie sekwencja zadań obliczeniowych bya dobrana tak, aby zdekomponowany proces obliczeniowy był niesprzeczny, czyli aby nie występowały w nim zapętlenia. Rysunek 1 pokazuje kolejność wykonywania zadań. Strzałka od zadania O_j do zadania O_k wskazuje na zależność zadania O_k od zadania O_j. Pomiędzy zadaniami O_j i O_k zachodzi relacja rangowa $O_j \prec O_k$.



Rys. 1. Sekwencja zadań użytych w eksperymencie 1

Otrzymano następujące wektory zależności ρ_j ($j = \{1,...,9\}$):

$$\begin{aligned}
\rho_1 &= [0,0,0,0,0,0,0,0,0]^T \\
\rho_2 &= [0,0,0,0,0,0,0,0,0]^T \\
\rho_3 &= [0,0,0,0,0,0,0,0,0]^T \\
\rho_4 &= [0,0,0,0,0,0,0,0,0]^T \\
\rho_5 &= [1,1,0,0,0,0,0,0,0]^T \\
\rho_6 &= [1,0,1,1,0,0,0,0,0]^T \\
\rho_7 &= [1,0,1,0,1,0,0,0,0]^T \\
\rho_8 &= [0,1,0,0,0,1,0,0,0]^T \\
\rho_9 &= [0,0,0,0,0,0,1,1,0]^T
\end{aligned} \tag{27}$$

Na podstawie relacji rangowych pomiędzy zadaniami i ich wektorów zależności wyznaczono zbiory R^+ i R^- (7) wektorów różnicowych $\mathbf{r}_{jk}$:

$$\begin{aligned}
\mathbf{r}_{1,5} &= [1,1,0,0,0,0,0,0,0]^T \\
\mathbf{r}_{1,6} &= [1,0,1,1,0,0,0,0,0]^T \\
\mathbf{r}_{1,7} &= [1,0,1,0,1,0,0,0,0]^T \\
\mathbf{r}_{2,5} &= [1,1,0,0,0,0,0,0,0]^T \\
\mathbf{r}_{2,8} &= [0,1,0,0,0,1,0,0,0]^T \\
\mathbf{r}_{3,6} &= [1,0,1,1,0,0,0,0,0]^T \\
\mathbf{r}_{3,7} &= [1,0,1,0,1,0,0,0,0]^T \\
\mathbf{r}_{4,6} &= [1,0,1,1,0,0,0,0,0]^T \\
\mathbf{r}_{5,7} &= [0,-1,1,0,1,0,0,0,0]^T \\
\mathbf{r}_{6,8} &= [-1,1,-1,-1,0,1,0,0,0]^T \\
\mathbf{r}_{7,9} &= [-1,0,-1,0,-1,0,1,1,0]^T \\
\mathbf{r}_{8,9} &= [0,-1,0,0,0,-1,1,1,0]^T
\end{aligned} \tag{28}$$

$$\begin{aligned}
R^+ &= \{\mathbf{r}_{1,5},\mathbf{r}_{1,6},\mathbf{r}_{1,7},\mathbf{r}_{2,5},\mathbf{r}_{2,8},\mathbf{r}_{3,6},\mathbf{r}_{3,7},\mathbf{r}_{4,6},\mathbf{r}_{5,7},\mathbf{r}_{6,8},\mathbf{r}_{7,9},\mathbf{r}_{8,9}\} \\
R^- &= \phi
\end{aligned}$$

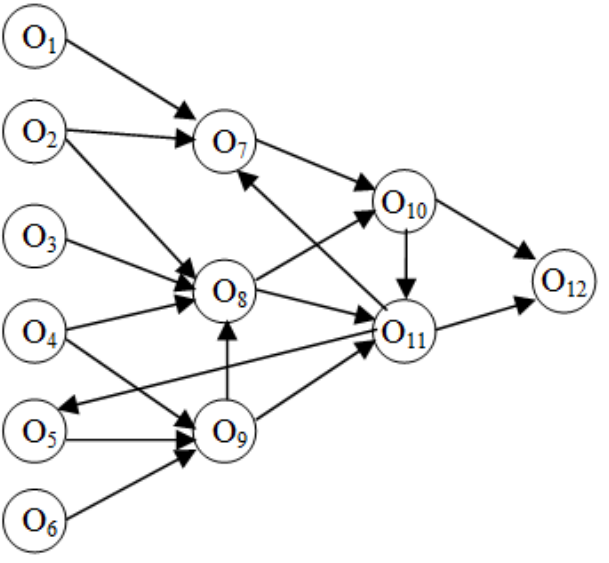
Poprzez minimalizację wartości funkcji kryterialnej $\Phi(\mathbf{w})$ (12) wyznaczono optymalny wektor parametrów $\mathbf{w}^*$:

$$\mathbf{w}^* = [-1,0,-1,0,0,0,-3,-4,0,0]^T \tag{29}$$

Wartość minimalna funkcji kryterialnej $\Phi(\mathbf{w}^*)$ (13) przyjmuje wartość zero, więc zgodnie z twierdzeniem 1 zestaw zadań jest niesprzeczny i może być realizowany w założonej sekwencji.

7.2 Eksperyment 2 - Sekwencja sprzeczna

Drugi eksperyment był identyczny z pierwszym. Została w nim jedynie użyta sekwencja zadań, w której występują pętle.



Rys. 2. Sekwencja zadań użytych w eksperymencie 2

Wyznaczone zostały wektory zależności ρ_j ($j = \{1,...,12\}$) i zbiory R^+ i R^- (7) wektorów różnicowych $\mathbf{r}_{jk}$.

$$\begin{aligned} R^+ &= \{\mathbf{r}_{1,7}, \mathbf{r}_{2,7}, \mathbf{r}_{2,8}, \mathbf{r}_{3,8}, \mathbf{r}_{4,8}, \mathbf{r}_{4,9}, \mathbf{r}_{5,9}, \mathbf{r}_{6,9}, \mathbf{r}_{7,10}, \mathbf{r}_{8,10}, \mathbf{r}_{8,11}, \mathbf{r}_{9,11}, \mathbf{r}_{10,11}, \mathbf{r}_{10,12}, \mathbf{r}_{11,12}\} \\ R^- &= \{\mathbf{r}_{5,11}, \mathbf{r}_{7,11}, \mathbf{r}_{8,9}\} \end{aligned} \tag{30}$$

Poprzez minimalizację wartości funkcji kryterialnej $\Phi(\mathbf{w})$ (12) wyznaczono optymalny wektor parametrów $\mathbf{w}^*$:

$$\mathbf{w}^* = [0,0,0,0,-1,0,-3,0,-2,0,-3.5,0]^T \tag{31}$$

Wartość minimalna funkcji kryterialnej $\Phi(\mathbf{w}^*)$ (13) przyjmuje wartość 0.250367, więc zgodnie z przewidywaniami otrzymano odpowiedź, że sekwencja zadań nie może być zrealizowana w takiej kolejności. Stosując mechanizm opisany w rozdziale 6 można wyznaczyć zadanie do modyfikacji i doprowadzić do niesprzeczności sekwencji.

8. Uwagi końcowe

Regresja rangowa może być użytecznym narzędziem analizy dużych systemów obliczeniowych. Budowa modeli rangowych do celów analizy systemów obliczeniowych opiera się na reprezentacji zadań obliczeniowych O_j za pomocą wektorów zależności ρ_j (1), które opisują wzajemne zależności $O_j \prec O_k$ (2) pomiędzy tymi zadaniami. Modele rangowe buduje się w formie takich odwzorowań liniowych z wielowymiarowej przestrzeni zależności na przestrzeń jednowymairową, które w maksymalnym stopniu zachowują (6) te zależności.

Konstrukcja odwzorowań rangowych opiera się na wyznaczaniu hiperpłaszczyzny $H(\mathbf{w}')$ (9) oddzielającej w możliwie najlepszy sposób dwa zbiory R^+ i R^- (7). Metoda wyznaczania hiperpłaszczyzny $H(\mathbf{w}')$ (9) bazuje na minimalizacji wypukłej i odcinkowo-liniowej (ang. *convex and piecewise linear - CPL*) funkcji kryterialnej $\Phi(\mathbf{w})$ (12). Zostało wykazane (Tw. 1), że wartość minimalna $\Phi(\mathbf{w}^*[m])$ (13) funkcji kryterialnej $\Phi(\mathbf{w}[m])$ (12) określonej na bazie m wektorów zależności ρ_j (1) reprezentujących zadania obliczeniowe O_j jest równa zeru ($\Phi(\mathbf{w}^*[m]) = 0$) wtedy i tylko wtedy, gdy zbiór C_0 (14) zawiera niesprzeczny zestaw zadań.

Minimalizacja funkcji kryterialnej $\Phi(\mathbf{w}[m])$ (12) pozwala nie tylko wykrywać sprzeczności w zestawach zadań obliczeniowych O_j, ale może być również użyteczna w wybieraniu zadań do modyfikacji mających na celu efekywne usunięcie tych sprzeczności.

Literatura

[1] Bobrowski, L.: Modele szeregowania zadań obliczeniowych wykorzystujące regresję rangową, „Symulacja w Badaniach i Rozwoju", PTSK, Warszawa, 2007.

[2] Bobrowski, L.: Linear ranked regression - designing principles, CORES05, IV International Conference on Computer Recognition Systems, Advances in Soft Computing, Springer, Berlin, 2005.

[3] Bobrowski, L., Niemiro, W.: A method of synthesis of linear discriminant function in the case of nonseparabilty, Pattern Recognition 17, pp.205-210, 1984.

[4] Błażewicz, J., Ecker, K., Pesh, E., Schmidt, G., Węglarz, J.: Scheduling Computer and Manufacturing Process, Springer, Berlin, 1966.

[5] Duda, O.R., Heart, P.E., Stork, D.G.: Pattern Classification, Wydanie drugie, zmienione, John Wiley & Sons, 2001.

[6] Janiak, A. (Ed.): Scheduling in computer and manufacturing systems, WKiŁ, 2006.

[7] Smutnicki, C.: Algorytmy szeregowania Exit 2002.

SCHEDULING BASED ON RANKED REGRESSION MODELS

Abstract: The issues of scheduling of tasks are found, among other things, in connection with the problems of realizeable of big computing processes and optimisation of them. The ranked regresion methods can be used to determine of this kind of problems. Separate computing tasks are characterized by multidimensional vectors of dependences in order to form the ranked regresion models . The vectors of dependences allow to state whether particular task can be realised only when certain other tasks have realised before. The ranked regresion includes the designing of such linear transformations from the multidimensional space of dependences to unidimensional space (time line), which reflect the dependences beetwen task as well as possible.

Keywords: scheduling of the computing tasks, ranked model, convex and piecewise linear (CPL) criterion functions

Praca wspierana częściowo przez projekt KBN 3T11F011 30, projekty W/WI/1/2008 i S/WI/2/2008 Politechniki Białostockiej.

Cezary Bołdak[1], Marcin Sadowski[1], Jerzy Jaroszewicz[2]

SEMI-AUTOMATIC COUNT OF HEPATIC STELLATE CELLS FROM MICROSCOPIC IN-VITRO IMAGES BY MODELING ELLIPTICAL NUCLEI

Abstract: A new method of semi-automatic liver cell counting is presented. Instead of segmenting the cells bodies (not regular and fragmented in some stages of the cells life) it localizes the cells nuclei which are bright, homogeneous and elliptical structures with darker body (body fragments) on their circumference. The nuclei are modeled by ellipses which can be found in two manners: by local region growing algorithm and by reconstruction of the ellipse equation from its contour points. The found ellipses set is then downsized (since all possible ellipses are initially considered) by eliminating the closest one to another and the worst ones by mean of a special fitness function. The method is implemented as a visual, multiplatform JAVA application, easy to use in the scientific every-day work. It is evaluated on real microscopic in-vitro images of the hepatic stellate cells.

Keywords: medical image processing, cell count, hepatic stellate cells

1. Introduction

In the modern medicine different techniques of imaging play very important role. Thanks to other than optical acquisition modalities (X-rays, ultrasounds, magnetic fields, infrared rays, etc.) they offer a non-invasive visualisation of internal body structures and even of body processes (e.g. fMRI). Images recording helps in the physicians' education, exchanging of knowledge between them and in temporal analysis of disease development or treatment efficacy.

On the other hand, the physician can get overwhelmed with the quantity, dimensions and complexity of such images. All this visual information (or its major part) would be useless without efficient and pertinent methods of image processing. Starting from sometimes trivial image quality enhancement (contrast, noise removing), through multimodal images registration, compression, tranmittion and visualization,

[1] Faculty of Computer Science, Balystok Technical University, Białystok, Poland
[2] Department of Infectious Diseases, Medical University of Biaşystok, Białystok, Poland

up to sofisticated segmenting of interesting structures from images and characterizing them[10], modern computer-aided tools can improve the human work or at least make it less tedious.

An example addressing the latter issue is a problem of object counting, where the goal is to give exact or approximative quantity of some (similar) objects present in the image. This work, quite common in the medicine, seems very easy to the human, but when the object number increases, it becomes tedious and error-prone, and this is why there have been many attemps to make it automatic. [5] and [12] count (segments) spermatozoides, [2,6,8,9] - blood cells, [1] - survived cells in photodynamic therapy, [11] - liver cells, [13] - neural stem cells, [7] - neutrophils and lymphocytes.

Chronic liver injury results in hepatocyte damage, which triggers activation of hepatic stellate cells (HSC) [4]. HSC play a central role in liver fibrosis development. Following the stimulation, those cells undergo phenotypic transdifferentiation to myofibrolasts. Along with morphological transformation they became major source of collagen type I in the liver and secrete pro-fibrogenic cytokines and inhibitors of matrix-degrading enzymes (tissue inhibitor of matrix metalloproteinase). This causes the increased extracellular matrix deposition over degradation, which eventually leads to liver cirrhosis development. Thus, HSC represent an appealing target for antifibrotic therapy.

This article is organized as follows. Section 2. gives a review of a few automatic counting methods from the bibliography. Section 3. describes 2 proposed algorithms for liver cells counting. Section 4. contains a short description of an application implementing our method. Experiments with real microscopic images of in-vitro liver cells using our 2 algorithms compared to a simple region growing technique and to an expert counting are presented in section 5.

2. Background

Numerous works start the counting with a preprocessing of the treated image. The noise can be removed by a low-pass filtering [1,9]. Very common technique used here is the thresholding [12], sometimes multilevel [8], adaptive [11] or local [1]. The objects boundaries can be also discovered here by an edge detector [1,9]. All this is very often followed by an extra operation improving the resulting binary image, like the mathematical morphology opening/closing [8,9,11].

The objects to count can then be identified as groups of connected binary pixels [8,9,11]. The watershed segmentation can also isolate the objects from the background [1]. If the image is not converted to the binary one, other sophisticated techniques can be applied to segment the object contours [6]. [13] uses Voronoi diagrams to se-

parate regions occupied by the stem cells. [7] combines 2 techniques: region-based (using basic texture features) and contour -based (cells boundaries detected as high-gradient structures) into one hybrid method to segment different cells.

At the end, these results can undergo a post-treatment phase. Among its goals one can mention: elimination of groups not satisfying some conditions (too small size [1,11], geometrical center located outside the object [1,11], shape contour not accepted by a neural network trained on desired shapes [6], presence of background pixels inside the object [8]), dividing groups into several distinct objects, touching one another (basing on too big size [11] or radius variation [8]), incorporating close, distinct fragments to the main object (e.g. basing on the Hough transform [8]).

Very exhaustive review of application of different techniques to the count problem is given in [5].

3. Method

As shown at Figures 1 the cells in different stage of life can have their bodies fragmented what makes segmenting them using algorithms based on a simple pixel grouping (e.g. region growing) impossible. Finding a model can also be a hard task because of their varied shapes. However, we have noticed there is one stable feature in these cells - their nuclei. They have in the most of cases an elliptical form, homogeneous inside and bordered by the cell body or their fragments. This is why we have focused on them and proposed a method counting regions with the given above characteristics.

Our method consists of three steps. In the first one the image is preprocessed to get a binary representation of pixels belonging to the background and to the segmented cells. In the second step we construct ellipses - candidates for the cells nuclei. Two approaches are considered - one based on the region growing and one approximating the ellipse equation from a set of border points. These candidate ellipses are far too numerous and in the final, third step the best local ones are chosen following a special fitness function.

Images treated in this work should be gray level ones - color information is not exploited here.

3.1 Preprocessing

The main goal of this step is to produce a binary image with one pixel set representing the background and the cells nuclei and the second set representing the cells bodies. The latter structures are darker in the processed images so a simple thresholding

could do this work. The first problem are variations in the cells bodies gray level because of their grain structure (Figure 1(a)). To smooth the levels we blur the images by convolving them with the Gaussian filter mask [10] (Figure 1(b)). The second problem is probably with the image acquisition process. The scanned images are sometimes not properly exposed and some their regions can be brighter than others (Figure 1(c)). One global threshold can not be applied here. To properly differentiate the cells pixels from the background we used an adaptive threshold technique where each pixel has its own threshold value calculated locally. It increases the calculation time but allows appropriate handling of real images. Each local threshold $T(x,y)$ is computed in a local square neighborhood of the point (x,y) as a mean gray level in it, decreased by a positive constant C - parameter of this technique. This constant can be interactively adjusted by the operator but in all the cases here its value was equal 5.

$$T(x,y) = \left(\frac{1}{N^2} \sum_{i=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \sum_{j=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} I(x+i, y+j) \right) - C \tag{1}$$

where N is a size of square local neighborhood (equal 21 in all our experiments), $\lfloor . \rfloor$ - floor operation, and $I(x,y)$ gray level of the image in location (x,y). To improve the effectiveness of this stage, the above adaptive threshold is calculated only for a limited set of pixels and linearly interpolated for the rest of them. Experiments have shown that calculating every $\lfloor N/2 \rfloor$ pixel in x and y dimensions preserves the thresholding accuracy and significantly reduces the calculation time. So the binary image BI is constructed following the rule below:

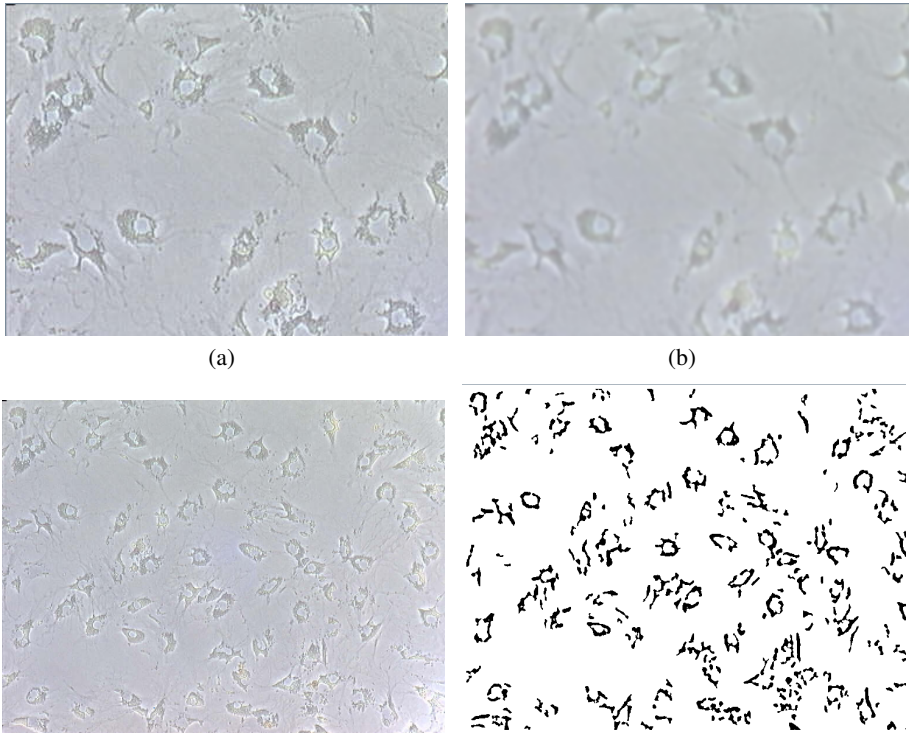
$$BI(x,y) = \begin{cases} 'cell' & \text{if} \quad I(x,y) < T(x,y), \\ 'background' & \text{if} \quad I(x,y) \geq T(x,y). \end{cases} \tag{2}$$

One can observe that the binary contours (Figure 1(d)) not always closely follow the real ones (Figure 1(c)) - they can be narrower, but this slight change does not influence the elliptical form of the nuclei. It can be stated very clearly here - the main interest of this work is to count the cells, not to segment their real shape.

At the end of this stage all connected groups which size is below some interactively chosen value are removed from the BI - they the most likely are artifacts.

3.2 Construction of ellipses

We propose 2 different techniques to construct ellipses representing the cells nuclei. The first one, based on the region growing algorithm, should be faster when applied to smoother images, with low noise level, but its performance can be disturbed by



(a) (b)

(c) (d)

Fig. 1. Analysed image (a) A fragment showing the grained structure of the cells bodies (b) Result of the Gaussian blurring removing the grained structure (c) Entire image showing the mean gray level changing smoothly (d) Result of the adaptive thresholding (after removing the smallest marked groups)

even one noise pixel inside the cell nucleus. The second one can restore an ellipse equation even in presence of several noise pixels but is more complex.

In two mentioned above solutions the entire image image is traversed horizontally and vertically to examine all potential locations of the cells nuclei centers. It does not necessarily demand visiting all pixels in the image - we have observed that using both vertical and horizontal step of traversing equal 2 does not influence at all the performance of the count and significantly accelerate the process. Further increasing of this step can be possible but requires deeper analysis in function of the cells dimensions.

Both technique use one parameter R_{max} - the maximal radius of the cell nucleus, given visually by the operator according the the treated image.

Region-growing based In this variant the ellipse centered in the point $S(S_x, S_y)$, with its two axes equal a and b and rotated by angle ω around the OX axis is given by the

following equations (3).

$$\begin{aligned} x &= S_x + a \cdot \cos\alpha \cdot \cos\omega - b \cdot \sin\alpha \cdot \sin\omega \\ y &= S_y + a \cdot \cos\alpha \cdot \sin\omega + b \cdot \sin\alpha \cdot \cos\omega \end{aligned} \tag{3}$$

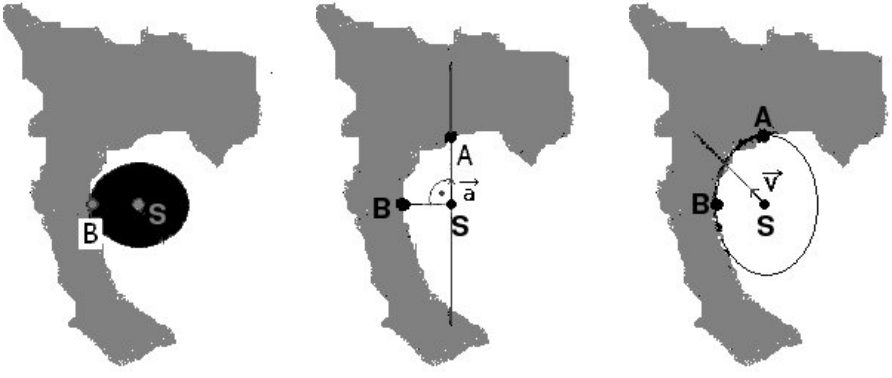
where $\alpha = 0..2\pi$.

In fact it is enough to know locations of 3 points: center point S and 2 extremal points, farer A and closer B, as shown at Figure 2, to compute all values from (4):

$$a = |A - S|, \qquad b = |B - S|, \qquad \omega = \arctan\frac{a_x}{a_y}, \tag{4}$$

where $\overrightarrow{a} = \overrightarrow{S-B} = [a_x, a_y]$.

As it has been stated above, we examine all possible locations of the seed point S. In order to localize the point B we look for the closest pixel marked $'cell'$ around S. It is found by the region growing algorithm [10] with a queue (FIFO - First In First Out) instead of a stack (FILO - First In Last Out) as a structure holding pixels locally marked as 'to be visited' (Figure 2(a)). This modification makes the algorithm to include the closest pixels in the first order. This search is of course spatially limited by the maximum nucleus radius R_{max}. Once the point B located, we place the point A at the closest pixel marked $'cell'$ along two vectors $\overrightarrow{a}$ and $-\overrightarrow{a}$ perpendicular to the vector $\overrightarrow{b} = \overrightarrow{S-B}$ (Figure 2(b)). Once again we limit the search to the radius R_{max}. If any of the points A and B is absent, we reject the ellipse centered at S.



(a) (b) (c)

Fig. 2. Finding of ellipses - method 1 (a) Region growing from the seed point S finding point B (b) Finding of point A (c) Found ellipse

Approximating of the ellipse equation The previous technique is quite sensitive to irregularities in the elliptical form of the cell nucleus and to not removed noise. This is why we have proposed the second one, which is to limit the influence of local variations of the nucleus form from the ideal ellipse.

We start again from the every possible location of the the nucleus center - the seed point S. Starting from it we launch lines equally dividing the angle 2π around this point and collect the closest pixels marked $'cell'$ along this lines - $\mathbf{x}_i = (x_y, y_i)$ (Figure 3). This search is also spatially limited by the maximum nucleus radius R_{max}. If these points number is below some threshold, the potential ellipse centered on S is rejected. To get an ellipse from this set of points we employ a solution proposed

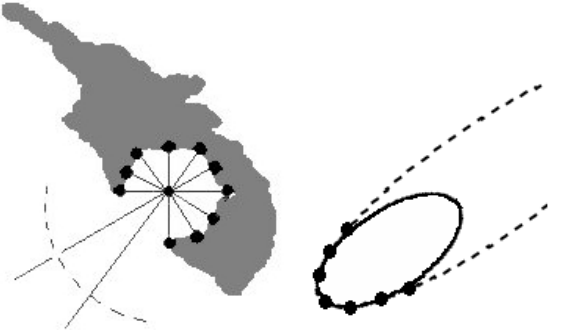


Fig. 3. Finding of ellipses - method 2 (a) Finding border points $\mathbf{x}_i$ from the seed point S (b) Reconstructing an ellipse from points

in [3]. A general conic is represented by a set of points $\mathbf{x}$ satisfying the following (5):

$$\mathbf{A} \cdot \mathbf{XY} = Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0, \tag{5}$$

where $\mathbf{A} = [A\,B\,C\,D\,E\,F]$ is a coefficients vector, initially uknown and $\mathbf{XY} = [x^2\ xy\ y^2\ x\ y\ 1]^T$. In [3] it was shown that a solution minimizing the least square distance between the points $\mathbf{x}_i$ and the conic in presence of some quadratic constraints on $\mathbf{A}$ (in form $A^T\mathbf{C}\mathbf{A} = 1$, to eliminate the trivial solution $\mathbf{A} = 0$) can be found by considering rank-deficient generalized eigenvalue system:

$$\mathbf{D}^T\mathbf{D}\mathbf{A} = \lambda\mathbf{C}\mathbf{A}, \tag{6}$$

where $\mathbf{D} = [\mathbf{XY}_1, \mathbf{XY}_2 \cdots \mathbf{XY}_n]$ is called the *design matrix*, and $\mathbf{C}$ is a 6x6 *contraint matrix*. [3] proposed a contraint $4AC - B^2 = 1$ what gave:

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{7}$$

From six pairs $(\lambda_j, \mathbf{A}_j)$ the only one, with $\lambda_j < 0$, is retained (it was proved there would be exactly one such pair), giving the coefficient vector $\mathbf{A}$ from (5).

However, to calculate contour points of the ellipse, similarly to (3), we need the parameters: center point S, axes a and b and rotation angle ω. To find them we use the following strategy: the coordinate system, turned by angle θ, changes the coordinates accordingly to:

$$x' = x\cos\theta + y\sin\theta \qquad y' = -x\sin\theta + y\cos\theta. \tag{8}$$

Inserting the inverse tranformation:

$$x = x'\cos\theta - y'\sin\theta \qquad y = x'\sin\theta + y'\cos\theta. \tag{9}$$

to (5) we obtain a new formulation of the ellipse equation in the new coordinate system:

$$A'x'^2 + B'xy + C'y'^2 + D'x' + E'y' + F' = 0. \tag{10}$$

where

$$\begin{aligned} A' &= A\cos\theta^2 + B\sin\theta\cos\theta + C\sin\theta^2 \\ B' &= -2A\cos\theta\sin\theta + B\cos\theta^2 - B\sin\theta^2 + 2C\sin\theta\cos\theta \\ C' &= A\sin\theta^2 - B\sin\theta\cos\theta + C\cos\theta^2 \\ D' &= D\cos\theta + E\sin\theta \\ E' &= -D\sin\theta + E\cos\theta \\ F' &= F \end{aligned}. \tag{11}$$

Since the ellipse oriented in parallel to the coordinate system has its B coefficient equal 0, from (11): $B' = 0$ we can compute $\omega = \theta$ satisfying:

$$\cot 2\theta = (A - C)/B \tag{12}$$

and reformulating (10) to the canonical ellipse equation:

$$\frac{(x' - S'_x)^2}{a^2} + \frac{(y' - S'_y)^2}{b^2} = 1 \tag{13}$$

we obtain:

$$a = a' = \sqrt{\frac{\frac{D'^2}{4A'} + \frac{E'^2}{4(C'-F')}}{A'}} \qquad b = b' = \sqrt{\frac{\frac{D'^2}{4A'} + \frac{E'^2}{4(C'-F')}}{C'}} \tag{14}$$

and

$$S'_x = -\frac{D'}{2A'} \qquad S'_y = \frac{-E'}{2C'} \tag{15}$$

from which, using the relations (9), we can calculate original S_x and S_y and compute all needed ellpse points from (3).

3.3 Final selection

So far, we have found a lot of ellipses representing all potential locations of the cells nuclei. Of course, many of them are redundant, located very close one to another and describing the same nucleus, others being "glued" to different cavities in the cells bodies. Such redundant, close ellipses, except one representing the entire group as the nucleus approximation, must be removed from the solution, as well as the false approximations outside the cells bodies.

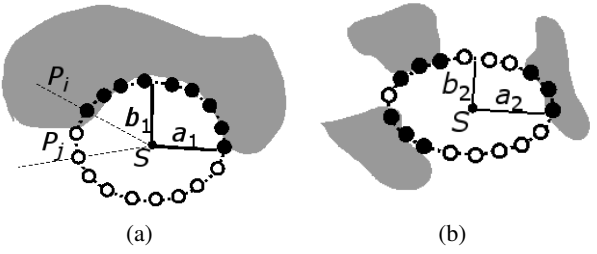


Fig. 4. Fitness function of 2 ellipses built from 18 points e_j, one half of them have binary value 1 (black circles) and the second half value 0 (white circles): (a) 1 sequense of 9 points, $a_1 \approx b_1$; (b) 3 sequences of 3 points each one, $a_2 \ll b_2$. The first ellipse has the fitness function value $(9^P \cdot \frac{a_1}{b_1} \approx 9^P)$ higher than the second one $((3^P + 3^P + 3^P) \cdot \frac{a_2}{b_2} \ll 9^P)$.

To complete this task we need a mean to evaluate the quality of every ellipse in order to eliminate the worst ones. We propose a fitness function measuring the "completeness" of the ellipse shape. If the ellipse **E** is represented by a list of its contour points $(e_1, e_2, \cdots, e_m)$ calculated from (3), each such point can be given a binary value $v(e_j) = 0$ or 1 in function of its location on the border 'cell nucleus/cell body'. The value '1' is assigned if on the profile P_j, going from the ellipse center S to

the point e_j, a pixel combination $('background','background','cell','cell')$ is found in a local neighborhood of the theoretical location of this border (point e_j). The local neighborhood means this part of the profile P_j which is defined by the point e_j moved by length $\frac{b}{2}$ outside and inside the ellipse $\mathbf{E}$, b being the minor ellipse axis. In the opposite case the value '0' is assigned. Among these contour points all sequences seq_k of values '1' are located:

$$Seq_k = \begin{cases} (1,2,\cdots,m) & \text{if} \quad \forall_{j=1,\cdots,m} v(e_j) = 1, \\ (f,\cdots,l) : v(e_{f-1}) = v(e_{l+1}) = 0, \forall_{j=f,\cdots,l} v(e_j) = 1 & \text{otherwise.} \end{cases} \tag{16}$$

where m is the ellipse points e_j number and all indices j are cyclic, i.e. $e_{m+1} = e_1$ and $e_0 = e_m$. The first formulation is for the case where all points e_j have value '1'. The fitness of the ellipse $\mathbf{E}$ is defined as a sum of the length of all the sequences Seq_k raised to the the power p:

$$fitness(\mathbf{E}) = \frac{\min(a,b)}{\max(a,b)} \cdot \sin\left(\frac{\max(a,b)}{R_{max}} \cdot \frac{\pi}{2}\right) \cdot \sum_k (card(Seq_k))^P . \tag{17}$$

The power $P > 1$ promotes ellipses having grouped the pixels with value '1' over those having them scattered (we use $P = 1.3$). The first multiplying factor favors symmetric ellipses (natural nuclei with circular shape, where shorter and longer ellipse axes - $min(a,b)$ and $max(a,b)$ respectively - are equal) and the second one favors ellipses of size close to $2R_{max}$.

Finally, we compare every two ellipses $\mathbf{E}_i$ and $\mathbf{E}_i$. If they are to close each to other we remove one of them with lower fitness function:

$$\begin{array}{lll} \text{if} & ((|S_i - S_j| < 2R_{max}) \text{ and } (fitness(\mathbf{E}_i) \geqslant fitness(\mathbf{E}_j)) & \Rightarrow \quad \text{remove}(\mathbf{E}_j) \\ \text{if} & ((|S_i - S_j| < 2R_{max}) \text{ and } (fitness(\mathbf{E}_i) < fitness(\mathbf{E}_j))) & \Rightarrow \quad \text{remove}(\mathbf{E}_i). \end{array} \tag{18}$$

We also remove all ellipses with the fitness function below some threshold, interactively and visually adjusted by the operator:

$$\text{if} \quad (fitness(\mathbf{E}) < fitness_{min}) \quad \Rightarrow \quad \text{remove}(\mathbf{E}). \tag{19}$$

4. Application

A visual application called "Smart Cell Counter" implementing 3 algorithms of counting (one based on the simple region growing and two variants of the method proposed in this work) was developed using the Java language. The language choice

was done in order to allow developing and using the application under various operating system even without recompiling and to simplify the development. The program was tested in the Linux and MS Windows environments. The cost of this choice is probably slower performance in comparison to other languages like C/C++.

One of the goal of this project was creating a tool to use in the scientific work of physicians from the Department of Infectious Diseases of Medical University of Białystok. The user interface was consulted with them and it allow visual and interactive counting of the liver cells. As the main features one can mention:

- visual and interactive setting of the threshold C (Equation (1)) during preparations of the binary image (Figure 1(d)) - the user can adjust a slider control and observe in the real time the influence of it on the image;
- visual setting of the R_{max} parameter describing the maximum nucleus radius - the user do it by defining a line with 2 mouse clicks on an example cell image;
- visual and interactive setting of the threshold $fitness_{min}$ (Equation (19)) to remove the worst ellipses - the user adjusts a slider control ad immediately sees which ellipses disappear;
- finally, the user can manually, with one mouse click, remove single ellipse judged as invalid and place a new one in a place considered the cell nucleus; however this facility is probably not to use in everyday practice since tedious when the cell number is elevated.

5. Experiments

The method was evaluated on 6 real microscopic images of the hepatic stellate cells in different stages of life (Figure 5), on which a specialist marked cells as the reference. We compare the results to the simple region growing technique where groups of connected pixels marked $'cell'$, which size is above some threshold, are counted as the liver cells. All 3 algorithms use the same binary image. For every image and every algotihm ALG ($SCC1$, $SCC2$) we computed the following statistics (Table 1):

- Ref - reference number of the ellipses (marked by the physician) $= TP + FN$;
- ALG_{TP} - cells of reference detected by the algorithm ALG, even with misplaced nuclei , since the main goal is to count the cells, no to give their exact locations (True Positive);
- ALG_{FP} - false cells detected by the algorithm ALG (False Positive);
- ALG_{FN} - cells of reference not detected by the algorithm ALG (False Negative).

We weren't able to give the True Negative statistics - in this formulation of the cell detection problem it is difficult to precise what it could mean (every pixel correctly

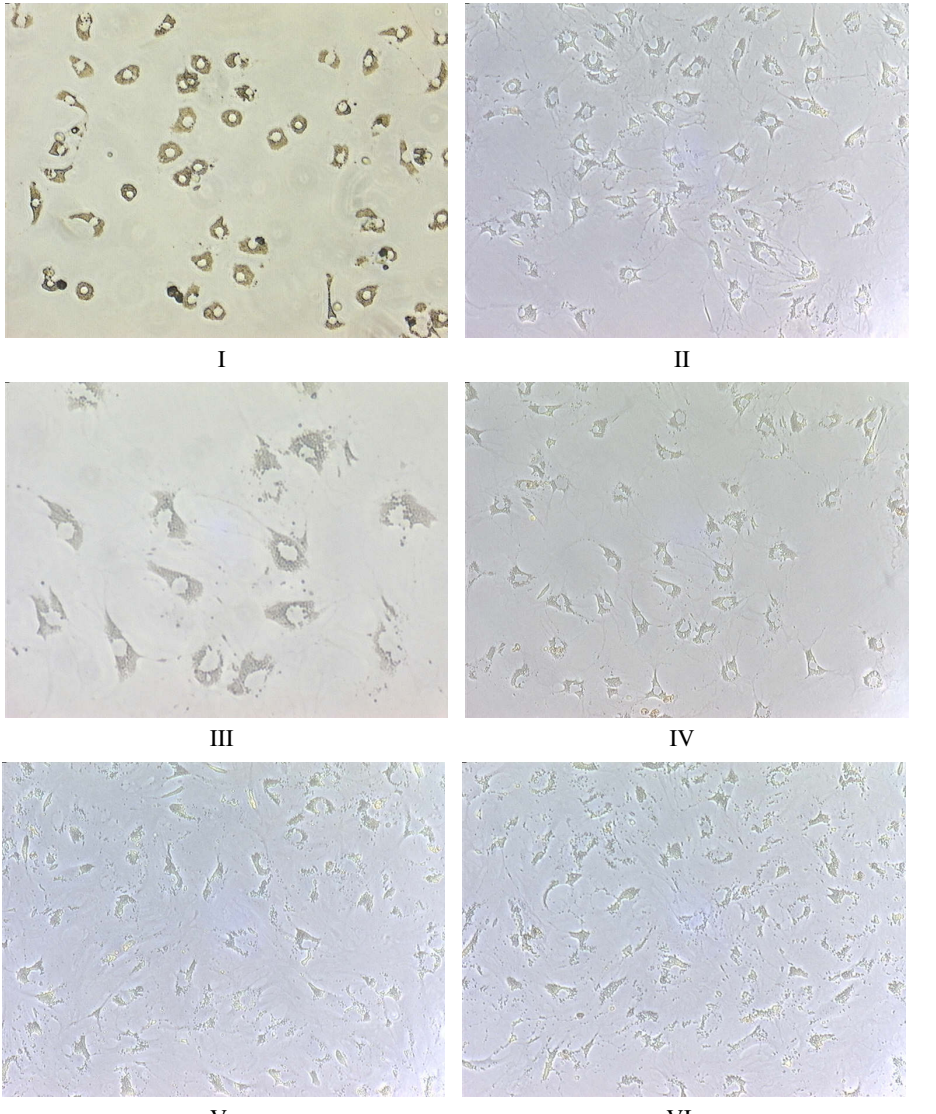
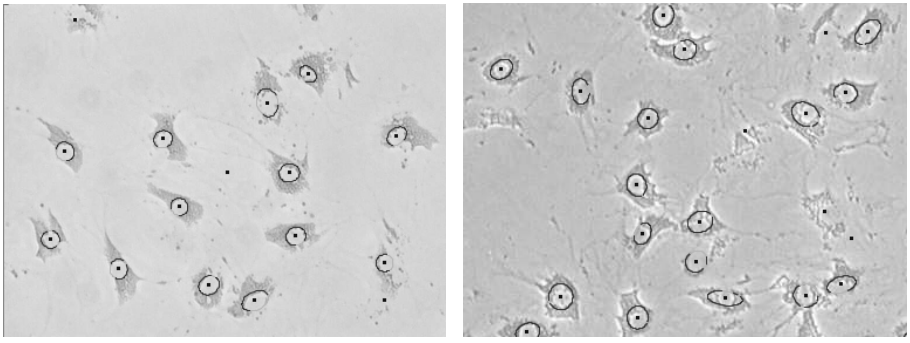


Fig. 5. Analyzed images

rejected as a cell nucleus, every non-cell shape correctly not classified as the cell body ?).

Using these statistics the following benchmarks were computed (Table 2):



(a) (b)

Fig. 6. Example of images with found ellipses marking the cells nuclei. Darker parts of the ellipses mean detected cells bodies.

Table 1. Statistics counted for each algorithm

Image	Ref	RG_{TP+FP}	$SCC1_{FN}$	$SCC1_{TP}$	$SCC1_{FP}$	$SCC2_{FN}$	$SC2_{TP}$	$SCC2_{FP}$
I	47	48	3	44	14	2	45	5
II	55	60	10	45	5	9	46	3
III	13	17	0	13	1	0	13	0
IV	46	48	3	43	4	2	44	4
V	29	53	2	27	12	7	22	10
VI	33	55	6	27	16	11	22	11

- *sensitivity* - ratio of the correctly detected cells to the to the reference cells number ($= \frac{TP}{TP+FN}$);
- $\frac{detected}{Ref}$ - how close the results of the algorithm are to the reality, in term of the cell number only - the detected cell number (ceoorely or not) will be the main result used by cphysicians in every-day work ($=\frac{TP+FP}{TP+FN}$).

Table 2. Benchmarks of the algoritmhs

Image	RG	SCC1		SCC2	
	$\frac{det}{Ref}$	$\frac{det}{Ref}$	sensitivity	$\frac{det}{Ref}$	sensitivity
I	**102.1%**	**123.4%**	93.6%	**106.4%**	95.7%
II	**109.1%**	**90.9%**	81.8%	**89.1%**	83,6%
III	**130.8%**	**107.7%**	100.0%	**100.0%**	100.0%
IV	**104.3%**	**102.2%**	93.5%	**104.3%**	95.6%
V	**182.6%**	**134.5%**	93.1%	**110.4%**	75.9%
VI	**166.7%**	**130.3%**	81.8%	**100.0%**	66.7%

The region growing algorithm was not evaluated in details. In general, if the image contains the distinct, solid cells, it performs well. However, when the cells get

fragmented (e.g. images V and VI), it start detecting every cell fragment as a distinct cell and heavily overestimate their number. Also when the cells are connected, they are detected as a single entity.

Two variants of the proposed method have similar score, but the second version (*SCC*2 - ellipses reconstructed by their equations) seems closer to the reality. The main ratio $\frac{det}{Ref}$ is more often closer to 100% for *SCC*2, especially for harder to treat images (V and VI). The sensitivity is also slightly better for the second version, except for those hard V and VI images (such images, in the late stage of the cell evolution, are rarely treated by physicians, and even manual counting in this case is very imprecise). If one looked at the graphical results (ellipses drawn on the images) it could also notice better modeling of the nuclei by *SCC*2, but this is irrelevant for the simple counting task.

Calculation time The experiments were run on the AMD Athlon XP 2400+ 2.0 Ghz machine with 512 MB RAM installed. All images were treated in their original resolution 760x574 (except for the image III that had been downsized two times in their x and y dimmensions to set the cell sizes comparable between all scans) and the local neighborhood was $N = 21$ (1) . For the preprocessing stage: the Gaussian smoothing took about 1 second, local thresholds calculation - also about 1 second (thanks to their interpolation); the final binarization (2) and the small groups removal were done in the real time.

For the rest of counting in two variants, depending on the complexity of the image: it took from 8/13 seconds (*SCC*1/*SCC*2, the simplest image I) up to 14/17s (image II) for images of the size 760x574 and 4/3 s for the half size image III. In general the *SCC*1 is a bit faster, but the difference is not very large.

6. Conclusion

The presented method seems to give good results even when the cells to count are fragmented. Especially is such cases it shows its superiority over the simple region growing algorithm. Two variants of ellipse founding techniques perform well, with a slight advantage of the one based on the direct ellipse equation reconstruction. However, it requires more profound validation, with more images, ideally during the every-day physician work. Is should be achieved more easily thanks to the simple-to-use, graphical, multiplatform and interactive application implementing the method.

The method has been designed to count the hepaitc stellate cells, but it could be also used to work with other cells/object with the same characteristic - homogeneous, elliptical, bright central region surrounded by darker fragments.

Bibliography

[1] Chen, Y., Biddell, K., Sun, A., Relue, P.A., Johnson, J.D.: An Automatic Cell Counting Method for Optical Images; Proc. BMES/EMBS 1999 Conf., vol. 2, pp. 819, 1999.

[2] Dorini, L.B., Minetto, R., Leite, N.J.: White blood cell segmentation using morphological operators and scale-space analysis; Proc. SIBGRAPI 2007 Conf., pp. 294-304, 2007.

[3] Fitzgibbon, A., Pilu, M., Fisher, R.B.: Direct Least Square Fitting of Ellipses; IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 21, no. 5, pp. 476-480, 1999.

[4] Friedman, S.L., Bansal M.B.: Reversal of hepatic fibrosis - fact or fantasy ?; Hepatology, 43(2 Suppl 1), pp. S82-88, February 2006.

[5] Guillon, P.: Contribution à l'évaluation de causes d'infertilité humaine d'origine masculine - Recherche de solutions optimales en segmentation, reconnaisance de form et classification appliquées à des images microscopique de spermatozoïdes; PhD Thesis, Université de Rennes I, France, 2000.

[6] He, W., Wilder, J.: Nucleus shape recognition for an automated hematology analyzing system; Proc. EMBS/BMES 2002 Conf., vol. 2, pp. 1043-1044, 2002.

[7] Korzyńska, A., Strojny, W., Hoppe, A., Wertheim, D., Hosner, P.: Segmentation of microscope images of living cells; Pattern Analysis and Applications, vol. 10, no. 4, pp. 301-319, 2007.

[8] Liao, Q., Deng, Y.: An accurate segmentation method for white blood cell images; Proc. IEEE BMI 2002 Int. Symp.., pp. 245-248, 2002.

[9] Ma, Y.D., Dai, R.L., Lian, L., Zhang, Z.F.: An Counting and Segmentation method of Blood Cell Image with Logical and Morphological Feature of Cell; Proc. ICONIP 2001 Conf., IDNumber115, `http://www.cse.cuhk.edu.hk/apnna/proceedings/iconip2001/` Shanghai, China, 2001.

[10] Pratt, W.K.: Digital Image Processing; Wiley, 2001.

[11] Refai, H., Li, L., Teague, T.K., Naukam, R.: Automatic count of hepatocytes in microscopic images; Proc. ICIP 2003 Conf., vol.2, pp. II-1101-4 vol.3, 2003.

[12] Witkowski, Ł.: Komputerowy system do śledzenia ruchu plemników, wyznaczania parametrów ruchu oraz wyliczenia gęstości nasienia; Proc. KBIB 2007 Conf., P50, 2007.

[13] Zduńczuk, M., Korzyńska, A.: Modyfikacja metod segmentacji hybrydowej pod kątem analizy obrazów komórek w hodowlach; Proc. KBIB 2007 Conf., P99, 2007.

PÓŁAUTOMATYCZNA METODA ZLICZANIA KOMÓREK GWIAŹDZISTYCH WĄTROBY NA MIKROSKOPOWYCH OBRAZACH IN-VITRO PRZEZ MODELOWANIE ICH ELIPTYCZNYCH JĄDER

Streszczenie: W artykule tym zaprezentowana jest nowa, półautomatyczna metoda zliczania komórek wątroby. Zamiast skupiać się na ciałach komórek (w niektórych fazach ich życia nieregularnych i pofragmentowanych) lokalizuje ona ich jądra, które są jasnymi, jednolitymi i eliptycznymi strukturami otoczonymi przez ciemniejsze ciało (lub jego fragmenty). Jądra komórek są modelowane na 2 sposoby: przez lokalny algorytm rozrostu obszaru i przez odtworzenie równania elipsy z jej punktów na obwodzie. Wśród wszystkich znalezionych elips (początkowa każda jej potencjalna lokalizacja jest brana pod uwagę) wybierane są lokalnie najlepsze oraz te, których dopasowanie, mierzone specjalną funkcją, jest powyżej pewnego progu. Metoda została zaimplementowana jako graficzna, wieloplatformowa, przyjazna dla użytkownika aplikacja w języku JAVA. Jej działanie zostało ocenione na rzeczywistych mikroskopowych obraz in-vitro komórek wątroby.

Słowa kluczowe: przetwarzanie obrazów medycznych, zliczanie komórek, komórki gwiaździste wątroby

Joanna Gościk[1], Józef Gościk[2]

NUMERICAL EFFICIENCY OF ITERATIVE SOLVERS FOR THE POISSON EQUATION USING COMPUTER CLUSTER

Abstract: We present a set of numerical results which were obtained by systematic investigation of efficiency of compilers implemented on Mordor cluster (*http://mordor.wi.pb.edu.pl*) running Linux distribution CentOS 4, kernel ver. 2.6. As a generic problem the finite difference based framework for solution of the Poisson equation has been taken (with discretization on grid topologically equivalent to a Cartesian grid). The PDE converted to an algebraic system of equations is solved by adopting so-called nonstationary, Krylov type, iterative methods: conjugate gradient (CG), bi-conjugate gradient (Bi-CG), conjugate gradient squared (CGS) and bi-conjugate gradient stabilized (Bi-CGSTAB). The code was implemented using two different compilers, such as *gcc* (GNU Compiler Collection - ver. 3.4.6) and *icc* (Intel C++ Compiler - ver. 9.1). All performances reported were done with the Xeon 3.2 GHz processor that has own memory 2 GB.

Keywords: Iterative solvers, Finite difference method, Poisson equation

1. Introduction

The need to solve systems of linear algebraic equations is ubiquitous through computational physics. Such systems typically arise from the discretization of partial differential equations and for practically important, real engineering problems are very large. The demands required for the computational resources cause that often so-called direct methods are not applicable and the iterative techniques are one reasonable alternative for solution such sets of linear equations [1]. It seems that this dynamically growing application area is the main reason why the iterative techniques are so intensive investigated last years. Another well known motivation is impact of parallel architectures. Direct methods are more complex to implement in parallel than are iterative methods.

[1] Faculty of Computer Science, Bialystok Technical University, Bialystok, Poland
[2] Faculty of Mechanical Engineering, Bialystok Technical University, Bialystok, Poland

Implementation of the iterative techniques however is still not so obvious despite of the huge progress in the last years. In another words there is no preferred or the best method preferable among others. From the other side it is very well known that even the system is well conditioned (even treated by proper pre-conditioning technique) often converges slowly or even diverges [2], [3].

The paper is devoted to illuminate an early stage of progress in developing own made library of the iterative solvers implemented on cluster computer - Mordor cluster (*http://mordor.wi.pb.edu.pl*). Mainly for the comparisons reasons the reported results were obtained by sequentially executed (using one procesor) algorithms. The plan of the paper is as follows. First, we define the Poisson equation and characterize resulting from discretization by finite difference method sets of linear equations. Next, we describe the group of the interested iterative techniques which belongs to the Krylov subspace methods. In particular, we describe short characteristic of the methods, their algorithmic realization schemes and show how they are related each other. Taxonomy and especially historical chronology are given here basing on excellent review paper by Saad and Van der Vorst [4]. Finally we present efficiency of the investigated methods in terms of the CPU time consuming and rate of convergence.

2. The Poisson equation and its discretization

The Poisson equation is very well known in computational physics. In fact the equation play very important role in many branches of the scientific computing and numerical simulation. Close to the physical interpretation it describes fluid dynamic (in a stream function – vorticity formulation), heat conduction and many others. It is also a kernel for generation of structured meshes in arbitrary domains (so called body fitted mesh generation techniques) and more sophisticated area as in medical imaging where is used for electro-encephalographic source analysis. The latest new branch are requirements of the current generation of the GPU (Graphical Processor Unit) in which range of application extends traditional graphic problems to capability of more general computing needed for example for physical modelling.

In the paper as a generic problem we consider the two dimensional Poisson equation with Dirichlet boundary conditions on the rectangular domains $\Omega = \left[0, L_x; 0, L_y\right]$ (in particular on the unit square $\Omega = [0,1;0,1]$)

$$\nabla^2 \phi(x) + q(x) = 0 \quad x \in \Omega \subset \mathbb{R}^2 \tag{1}$$

with boundary conditions

$$\phi(x) = \phi_D(x) \quad x \in \partial\Omega. \tag{2}$$

We next assume that the boundary value problem (1) and (2) is solved by means of finite differences. This transforms differential equation to its finite difference form. So, after discretization, we have defined difference problem

$$L_h \cdot \phi_h = b_h \quad , \tag{3}$$

where b_h is the grid function that is a projection of the right hand side of the original differential problem on the grid and ϕ_h is the grid function which is a projection of the exact solution on the grid. The operator L_h is determined from the grid functions and depends in general on parameters called grid steps h_x, h_y. In particular we assume that L_h is linear operator acting on the regular grid 2D grid ϕ_h (it is assumed that the grid cover the whole interested region Ω uniformly, so $h_x = h_y \rightarrow h$) and is described by a set of stencils, with possibly varying entries each of size no longer than 3×3 (2×3 at the boundary and 2×2 at the corners).

Finally, the main subject of interest in the paper is a vectorial matrix form of a system of linear difference equations (3) which leads to a linear algebra problem

$$\mathbf{A}\phi = \mathbf{b} \quad , \tag{4}$$

where $\mathbf{A}$ and $\mathbf{b}$ are given and to be real, $\mathbf{A}$ is sparse, non-singular $N \times N$ matrix, $\mathbf{b}$ an N-vector (with assumption that N is large) and ϕ is the vector of the unknowns. Strictly we will try to find acceptable approximations of the Poisson equation solution by solve the equation (4). The most important is that in computational schemes the equation needs to be solved repeatedly for different source contributions. In consequence the solution of the Poisson equation is very often the most time consuming part of the overall computational scheme. Then naturally is still needing for working the most efficient solvers for this task.

3. Iterative solvers and methods

Each an iterative solver works by repeatedly apply a series of operations to an approximate solution to the linear system, with the error in the approximate solution being reduced by each application of the operations. The basic form of all iterative methods is given in Fig. 1. According to the pseudo code given in Fig. 1 an iterative scheme produces a sequence of vectors $\mathbf{x}$ which should converge to the vector $\mathbf{x}$ satisfying the system of equations (4). So apart of choosing of the most effective F function it is very important also to arrange a criterion which will decide when to stop creation a sequence of $\mathbf{x}$. Detailed discussion about the subject can be found in [3]. In our work we decide to apply two stopping criteria which are defined in Section 4.3.

```
set x_0 to an initial estimate of x
j=0
while j < j_max and error > tolerance
  perform some operations x_{j+1} = F(x_j)
  j=j+1
```

Fig. 1. A generic iterative method.

Relating to the iterative schemes which produce successive approximation we restrict ourselves to F functions constructed on the base of the simplest Krylov subspace based iterative techniques. The methods implemented and tested in this study were Conjugate Gradient (CG), Bi-Conjugate Gradient (Bi-CG), Conjugate Gradient Squared (CGS) and Bi-Conjugate Gradient STABilised (Bi-CGSTAB).

It is also especially needed to pointed out that the algorithms for all of solver mentioned were consciously elaborated in the standard (unpreconditioned) representation.

3.1 The Conjugate Gradient (CG) method.

$$
\begin{aligned}
&\mathbf{r}_0 \Leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0 \\
&\mathbf{p}_0 \Leftarrow \mathbf{r}_0 \\
&\text{FOR } j = 0, 1, \ldots, \text{until convergence DO} \\
&\quad \alpha_j \Leftarrow \mathbf{r}_j^{\mathrm{T}}\mathbf{r}_j / (\mathbf{A}\mathbf{p}_j)^{\mathrm{T}}\mathbf{p}_j \\
&\quad \mathbf{x}_{j+1} \Leftarrow \mathbf{x}_j + \alpha_j \mathbf{p}_j \\
&\quad \mathbf{r}_{j+1} \Leftarrow \mathbf{r}_j - \alpha_j \mathbf{A}\mathbf{p}_j \\
&\quad \beta_j \Leftarrow \mathbf{r}_{j+1}^{\mathrm{T}}\mathbf{r}_{j+1} / \mathbf{r}_j^{\mathrm{T}}\mathbf{r}_j \\
&\quad \mathbf{p}_{j+1} \Leftarrow \mathbf{r}_{j+1} + \beta_j \mathbf{p}_j
\end{aligned}
$$

Fig. 2. The standard (unpreconditioned) CG algorithm.

The CG method seems to be the most representative for a number of projection-type methods on Krylov subspaces. It is especially suited for symmetric positive definite matrices, for which it was originally devised in 1952 by Hestens and Stiefel [5]. However as its pointed out in [4] scarsely in the early 1970's it received the more attention in computational practice. CG is a descendant of the method of steepest

descent that avoids repeated search in the same direction by making search directions orthogonal to each other in the energy norm associated with the matrix.

For clarity of the pseudo codes description we use following, uniform notation throughout the paper: $\mathbf{x_j}$ denotes the vector $\mathbf{x}$ during the j-th iteration (consequently $\mathbf{x_0}$ is the initial guess), $\mathbf{r_j}$ is the residual which indicates how far the iterative sequence of $\mathbf{x}$ is from the solution and is defined as $\mathbf{r_j} = \mathbf{b} - \mathbf{A}\mathbf{x_j}$.

3.2 The Bi-Conjugate Gradient (Bi-CG) method.

$r_0 \Leftarrow b - Ax_0$
Choose r_0^* so that $r_0^T r_0^* \neq 0$
$p_0 \Leftarrow r_0$
$p_0^* \Leftarrow r_0^*$
FOR $j = 0, 1, \ldots,$ until convergence DO
$\alpha_j \Leftarrow r_j^T r_j^* / (Ap_j)^T p_j^*$
$x_{j+1} \Leftarrow x_j + \alpha_j p_j$
$r_{j+1} \Leftarrow r_j - \alpha_j A p_j$
$r_{j+1}^* \Leftarrow r_j^* - \alpha_j A^T p_j^*$
$\beta_j \Leftarrow r_{j+1}^T r_{j+1}^* / r_j^T r_j^*$
$p_{j+1} \Leftarrow r_{j+1} + \beta_j p_j$
$p_{j+1}^* \Leftarrow r_{j+1}^* + \beta_j p_j^*$

Fig. 3. The standard (unpreconditioned) Bi-CG algorithm.

The pioneering idea of Bi-CG was fomulated in 1952 by [6], which used biothogonality relation to redure iteratively a matrix to the tri-diagonal form. Later his idea was adopted by [7] mainly to develop the algorithms which free the CG solver from its limitation of only being applicable to symmetric systems. For this the orthogonal sequence used in the CG method has been replaced by two mutually orthogonal sequences, one based on the system $\mathbf{A}$ and the other on its transpose $\mathbf{A^T}$. In result, implicitly the Bi-CG algorithm solves additionally a dual linear system $\mathbf{A^T x^* = b^*}$. Consequentely the symbols with the asterix which will appear next in the paper should be treated as a connected with the dual approximate solution.

3.3 The Conjugate Gradient Squared (CGS) method.

$$
\begin{aligned}
&\mathtt{r}_0 \Leftarrow \mathtt{b} - \mathtt{A}\mathtt{x}_0 \\
&\text{Choose } \mathtt{r}_0^* \text{ arbitrarily} \\
&\mathtt{p}_0 \Leftarrow \mathtt{u}_0 \Leftarrow \mathtt{r}_0 \\
&\text{FOR } \mathtt{j} = 0, 1, \ldots, \text{until convergence DO} \\
&\quad \alpha_j \Leftarrow \mathtt{r}_j^{\mathrm{T}} \mathtt{r}_0^* / (\mathtt{A}\mathtt{p}_j)^{\mathrm{T}} \mathtt{r}_0^* \\
&\quad \mathtt{q}_j \Leftarrow \mathtt{u}_j - \alpha_j \mathtt{A}\mathtt{p}_j \\
&\quad \mathtt{x}_{j+1} \Leftarrow \mathtt{x}_j + \alpha_j (\mathtt{u}_j + \mathtt{q}_j) \\
&\quad \mathtt{r}_{j+1} \Leftarrow \mathtt{r}_j - \alpha_j \mathtt{A}(\mathtt{u}_j + \mathtt{q}_j) \\
&\quad \beta_j \Leftarrow \mathtt{r}_{j+1}^{\mathrm{T}} \mathtt{r}_0^* / \mathtt{r}_j^{\mathrm{T}} \mathtt{r}_0^* \\
&\quad \mathtt{u}_{j+1} \Leftarrow \mathtt{r}_{j+1} + \beta_j \mathtt{q}_j \\
&\quad \mathtt{p}_{j+1} \Leftarrow \mathtt{u}_{j+1} + \beta_j (\mathtt{q}_j + \beta_j \mathtt{p}_j)
\end{aligned}
$$

Fig. 4. The standard (unpreconditioned) CGS algorithm.

The CGS method was proposed in 1989 by Sonneveld [8] and represents some idea of improvement of the Bi-CG solver. In computational realization it applies the updating operations for the **A** sequence and $\mathbf{A^T}$ sequence to both vectors. In concept, or theoretically this approach would double the convergence rate, but in practice convergence is not so ideal.

3.4 The Bi-Conjugate Gradient STABilized (Bi-CGSTAB) method.

The Bi-CGSTAB method was developed by Van der Vorst [9] as a hybrid of different conjugate gradient based methods (CGS, Bi-CG and not investigated in this paper GMRES method). In intention the method has been elaborated to solve general systems of equations (with **A** symmetric and non symmetric) and especially in order to avoid the often highly irregular convergence patterns of the CGS and Bi-CG.

$$
\begin{aligned}
&\mathbf{r}_0 \Leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0 \\
&\text{Choose } \mathbf{r}_0^* \text{ arbitrarily} \\
&\mathbf{p}_0 \Leftarrow \mathbf{r}_0 \\
&\text{FOR } \mathbf{j} = 0, 1, \ldots, \text{until convergence DO} \\
&\quad \alpha_j \Leftarrow \mathbf{r}_j^{\mathrm{T}} \mathbf{r}_0^* / (\mathbf{A}\mathbf{p}_j)^{\mathrm{T}} \mathbf{r}_0^* \\
&\quad \mathbf{s}_j \Leftarrow \mathbf{r}_j - \alpha_j \mathbf{A}\mathbf{p}_j \\
&\quad \omega_j \Leftarrow (\mathbf{A}\mathbf{s}_j)^{\mathrm{T}} \mathbf{s}_j / (\mathbf{A}\mathbf{s}_j)^{\mathrm{T}} \mathbf{A}\mathbf{s}_j \\
&\quad \mathbf{x}_{j+1} \Leftarrow \mathbf{x}_j + \alpha_j \mathbf{p}_j + \omega_j \mathbf{s}_j \\
&\quad \mathbf{r}_{j+1} \Leftarrow \mathbf{s}_j - \omega_j \mathbf{A}\mathbf{s}_j \\
&\quad \beta_j \Leftarrow \left(\mathbf{r}_{j+1}^{\mathrm{T}} \mathbf{r}_0^* / \mathbf{r}_j^{\mathrm{T}} \mathbf{r}_0^*\right) \cdot (\alpha_j / \omega_j) \\
&\quad \mathbf{p}_{j+1} \Leftarrow \mathbf{r}_{j+1} + \beta_j \left(\mathbf{p}_j - \omega_j \mathbf{A}\mathbf{p}_j\right)
\end{aligned}
$$

Fig. 5. The standard (unpreconditioned) Bi-CGSTAB algorithm.

4. A comparison of solvers

All results reported in this section are done with the Xeon 3.2 GHz processor on Mordor cluster (*http://mordor.wi.pb.edu.pl*) running Linux distribution CentOS 4, kernel ver. 2.6. The codes were generated using two different compilers: gcc (GNU Compiler Collection - ver. 3.4.6) and icc (Intel C++ Compiler - ver. 9.1). For each solver we report on the elapse CPU time with a systematic investigation of the impact of

- the compiler optimization options,
- the grid size (nx,ny).

An attention is also directed on comparison of the linear solvers in terms of their convergence rate.

4.1 The solvers test case

To compare the convergence and overall behaviour of the iterative procedure of the tested solvers they were used to solve a two-dimensional Poisson problem with Dirichlet boundary conditions. The problem has been chosen taking into account two aspects, namely physical connections with the cases encountered in computational fluid dynamic (CFD) as well as relative simplicity of the geometry and boundary conditions which provide to case which have an analytical (exact) solution.

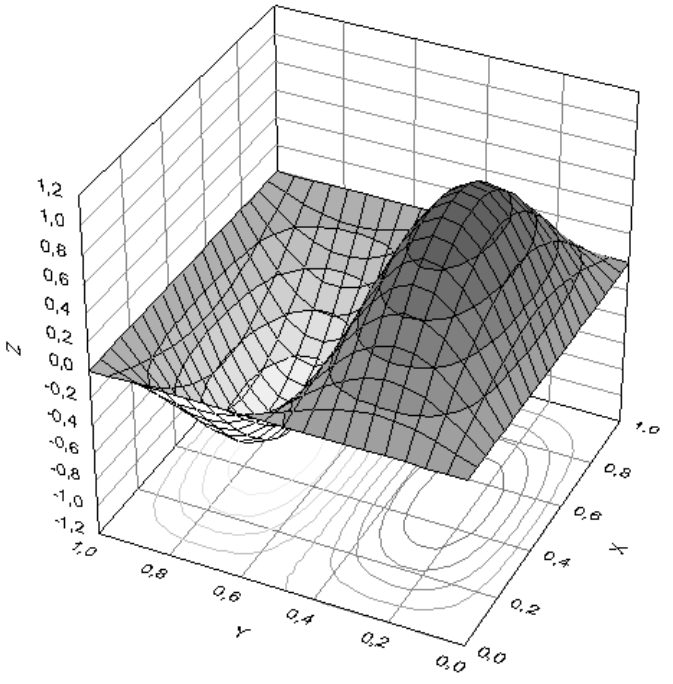


Fig. 6. Analytical solution.

The test case was a finite difference discretization of the Poisson equation applied to a rectangular domain ($0 \le x \le L_x$, $0 \le y \le L_y$) within is defined the two dimensional equation

$$\partial^2_{xx}\phi + \partial^2_{yy}\phi + q = 0 \tag{5}$$

where ϕ is treated as a continuous scalar field and q is a sinusoidally varying source term given by

$$q(x,y) = \left[\left(\frac{\pi}{L_x}\right)^2 + \left(\frac{2\pi}{L_y}\right)^2\right] \cdot \sin\left(\frac{\pi}{L_x} \cdot x\right) \cdot \sin\left(\frac{2\pi}{L_y} \cdot y\right) \tag{6}$$

For the Dirichlet problem which is defined imposing $\phi = 0$ on all boundaries, the test problem (5), (6) has the exact solution

$$\phi(x,y) = \sin\left(\frac{\pi}{L_x} \cdot x\right) \cdot \sin\left(\frac{2\pi}{L_y} \cdot y\right) \tag{7}$$

which is also additionaly presented in Fig. 6

4.2 Assessment of the CPU time according to compilers optimization options

Test of influence of the compiler optimization options affecting the code speed is presented in Tables 1 and 2. The details of the compiler features which enhance an application performance are described in [10] for *gcc* compiler and in [11] for *icc* compiler.

Table 1. gcc (version 3.4.6) performance.

Compiler options	Iterative solver			
	CG	BiCG	CGS	BiCGSTAB
-O0[1]	73.61	117.81	136.76	122.88
-O	44.47	71.06	82.54	76.58
-O1	44.81	71.13	82.36	76.54
-O1 -O	44.37	71.13	82.40	76.30
-O2	44.93	71.70	83.44	76.91
-O3	44.92	74.84	93.01	81.79
-Os	46.10	74.35	85.44	78.65

The elapsed CPU times given in Tables 1 and 2 are documented by solution of the biggest problem. Timings were made using the C `clock()` function which provide accuracy to 1/1000th of second. The test cases were run to fulfill stopping criteria (see eq. (8)).

Table 2. icc (version 9.1) performance.

Compiler options	Iterative solver			
	CG	BiCG	CGS	BiCGSTAB
-O1	35.95	59.41	71.01	58.95
-O2[1]	35.66	59.79	71.24	59.70
-O3	36.18	59.62	71.41	59.64
-fast	27.31	45.77	54.99	44.47
-O2 -ipo	36.48	60.78	71.73	60.63

Summing up the results we can formulate the following general conclusions. When using *gcc* practically we should not expect any increase in performance by

[1] default compiler option

proper choosing of optimizing compiler options. Only one recommendation should be formulated to no using *gcc* compiler without specification of any optimizing options. In other words that means that it should be omitted using the default compiler option [10].

Almost the same remark we can write characterizing results obtained by using *icc*. We found also that the execution time is relative more sensitive to the chosen compiler options. There is also one remarkable exception when the option `-fast` is used. By analyzing the times given in the Table 2 we can expect then a significant increase in performance of the code (the CPU time elapsed is shorter in a $20 \div 40\%$).

However the general conclusion is that the involving *icc* compiler generates codes which are considerably faster than those resulted from *gcc*.

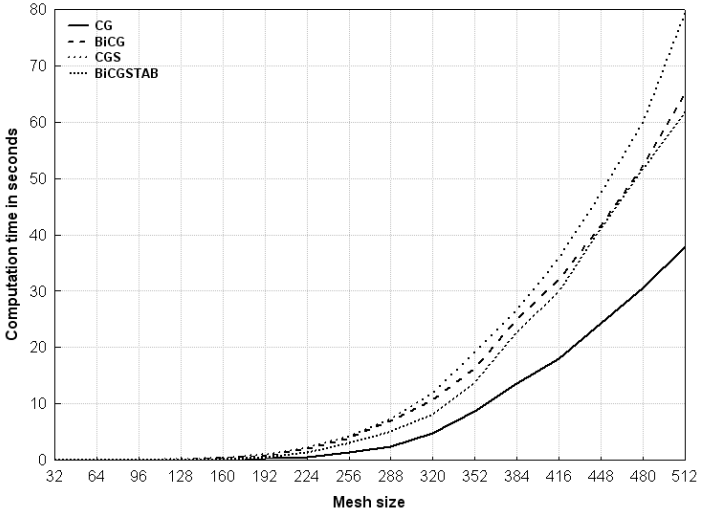


Fig. 7. Computation time.

Taking into account those observations for investigate the impact of the grid size on the CPU time, we performed a series of systematic calculation using *icc* with `-fast` option (which generate a fastest code). Results are given in Figure 7 in a form of variation of the solution time as the number of equations is increased. For the test problem at hand, a symmetric sparse linear system, the iterative methods we have tested confirm superior recommendation of the CG method for their solve. In that context is obvious that other methods took longer to solve the same problem.

4.3 Convergence of the solvers

A stopping criteria is based on the norm of the current residual related to the norm of the vector **b**, initial right hand side vector of the equation (4), and is defined as

$$\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} < \varepsilon \tag{8}$$

where ε is a tolerance chosen from the $\left\langle 10^{-6}, 10^{-1} \right\rangle$. It is worth to mention, that the calculation is also stopped, if the left hand side of the equation (8) does not become smaller than ε within the chosen maximum number of iterations. At our calculations we took that the maximum number of iterations permitted each algorithm to perform has been set as a 1000. With no exception for each run we took $\mathbf{x_0} = 1$ as an initial guess.

To compare the convergence rates of the four iterative solvers, runs were made using 5^2 and 512^2 meshes, with the residual at each iteration being printed out for plotting, the abscissa being as iteration number. Figures 8 and 9 show the convergence of the solvers for the 5^2 mesh and the 512^2 mesh respectively. As is clearly shown the solution on the mesh 5^2 fails the convergence due to small h. The very nice and representative for the Krylov space methods is Figure 9 containing convergence rate history for 512^2 mesh.

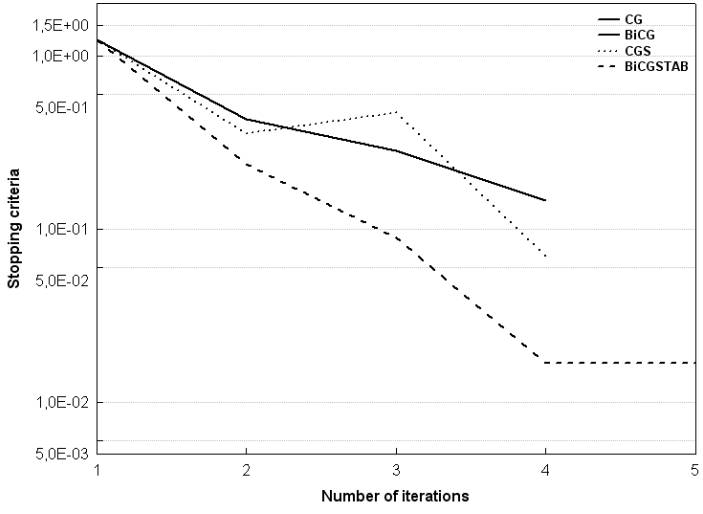


Fig. 8. Convergence plot for mesh 5^2.

First of all it exhibits an irregular convergence with typical [2], a non-monotonic reduction in residual. The CG solver shows an initial period approximately linear convergence, after which the rate of convergence increases. The Bi-CGSTAB solver has a more irregular rate of convergence than their CG or Bi-CG counterparts.

However in general the rate of convergence is highest among the four tested solvers. The worst of all rate exhibits the CGS method. In an initial period the CGS solver shows even some tendency to divergence. After this initial period, the rate of convergence increases and finally is near close those observed for CG and Bi-CG methods.

Finally we should pointed out that the poor in general behavior of the convergence rate of the Krylov type solvers once again confirms that they should be implemented with segments which guaranties a proper preconditioning.

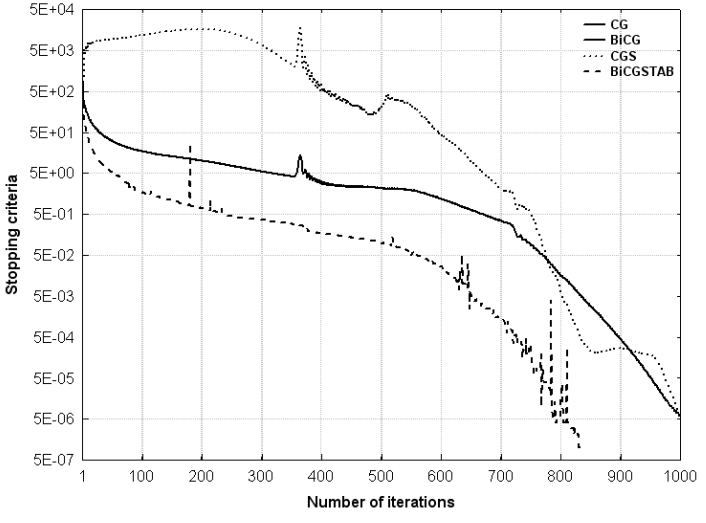


Fig. 9. Convergence plot for mesh 512^2.

5. Conclusion

The presented paper documents the first step in realization of the project leading in intention to develop numerically efficient software addressed to solution of the Poisson equation using parallel computer with distributed memory. Because results were obtained consequently in the sequential environment without any special tuning of

the source they will be helpful in reliable investigation of performance parallelisation. In the next step we will consider in details implementation analyzed solvers on Mordor cluster by appropriate construction of

- data structures for sparse matrices,
- data parallel algorithms for sparse matrix vector multiplies,
- reduction operators for inner product computation.

Bibliography

[1] Saad, Y.: Iterative Methods for Sparse Linear Systems, Second Edition, SIAM, Philadelphia, Pa, 2003.

[2] Van der Vorst, H.A.: The Iterative Krylov Methods for Large Linear Systems, Cambridge University Press, Cambridge, 2003.

[3] Barrett, R., Berry, M., Chan, T., Demmel, J., June, D., Dongarra, J., Eijkhout, V., Pozo, R., Romine, Ch., Van der Horst, H.: Templates for the solution of linear systems: Building blocks for iterative methods, Second Edition, SIAM Publication, 2003.

[4] Saad, Y., Van der Vorst, H.A.: Iterative solution of linear systems in the 20-th Century, Journal of Computational and Applied Mathematics, Vol. 123, No. 1-2, pp. 1-33, 2000.

[5] Hestens, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bur. Stand, Vol. 49, No. 6, pp. 409-436, 1952.

[6] Lanczos, C.: Solution of systems of linear equations by minimized iterations, J. Res. Nat. Bur. Stand, Vol. 49, No. 1, pp. 33-53, 1952.

[7] Fletcher, R.: Conjugate gradient methods for indefinite systems, In Edited by G.A.Watson, Proceedings of the Dundee Biennal Conference on Numerical Analysis 1974, Springer Verlag, New York, pp. 73-89, 1975.

[8] Sonneveld, P.: CGS: A fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Statist. Comput, Vol. 10, pp. 36-52, 1989.

[9] Van der Vorst, H.A.: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comput, Vol. 13, pp. 631-644, 1992.

[10] Stallman, R.M., GCC Developer Community: Using the GNU Compiler Collection, GNU Press, 2004.

[11] Intel C++ Compiler Optimization Applications, Intel Corporation, 1996-2006.

EFEKTYWNOŚĆ NUMERYCZNA ITERACYJNYCH TECHNIK ROZWIĄZANIA RÓWNANIA POISSONA NA KLASTRZE KOMPUTEROWYM

Streszczenie: Przedstawiono wstępne wyniki badania efektywności sekwencyjnego przetwarzania danych w algorytmach rozwiązywania dużych układów równań liniowych na klastrze obliczeniowym Mordor (*http://mordor.wi.pb.edu.pl*) zarządzanym przez system operacyjny Linux (dystrybucja CentOS 4, wersja jądra 2.6). Szczególną uwagę zwrócono na wpływ doboru opcji optymalizacyjnych w dostępnych kompilatorach na wydajność obliczeniową kodu komputerowego. Jako bazowe do rozważań przyjęto duże układy równań liniowych z macierzą współczynników o strukturze rzadkiej. Takie układy równań generowane są w procedurze numerycznego rozwiązania równania Poissona, którego aproksymację otrzymuje się na gruncie metody różnic skończonych (dyskretyzacja na uporządkowanej siatce różnicowej w kartezjańskim układzie współrzędnych prostokątnych). Cząstkowe równanie różniczkowe przekształcone do postaci układu równań liniowych rozwiązano z wykorzystaniem czterech metod iteracyjnych typu Kryłowa: gradientów sprzężonych (CG), gradientów bisprzężonych (Bi-CG), kwadratowego gradientu sprzężonego (CGS) oraz stabilizowaną metodą wzajemnie sprzężonych gradientów (Bi-CGSTAB). Metody te wdrożono generując własne oprogramowanie oraz zaimplementowano z wykorzystaniem dwóch różnych kompilatorów *gcc* (GNU Compiler Collection - wesja 3.4.6) oraz *icc* (Intel C++ Compiler - wersja 9.1). Wyniki wszystkich testów efektywności obliczeniowej uzyskano rozwiązując sformułowane zagadnienie testowe przy użyciu jednego procesora Xeon 3.2 Ghz wchodzącego w skład jednego węzła obliczeniowego z pamięcią własną 2GB.

Słowa kluczowe: Metody iteracyjne, Metoda różnic skończonych, równanie Poissona

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Tomasz Grześ[1], Valery Salauyou[1]

ALGORYTMY KODOWANIA STANÓW WEWNĘTRZNYCH AUTOMATU SKOŃCZONEGO DO MINIMALIZACJI POBORU MOCY

Streszczenie: Kodowanie stanów wewnętrznych automatu skończonego jest jednym z ważniejszych procesów podczas syntezy automatu. Zastosowanie odpowiedniego algorytmu pozwala m.in. obniżyć pobór mocy. W artykule skoncentrowano się na algorytmach minimalizujących pobór mocy. Przeprowadzono badania nad algorytmem kodowania kolumnowego, opisanego w pracy [1] oraz nad dwoma algorytmami opracowanymi przez autorów: sekwencyjnym [7] oraz rafinacyjnym. Badania przeprowadzono na standardowych układach testowych, opracowanych w Microelectronics Center of North Carolina [9]. Wyniki badań wykazują znaczące zmniejszenie poboru mocy układów zakodowanych z wykorzystaniem algorytmu sekwencyjnego w porównaniu z poborem z wykorzystaniem algorytmu kodowania kolumnowego (średnio o 12%); zastosowanie algorytmu rafinacyjnego pozwoliło obniżyć moc średnio o kolejny 1%.

Słowa kluczowe: Automat skończony, kodowanie stanów, obniżanie poboru mocy

1. Wstęp

Problem kodowania stanów wewnętrznych ma niebagatelne znaczenie podczas implementacji automatu skończonego w układzie cyfrowym (np. w strukturze programowalnej). Dzięki zastosowaniu odpowiednich algorytmów istnieje możliwość lepszego wykorzystania struktury, jak również obniżenia poboru mocy przez układ, co jest szczególnie ważne przy konstruowaniu systemów cyfrowych mobilnych, zasilanych bateryjnie, jak również dla zwiększenia wydajności i szybkości systemu.

Moc pobieraną przez układ CMOS można obliczyć z przedstawionego poniżej równania [5]:

$$P_a = \frac{1}{2} \cdot \frac{V_{DD}^2}{T_{cycle}} \cdot N_a \cdot C_a \tag{1}$$

[1] Wydział Informatyki, Politechnika Białostocka, Białystok

gdzie: V_{DD} – napięcie zasilające układ; T_{cycle} – czas, względem którego rozpatrywana jest aktywność przełączania (czas obliczania średniej liczby zmian stanu); C_a – pojemność wyjściowa elementu a; N_a – aktywność przełączania (średnia liczba zmian stanu wyjścia w czasie trwania jednego cyklu T_{cycle}) elementu a.

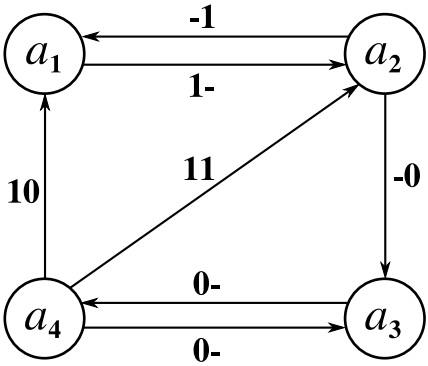
Zmniejszenie mocy dokonuje się poprzez zminimalizowanie wartości występujących po prawej stronie równania (1). Jednakże zmniejszenie V_{DD} oraz C_a wymaga zmiany technologii wykonania układu scalonego. Wartość T_{cycle} wynika z wymaganej prędkości pracy układu (zależy od częstotliwości pracy) i jest dla danej aplikacji stała. Jedynym parametrem, który można zminimalizować w sposób programowy, bez ingerowania w fizyczną budowę układu, jest aktywność przełączania N_a.

W układach sekwencyjnych aktywność przełączania jest ściśle związana ze sposobem kodowania stanów wewnętrznych. Zmiana stanu z a_i na a_j powoduje zmianę wartości kodu stanu zapamiętanego w przerzutnikach. Każda zmiana stanu przerzutnika powoduje wydzielenie mocy. W związku z tym zmiana sposobu kodowania będzie wpływała na aktywność przełączania, a co za tym idzie na ilość pobieranej mocy. Przykładami algorytmów kodowania stanów wewnętrznych, prowadzących do obniżenia poboru mocy są: algorytmy z prac [6] i [4], w których zastosowano wyżarzanie (ang. *annealing*) oraz algorytm z pracy [1], gdzie zastosowano kodowanie kolumnowe (ang. *column-based*).

2. Moc w układach sekwencyjnych

Dany jest automat skończony $F = \{A, X, Y, \phi, \psi, a_1\}$, gdzie A – zbiór stanów wewnętrznych, X – zbiór wektorów wejściowych, Y – zbiór wektorów wyjściowych, ϕ – funkcja przejść, ψ – funkcja wyjść, a_1 – stan początkowy. Dodatkowo zbiór stanów wewnętrznych A składa się ze skończonej liczby M stanów $A = \{a_1, a_2, \ldots, a_M\}$, gdzie M – liczba stanów automatu. Automat można przedstawić graficznie w postaci grafu przejść. Na rys. 1 przedstawiono graf przejść przykładowego automatu skończonego. Węzłami grafu są stany automatu (a_1, a_2, a_3 oraz a_4), natomiast krawędzie odpowiadają przejściom między stanami, które następują po pojawieniu się danego wektora wejściowego (np. przejście ze stanu a_2 do a_3 nastąpi, gdy na wejściu pojawi się wartość 00 lub 10). Na rysunku pominięto przejścia w ten sam stan, gdyż nie powodują one wydzielenia mocy w układzie.

Automaty skończone można opisywać za pomocą dyskretnych łańcuchów Markowa [2], co pozwala obliczać wartości prawdopodobieństw przejścia między stanami. Prawdopodobieństwa statyczne, określające prawdopodobieństwo znalezienia automatu w określonym stanie w chwili $t \to \infty$, można obliczyć korzystając z równań Chapmana-Kołmogorowa [8]. Zakładając zerowy współczynnik korelacji pomiędzy



Rysunek 1. Graf przejść przykładowego automatu skończonego

wartościami prawdopodobieństw wektorów wejściowych dla każdego stanu a_i automatu skończonego $(1 \leqslant i \leqslant M)$ można zapisać układ równań (2):

$$P(a_i) = \sum_{k=1}^{M} P(a_k) \cdot P(I_{ki}) \tag{2}$$

gdzie: $P(a_i)$ – prawdopodobieństwo statyczne stanu a_i; $P(a_k)$ – prawdopodobieństwo statyczne stanu a_k; $P(I_{ki})$ – prawdopodobieństwo pojawienia się wektora wejściowego I_{ki}, który spowoduje przejście automatu ze stanu a_k do stanu a_i [8].

Układ równań (2) jest liniowo zależny (dowolne równanie może zostać wyprowadzone z $M-1$ pozostałych), dlatego jedno z równań należy zastąpić równaniem (3), wynikającym z definicji prawdopodobieństwa:

$$\sum_{i=1}^{M} P(a_i) = 1 \tag{3}$$

Wybór równania, które należy zastąpić równaniem (3), nie wpływa na wynik obliczeń. Rozwiązaniem układu M równań (2) oraz (3) jest wektor prawdopodobieństw statycznych stanów automatu skończonego.

Prawdopodobieństwo zmiany stanu z a_i na a_j, pod warunkiem, że automat znajduje się w stanie a_i, jest zależne od prawdopodobieństwa pojawienia się na wejściu wektora I_{ij}, który spowoduje zmianę stanu z a_i na a_j i można je wyliczyć korzystając z równania (4):

$$P(a_i \rightarrow a_j) = P(a_i) \cdot P(I_{ij}) \tag{4}$$

gdzie: $P(a_i \to a_j)$ – prawdopodobieństwo zmiany stanu z a_i na a_j; $P(a_i)$ – prawdopodobieństwo statyczne stanu a_i; $P(I_{ij})$ – prawdopodobieństwo pojawienia się wektora wejściowego I_{ij}, który spowoduje zmianę stanu automatu z a_i na a_j.

Wyliczone prawdopodobieństwa pozwalają obliczyć aktywność przełączania przerzutników, a co za tym idzie moc wydzieloną w układzie [3].

Graf przejść automatu jest grafem skierowanym, jednakże zmiana stanu automatu z a_i na a_j wymaga przełączenia takiej samej liczby przerzutników, co zmiana stanu z a_j na a_i. Pozwala to przekształcać graf skierowany w nieskierowany. Wagi poszczególnych krawędzi będą miały wartość [1]:

$$w_{i,j} = P(a_i \to a_j) + P(a_j \to a_i) \tag{5}$$

Przykład

Dla grafu przejść z rys. 1 równania (2) będą miały postać:

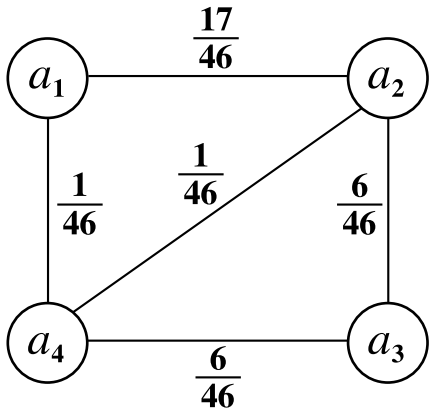
$$\begin{cases} P(a_1) = P(I_{21}) \cdot P(a_2) + P(I_{41}) \cdot P(a_4) \\ P(a_2) = P(I_{12}) \cdot P(a_1) + P(I_{42}) \cdot P(a_4) \\ P(a_3) = P(I_{23}) \cdot P(a_2) + P(I_{43}) \cdot P(a_4) \\ P(a_4) = P(I_{34}) \cdot P(a_3) \end{cases} \tag{6}$$

Jeśli założyć prawdopodobieństwo pojawienia się jedynki na każdym z wejść równe $\frac{1}{2}$, prawdopodobieństwa pojawienia się wektorów wejściowych będą równe: $P(I_{12}) = P(I_{21}) = P(I_{23}) = P(I_{34}) = P(I_{43}) = \frac{1}{2}$ oraz $P(I_{41}) = P(I_{42}) = \frac{1}{4}$. Podstawiając wartości do układu równań (6) oraz zastępując pierwsze równanie równaniem (3) otrzymamy:

$$\begin{cases} P(a_1) + P(a_2) + P(a_3) + P(a_4) = 1 \\ P(a_2) = \frac{1}{2} \cdot P(a_1) + \frac{1}{4} \cdot P(a_4) \\ P(a_3) = \frac{1}{2} \cdot P(a_2) + \frac{1}{2} \cdot P(a_4) \\ P(a_4) = \frac{1}{2} \cdot P(a_3) \end{cases} \tag{7}$$

Rozwiązaniem układu równań (7) są prawdopodobieństwa statyczne automatu: $P(a_1) = \frac{11}{23}$, $P(a_2) = \frac{6}{23}$, $P(a_3) = \frac{4}{23}$, $P(a_4) = \frac{2}{23}$.

Wyliczone wartości pozwalają obliczyć prawdopodobieństwo zmiany stanu. Podstawiając je do równania (4) otrzymujemy: $P(a_1 \to a_2) = \frac{11}{46}$, $P(a_2 \to a_1) = \frac{6}{46}$, $P(a_2 \to a_3) = \frac{6}{46}$, $P(a_3 \to a_4) = \frac{4}{46}$, $P(a_4 \to a_1) = \frac{1}{46}$, $P(a_4 \to a_2) = \frac{1}{46}$, $P(a_4 \to a_3) = \frac{2}{46}$. Wagi krawędzi grafu będą miały wartości: $w_{1,2} = \frac{17}{46}$, $w_{1,4} = \frac{1}{46}$, $w_{2,3} = \frac{6}{46}$, $w_{2,4} = \frac{1}{46}$, $w_{3,4} = \frac{6}{46}$. W związku z tym graf przejść przybierze postać przedstawioną na rys. 2.



Rysunek 2. Graf przejść automatu skończonego przekształcony do grafu nieskierowanego

3. Sformułowanie problemu

Implementując automat skończony w postaci układu sekwencyjnego, każdemu stanowi a_i należy przypisać kod c_i. W związku z tym zbiorowi stanów automatu A odpowiada zbiór C kodów stanów automatu skończonego, $C = \{c_1, c_2, \ldots, c_M\}$. Każdy kod musi być ortogonalny z wszystkimi pozostałym kodami. Liczba bitów kodu N może być wartością z zakresu $[\text{int}\log_2 M, M]$.

Niech c_i^l oznacza l-ty bit kodu stanu a_i. Odległość Hamminga $H(c_i, c_j)$, określona jest jako liczba bitów w kodach c_i i c_j, znajdujących się na tych samych pozycjach mających przeciwne wartości, może być opisana wzorem (8):

$$H(c_i, c_j) = \sum_{l=1}^{N} c_i^l \oplus c_j^l \tag{8}$$

Problem znalezienia sposobu kodowania, który doprowadzi do minimalizacji poboru mocy można przedstawić w postaci zadania całkowitoliczbowego programowania liniowego:

$$Min\left(\sum_{i,j=1}^{M} w_{i,j} \cdot H(c_i, c_j)\right) \tag{9}$$

przy ograniczeniach:

$$\sum_{l=1}^{N} c_i^l \oplus c_j^l \geqslant 1; \forall c_i, c_j; c_i \neq c_j \tag{10}$$

Dokładne rozwiązanie zadania opisanego równaniami (9) i (10) może być niewykonalne, szczególnie dla dużych automatów. Dlatego stosuje się rozwiązania heurystyczne.

4. Algorytmy kodowania stanów wewnętrznych

4.1 Algorytm kodowania kolumnowego

Algorytm kodowania kolumnowego bazuje na pojęciu klasy nierozróżnialności kodów stanów wewnętrznych [1]. Kody stanów mające identyczne kody częściowe należą do tej samej klasy nierozróżnialności (11):

$$c_i, c_j \in C_{k,l} \iff c_i^m = c_j^m, \forall 1 \leqslant m \leqslant l \tag{11}$$

Kodowanie zakończy się sukcesem tylko wtedy, gdy liczba kodów w każdej klasie jest ograniczona zależnością (12):

$$|C_{k,l}| \leqslant 2^{N-l} \tag{12}$$

Wykorzystując klasy nierozróżnialności można zakodować stany wewnętrzne „bit po bicie", bez konieczności wykonywania operacji na całym kodzie. W trakcie przypisywania wartości bitów do kolejnych kodów stanów automatu należy przestrzegać ograniczenia (12).

Poniżej przedstawiono algorytm działający na podstawie opisanych powyżej zależności, bazowany na algorytmie opisanym w [1].

1. Stwórz tablicę przejść, zawierającą pary stanów i wagi grafu odpowiadające parze stanów.
2. Punkty 3-7 wykonaj kolejno dla $l = 1 \ldots N$.
3. Posortuj tablicę przejść w porządku malejącym względem wagi.
4. Dla każdej pary stanów a_i oraz a_j z tablicy przejść wykonaj punkty 5-6.
5. Jeżeli bity kodów c_i^l oraz c_j^l nie zostały przypisane, wykonaj poniższe operacje.
   - Wybierz bit do przypisania w taki sposób, aby po przypisaniu suma wag stanów sąsiednich do a_i oraz a_j, których l-te bity kodów różnią się od l-tych bitów kodów stanów a_i oraz a_j, była jak najmniejsza.
   - Jeżeli przypisanie do obu stanów wybranego bitu nie spowoduje przekroczenia maksymalnych liczności klas, to przypisz tę samą wartość do c_i^l oraz c_j^l. W przeciwnym wypadku przypisz wartości przeciwne.
6. Jeżeli jeden z bitów kodów c_i^l lub c_j^l nie został przypisany, wykonaj poniższe operacje.

- Wybierz bit do przypisania, równy l-temu bitowi w przypisanym kodzie.
- Jeżeli przypisanie wybranego bitu nie spowoduje przekroczenia maksymalnych liczności klasy, to przypisz jego wartość do bitu, który nie został jeszcze przypisany (c_i^l lub c_j^l). W przeciwnym wypadku przypisz wartość przeciwną.

7. Dostosuj tablicę przejść korzystając z wzoru (13).

$$nowa(w_{i,j}) = stara(w_{i,j}) \cdot (H(c_i, c_j) + 1) \tag{13}$$

8. Koniec.

Przykład

Dla grafu z rys. 2 możemy stworzyć następującą tablicę przejść (dla ułatwienia każda waga została pomnożona przez 46, co nie wpływa na wynik kodowania): $\{a_1, a_2, 17\}$, $\{a_1, a_4, 1\}$, $\{a_2, a_3, 6\}$, $\{a_2, a_4, 1\}$, $\{a_3, a_4, 6\}$.

Najpierw kodujemy pierwszy bit kodu. Po posortowaniu tablica przybierze postać: $\{a_1, a_2, 17\}$, $\{a_3, a_4, 6\}$, $\{a_2, a_3, 6\}$, $\{a_1, a_4, 1\}$, $\{a_2, a_4, 1\}$. Bierzemy pierwszą parę stanów z tablicy: a_1 oraz a_2. Ponieważ oba stany nie mają przypisanego pierwszego bitu kodu, wybieramy taki sam bit do przypisania, np. 0 (żadne stany nie mają przypisanego żadnego bitu kodu, więc wartość może być dowolna). Przypisanie bitu do kodów stanów a_1 oraz a_2 nie spowoduje przekroczenia liczności klas (maksymalna liczność klasy na podstawie ograniczenia (12) wynosi 2).

Następnie wybieramy do przypisania parę a_3 oraz a_4. Podobnie jak w poprzednim przypadku, do obu stanów nie zostały przypisane kody, więc wybieramy taką samą wartość bitu do przypisania. Żeby suma wag kodów połączonych z kodami a_3 oraz a_4, których pierwsze bity mają różne wartości, była jak najmniejsza, należy wybrać wartość bitu równą 1. Przypisanie bitu do kodów stanów a_3 oraz a_4 nie spowoduje przekroczenia liczności klas.

Otrzymaliśmy następujące kody cząstkowe: $c_1 = 0$, $c_2 = 0$, $c_3 = 1$, $c_4 = 1$.

Dostosowując tablicę przejść zgodnie ze wzorem (13), otrzymamy nową tablicę: $\{a_1, a_2, 17\}$, $\{a_3, a_4, 6\}$, $\{a_2, a_3, 12\}$, $\{a_1, a_4, 2\}$, $\{a_2, a_4, 2\}$.

Kodujemy drugi bit kodu. Po posortowaniu tablica przybierze postać: $\{a_1, a_2, 17\}$, $\{a_2, a_3, 12\}$, $\{a_3, a_4, 6\}$, $\{a_1, a_4, 2\}$, $\{a_2, a_4, 2\}$. Bierzemy pierwszą parę stanów z tablicy: a_1 oraz a_2. Ponieważ oba stany nie mają przypisanego drugiego bitu kodu, wybieramy taki sam bit do przypisania, np. 0 (żadne stany nie mają przypisanego drugiego bitu kodu, więc wartość może być dowolna). Jednakże przypisanie bitu do kodów stanów a_1 oraz a_2 spowoduje przekroczenie liczności klas (maksymalna liczność klasy wynosi 1). W związku z tym do drugiego bitu kodu stanu a_1 przypisujemy 0, a do kodu stanu a_2 przypisujemy 1.

Następnie wybieramy parę a_2 oraz a_3. Kod stanu a_2 ma już przypisany drugi bit, więc należy przypisać bit do kodu stanu a_3. Do przypisania wybieramy wartość 1 (taką samą, jak drugi bit w kodzie stanu a_2). Takie przypisanie nie spowoduje przekroczenia liczności klas. Na koniec wybieramy parę kodów a_3 oraz a_4. Podobnie jak w poprzednim przypadku, kod stanu a_3 ma już przypisaną wartość. Jednakże wybranie takiego samego bitu (1) spowoduje przekroczenie liczności klasy, więc do przypisania wybieramy wartość 0.

Po zakończeniu kodowania otrzymamy następujące kody stanów: $c_1 = 00$, $c_2 = 01$, $c_3 = 11$, $c_4 = 10$.

4.2 Algorytm sekwencyjny

W algorytmie sekwencyjnym kodowanie zależy od przypisanych uprzednio kodów stanów wewnętrznych automatu. Algorytm wykorzystuje funkcję γ określającą sumę mocy wydzielonej przez automat przy przejściu ze stanu a_i do dowolnego stanu (14):

$$\gamma(c_i) = \sum_{j=1}^{M} w_{i,j} \cdot H(c_i, c_j) \tag{14}$$

Funkcja pozwala wyznaczyć kod przypisywany do stanu, aby wydzielana moc była jak najmniejsza [7].

Niech K^N oznacza zbiór wszystkich wartości kodów stanów o długości N bitów. Liczność zbioru K^N wynosi 2^N. Wartość N należy do przedziału $[\operatorname{int}\log_2 M, M]$.

Poniżej przedstawiono algorytm sekwencyjny kodowania stanów automatu skończonego.

1. $C = \{\emptyset\}$, $K^N = \{k_1, \ldots, k_{2^N}\}$
2. Wybierz dwa stany a_i i a_j, dla których $w_{i,j}$ jest największe.
3. Dla stanów a_i oraz a_j przypisz dowolne kody c_i oraz c_j ze zbioru K^N, dla których $H(c_i, c_j) = 1$. Usuń wartości przypisanych kodów ze zbioru K^N.
4. Powtarzaj kroki 5-6, aż wszystkie stany będą zakodowane.
5. Wybierz ze zbioru A stan a_i, któremu nie przypisano kodu i dla którego suma wag krawędzi grafu łączących stan a_i ze stanami, dla których kody są już przypisane, osiąga wartość maksymalną.
6. Wybierz ze zbioru kodów K^N wartość kodu c_i, dla którego funkcja γ ma wartość minimalną. Usuń wartość kodu ze zbioru K^N.
7. Koniec.

Przykład

Na początku zbiory $C = \{\emptyset\}$ oraz $K^2 = \{00, 01, 10, 11\}$.

Wartość $w_{i,j}$ jest największa dla pary stanów a_1 oraz a_2. Dlatego przypisujemy do nich kody, np. $c_1 = 00$, $c_2 = 01$ (odległość Hamminga dla tej pary kodów wynosi 1). Kody te usuwamy ze zbioru K^2. W związku z tym $C = \{c_1, c_2\}$ oraz $K^2 = \{10, 11\}$

Następnie wybieramy kolejny stan do zakodowania. Dla stanu a_3 suma wag krawędzi grafu łączących stan a_3 ze stanami, dla których kody są już przypisane (czyli a_1 oraz a_2), wynosi $\frac{12}{46}$, natomiast dla stanu a_4 wynosi $\frac{8}{46}$. Do kodowania wybieramy stan a_3.

Ze zbioru K^2 wybieramy kod o najmniejszej wartości funkcji γ. W naszym przypadku będzie to kod 11. Przypisujemy go do kodu stanu a_3 i usuwamy ze zbioru K2. W związku z tym $C = \{c_1, c_2, c_3\}$ oraz $K^2 = \{10\}$

Na koniec do stanu a_4 zostanie przypisany kod 10.

Po zakończeniu kodowania otrzymamy następujące kody stanów: $c_1 = 00$, $c_2 = 01$, $c_3 = 11$, $c_4 = 10$.

4.3 Algorytm rafinacyjny

Algorytm rafinacyjny pozwala „poprawić" wyniki kodowania przeprowadzonego za pomocą innego algorytmu. Jego działanie polega na zamianie kodów stanów w taki sposób, aby zmniejszyć wartość mocy.

Poniżej przedstawiono pseudokod algorytmu rafinacyjnego kodowania stanów wewnętrznych automatu skończonego.

```
bez_zmiany := 0;

REPEAT

  zmiana := FALSE;
  FOR i := 1 TO M
    stan_a := ai;
    FOR kod := 0 TO 2^N - 1
      stan_b := Znajdz_Stan(kod);
      moc_przed := Oblicz_Moc();
      IF stan_b
        Zamien_Kody(stan_a, stan_b);
        IF moc_przed < Oblicz_Moc()
          Zamien_Kody(stan_a, stan_b);
        ELSE
          zmiana := TRUE;
```

```
        ENDIF
      ELSE
        stary_kod :=  stan_a.kod;
        stan_a.kod := kod;
        IF moc_przed < Oblicz_Moc()
          stan_a.kod := stary_kod;
        ELSE
          zmiana := TRUE;
        ENDIF
      ENDIF
    NEXT
  NEXT

  IF zmiana
    bez_zmiany := 0;
  ELSE
    bez_zmiany := bez_zmiany + 1;
  ENDIF

UNTIL bez_zmiany > epsilon;
```

Zastosowane funkcje: `Znajdz_Stan` – wyszukuje stan, któremu przypisano podany kod; `Oblicz_Moc` – zwraca wartość mocy wydzielonej przez układ przy bieżącym kodowaniu; `Zamien_Kody` – zamienia kody stanów.

Wartość `epsilon` jest ustalana przed wykonaniem algorytmu i określa maksymalną ilość przebiegów bez zmiany kodowania.

5. Badania eksperymentalne

Badania przeprowadzono wykorzystując standardowe testy (benchmark) [9]. Obliczenia zostały wykonane przy założeniu następujących wartości: $C = 3$pF, $f = 5$MHz, $V_{DD} = 5$V oraz $P(x_i = 1) = 0{,}5$.

Wyniki zebrane w tab. 1. przedstawiają wyniki badań porównawczych nad algorytmami kodowania kolumnowego oraz sekwencyjnym. Dodatkowo obliczono wartość mocy dla kodowania binarnego (gdzie kod odpowiadał numerowi stanu) oraz „one-hot" (gdzie pozycja jedynki w kodzie odpowiadała numerowi stanu). Kolumna „Benchmark" zawiera nazwę układu testowego, w kolumnie „Stany" umieszczono liczbę stanów układu testowego, kolumna „Kod" określa liczbę bitów kodu, który został wykorzystany w kodowaniu, poza kodowaniem „one-hot", gdzie długość kodu jest równa liczbie stanów automatu. Następne kolumny zawierają wyniki obliczeń dla

czterech sposobów kodowania stanów. Dla każdego sposobu kodowania podano wartości obliczonej mocy („P_X" w mW, gdzie indeks „X" oznacza metodę kodowania: B – binarne, O – one-hot, K – kolumnowe, S – sekwencyjne). Na koniec w wierszu oznaczonym „Średnia" obliczono średnią wartość mocy.

Tablica 1. Wyniki badań eksperymentalnych algorytmów kodowania stanów wewnętrznych

Benchmark	**Stany**	**Kod**	$\mathbf{P_B}$	$\mathbf{P_O}$	$\mathbf{P_K}$	$\mathbf{P_S}$
bbara	10	4	62,14	83,48	56,26	52,77
bbtas	6	3	134,51	166,3	83,15	83,15
beecount	7	3	113,28	169,56	108,92	89,42
dk14	7	3	239,67	308,59	207,28	223,65
dk16	27	5	401,14	360,8	377,53	309,09
dk27	7	3	290,18	375	223,21	223,21
dk512	15	4	298,55	375	319,75	238,84
donfile	24	5	324,22	281,25	265,63	222,66
ex1	20	5	259,04	238,56	157,7	138,55
ex5	9	4	231,89	246,79	176,4	159,29
modulo12	12	4	171,88	187,5	93,75	93,75
opus	10	4	150,2	243,89	133,38	133,38
pma	24	5	252,5	199,3	105,55	104,76
s1	20	5	329,3	274,19	250,74	200,98
s1a	20	5	329,3	274,19	250,74	200,98
s27	6	3	192,75	255,25	168,33	168,33
s8	5	3	62,27	65,65	33,9	33,9
train11	11	4	101,9	124,32	86,96	63,52
Średnia			219,15	234,98	172,18	152,24

Z zestawienia przedstawionego w tab. 1 wynika, że najlepsze wyniki uzyskano po zastosowaniu algorytmu rafinacyjnego (średnia wartość mocy: 152,24), natomiast algorytm kodowania kolumnowego dał średnie wyniki gorsze o prawie 20 mW (średnia wartość mocy: 172,18). Najlepsze wyniki uzyskano przy zastosowaniu kodowania sekwencyjnego dla 17 z 18 testów. W 7 z 18 przypadków najlepsze wyniki osiągnięto przy zastosowaniu kodowania kolumnowego. Najgorsze wyniki dało zastosowanie kodowania „one-hot". Jeśli porównać dwa najlepsze algorytmy (kodowanie kolumnowe i sekwencyjne) okazuje się, że kodowanie sekwencyjne jest lepsze w 11 przypadkach, w 6 daje takie same wyniki, natomiast w 1 gorsze (dk14) w porównaniu do kodowania kolumnowego. Stosunek średniej wartości mocy algorytmu sekwencyjnego do algorytmu kodowania kolumnowego wyniósł 0,88.

W tab. 2 przedstawiono wyniki działania algorytmu rafinacyjnego. Algorytm miał za zadanie poprawić wyniki kodowania: binarnego, kolumnowego oraz sekwencyjnego. Kolumna „Benchmark" zawiera nazwę układu testowego, w kolumnach „Binarne", „Kolumnowe" oraz „Sekwencyjne" zebrano wyniki działania algorytmu rafinacyjnego przy zadaniu początkowego kodowania z użyciem algorytmu odpowiednio: binarnego, kodowania kolumnowego oraz sekwencyjnego. Kolumny: „P_{przed}" zawierają wartość mocy przed wykonaniem algorytmu rafinacyjnego, „P_{po}" zawiera wartość po wykonaniu algorytmu rafinacyjnego, natomiast „P_{po}/P_{przed}" stosunek wartości mocy przed wykonaniem algorytmu rafinacyjnego do wartości po wykonaniu algorytmu rafinacyjnego.

Tablica 2. Wyniki badań eksperymentalnych algorytmu rafinacyjnego

Benchmark	**Binarne**			**Kolumnowe**			**Sekwencyjne**		
	P_{przed}	P_{po}	P_{po}/P_{przed}	P_{przed}	P_{po}	P_{po}/P_{przed}	P_{przed}	P_{po}	P_{po}/P_{przed}
bbara	62,14	55,34	0,89	56,26	53,49	0,95	52,77	52,77	1
bbtas	134,51	83,15	0,62	83,15	83,15	1	83,15	83,15	1
beecount	113,28	89,42	0,79	108,92	89,42	0,82	89,42	89,42	1
dk14	239,67	207,28	0,86	207,28	207,28	1	223,65	207,28	0,93
dk16	401,14	300,95	0,75	377,53	303,24	0,8	309,09	290,41	0,94
dk27	290,18	223,21	0,77	223,21	223,21	1	223,21	223,21	1
dk512	298,55	223,21	0,75	319,75	215,40	0,67	238,84	236,61	0,99
donfile	324,22	226,56	0,7	265,63	214,84	0,81	222,66	207,03	0,93
ex1	259,04	136,43	0,53	157,70	133,29	0,85	138,55	138,55	1
ex5	231,89	163,61	0,71	176,40	159,29	0,9	159,29	159,29	1
modulo12	171,88	93,75	0,55	93,75	93,75	1	93,75	93,75	1
opus	150,2	133,32	0,89	133,38	133,32	1	133,38	133,32	1
pma	252,50	104,76	0,41	105,55	105,18	1	104,76	104,28	1
s1	329,30	204,28	0,62	250,74	208,45	0,83	200,98	198,65	0,99
s1a	329,30	204,28	0,62	250,74	208,45	0,83	200,98	198,65	0,99
s27	192,75	166,23	0,86	168,33	166,23	0,99	168,33	166,23	0,99
s8	62,27	33,9	0,54	33,9	33,9	1	33,9	33,9	1
train11	101,9	63,52	0,62	86,96	63,52	0,73	63,52	63,52	1
Średnia	219,15	150,73	0,69	172,18	149,75	0,9	152,24	148,89	0,99

Z tab. 2 wynika, że algorytm rafinacyjny w większości przypadków (40 z 54) poprawił sposób kodowania uzyskany za pomocą pozostałych algorytmów. Największa poprawa wystąpiła dla kodowania binarnego (0,69 mocy przed wykonaniem algorytmu). Najmniejszą poprawę uzyskano dla kodowania sekwencyjnego (0,99). Jed-

nakże w przypadku każdej metody stosowanie algorytmu rafinacyjnego pozwoliło na poprawę wyników.

Zastosowanie algorytmu rafinacyjnego dało najlepsze wyniki średnie dla kodowania sekwencyjnego (148,89), jednakże niewiele gorsze wyniki uzyskano dla kodowania kolumnowego (149,75). Najgorsze wyniki otrzymano dla kodowania binarnego (150,73), aczkolwiek różnica pomiędzy najlepszym a najgorszym wynikiem (czyli pomiędzy kodowaniem sekwencyjnym a binarnym) nie przekroczyła 2%.

6. Podsumowanie i wnioski

Przeprowadzone badania pokazały znaczne różnice w ilości mocy pobieranej przez układ sekwencyjny, dla którego kodowanie przeprowadzono algorytmem kodowania kolumnowego i algorytmem sekwencyjnym. Wykazano również duży wpływ algorytmu rafinacyjnego na wyniki uzyskane za pomocą analizowanych algorytmów.

Zastosowanie algorytmu sekwencyjnego dało średnio 1,12 razy mniejsze wartości mocy w porównaniu z zastosowaniem algorytmu kodowania kolumnowego. Dodatkowo, przy zastosowaniu algorytmu rafinacyjnego wartość mocy została zmniejszona średnio o 1%.

Działanie algorytmu rafinacyjnego było szczególnie widoczne w przypadku ustawienia jako początkowego kodowania binarnego. W takim przypadku moc zmniejszyła się po zastosowaniu algorytmu średnio do 0,68 wartości przed wynkonaniem algorytmu. Jednakże rezultat różni się niewiele (o ok. 2%) – w zależności od początkowego kodowania.

Najniższą moc średnią osiągnął algorytm rafinacyjny (148,89 mW), który może stanowić alternatywę dla pozostałych algorytmów. Dodatkowe zmniejszenie mocy może zostać osiągnięte m.in. poprzez zwiększenie liczby bitów kodu z zakresu $[\text{int}\log_2 M, M]$.

Literatura

[1] Benini L., DeMicheli G.: State Assignment for Low Power Dissipation, IEEE Journal on Solid-state Circuits, Vol. 30, No. 3 (1995), pp. 259-268.

[2] Freitas A. T., Oliveira A. L.: Implicit Resolution of the Chapman-Kolmogorov Equations for Sequential Circuits: An Application in Power Estimation, Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE) 2003, pp. 10764-10769.

[3] Grześ T., Salauyou V.: Metody obliczania mocy w układach cyfrowych, „Pomiary, Automatyka, Kontrola" nr 7bis (2006), str. 101-102.

[4] Koegst M., Franke G., Feske K.: State Assignment for FSM Low Power Design, Proceedings of the Conference on European Design Automation, Geneva 2003, pp. 28-33.
[5] Pedram M.: Power simulation and estimation in VLSI circuits, "The VLSI Handbook", Edited by W-K. Chen, The CRC Press and the IEEE Press, 1999.
[6] Roy K., Prasad S. C.: Circuit Activity Based Logic Synthesis for Low Power Reliable Operations, IEEE Transactions on VLSI Systems, Vol. 1, No. 4 (1993), pp. 503-513.
[7] Salauyou V., Grzes T.: FSM state assignment methods for low-power design, Proceedings of 6th International Conference on Computer Information Systems and Industrial Management Applications (CISIM'2007), IEEE Computer Society, pp. 345-348.
[8] Tsui C.-Y., Monteiro J., Pedram M., Devadas S., Despain A. M., Lin B.: Power Estimation Methods for Sequential Logic Circuits, IEEE Transactions on VLSI Systems, Vol. 3, No. 3 (1995), pp. 404-416.
[9] Yang S.: Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0, Technical Report, Microelectronics Center of North Carolina, 1991, 43 p.

FINITE STATE MACHINES STATE ASSIGNMENT ALGORITHMS FOR POWER MINIMIZATION

Abstract: State assignment for a finite state machine (FSM) is an important process in logic synthesis of the sequential circuits in programmable devices. Using the proper algorithm provides among other things the reduction of the power dissipation. In this paper we focused on the algorithms that reduce power dissipation. The analysis of the column based algorithm (described in [1]) as well as two algorithms proposed by authors: sequential [7] and iterational was made. Experiments were made on standard benchmarks, researched in Microelectronics Center of North Carolina [9]. Obtained results showed significant reduction of the power dissipation when using the sequential algorithm (12% in comparison with the column-based algorithm). Iterational algorithm improves the results by additional 1%.

Keywords: finite state machine, state assignment, low-power design

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Anna Łupińska–Dubicka,[1] Marek J. Drużdżel[1,2]

ANALYZING CERTAIN TEMPORAL DEPENDENCES IN NETFLIX DATA

Abstract: Netflix (see http://www.netflix.com/), an American Internet-based movie rental company, uses data mining in their recommendation system. In October 2006 Netflix made a huge data base of their users and movie evaluations available to the community and announced a million dollars prize to the team that beats the accuracy of their recommendations by at least 10%. The data have since become an object of interest of the machine learning community. In this paper, we focus on one aspect of the data that, to our knowledge, has been overlooked — their temporal dependences. We have looked at the impact of the day of the week, month of the year, length of membership, month from the start of Netflix, etc., on the average evaluation.

Keywords: Data analysis, temporal dependences, Netflix

1. Introduction

Netflix (see http://www.netflix.com/), an American Internet-based movie rental company, offers its over 6.7 milllion subscribers access to over 85,000 DVD titles plus a growing library of over 4,000 full-length movies and television episodes that are available for instant watching on their PCs. Based on a user's viewing pattern and evaluations of previously watched movies, their user interface recommends to their clients movies that they might like as well. These recommendations, if reasonably accurate, are a valuable source of information to Netflix users.

Netflix recently announced a million dollar prize to the team that beats the accuracy of their recommendations by at least 10%. For this purpose, the company has made a huge subset of their movie evaluations database available to the community. This database contains over 500,000 users (out of roughly 2.5 million) and over 100 million evaluations. As of the start of the contest, the square root of the mean square error (RMSE) of Cinematch, the company's recommendation system, was 0.9525

[1] Faculty of Computer Science, The Bialystok Technical University, Bialystok, Poland
[2] Decision Systems Laboratory, School of Information Sciences, University of Pittsburgh, Pittsburgh, Pensylwania, USA

(see http://www.netflixprize.com/). Over 20 thousand teams have joined the competition so far and almost 700 teams have beaten Netflix, although none of them has beaten Netflix accuracy by more than 10% as of December 2007. 90 teams have improved the accuracy of Netflix predictions by more than 5%.

The Netflix data set is at the moment one of the most comprehensive and most challenging data sets available to the machine learning community. Many top machine learning and data mining teams have studied it. The best team participating in the competition as of December 2007, has beaten Netflix accuracy by 8.5%. The remaining 1.5% is fairly hard to overcome. To improve the accuracy of predictions, any source of information, even one that is weak, can play a role. We focus on one aspect of data that, to our knowledge, has been overlooked — temporal dependences among the measured variables. Every evaluation (of over 100 million evaluations available in the data) has a time stamp. Surprisingly, it turns out that this variable is quite depend on how people rate movies. We have looked at the impact of the day of the week, month of the year, length of membership, month from the start of Netflix, etc., on the average evaluation and found that incorporating this information improves the quality of predictions.

2. The data sets

Netflix has collected over 1.9 billion ratings from more than 11.7 million subscribers and over 85 thousand titles since October 1998 and has shipped over 1 billion DVDs. It receives over 2 million ratings per day. For the competition, the company provided over 100 million ratings from over 480 thousand randomly-chosen, anonymous subscribers over nearly 18 thousand movie titles. The date of each rating is also included. The data were collected between October 1998 and December 2005 and reflect the distribution of all ratings received by Netflix during this period. The ratings are on a scale from 1 to 5 and from the point of view of a user, are pictured as a collection of integral stars. To protect customer privacy, each customer id has been replaced in the data by a randomly-assigned id. Figure 1 shows the cumulative number of Netflix customers as a function of time in the Netflix data set. Assuming that the sample is representative, the figure gives an idea of the growth of the company in terms of new customers [3].

The contestants are also given the title and the year of release of each movie. The title is the Netflix movie title, year of release can range from 1890 to 2005 and typically, although not consistently, corresponds to the release year of the movie.

The complete data set consists of two separated subsets: the training set and the qualifying set. The ratings from the training set were released to contestants while

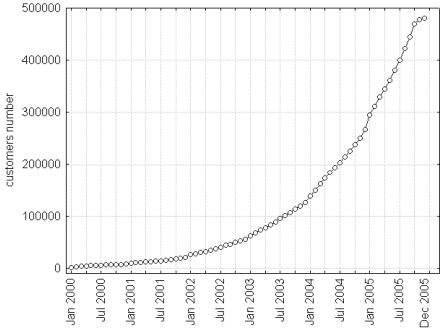


Fig. 1. Cumulative number of customers as a function of time in the Netflix data

the qualifying ratings were withheld and form the basis of the contest scoring system. Contestants are required to make prediction for all 3 million withheld ratings in the qualifying data set. Testing contestants' results on the most recent ratings reflects Netflix's business goal of predicting future ratings based on past ratings. The RMSE is computed automatically and reported to the contestants [4].

3. Differences between the probe and the training data sets

It is critical for analyzing the Netflix data to realize that they consist of two data sets with quite different statistical properties. We will review briefly the method by which the two data sets were obtained.

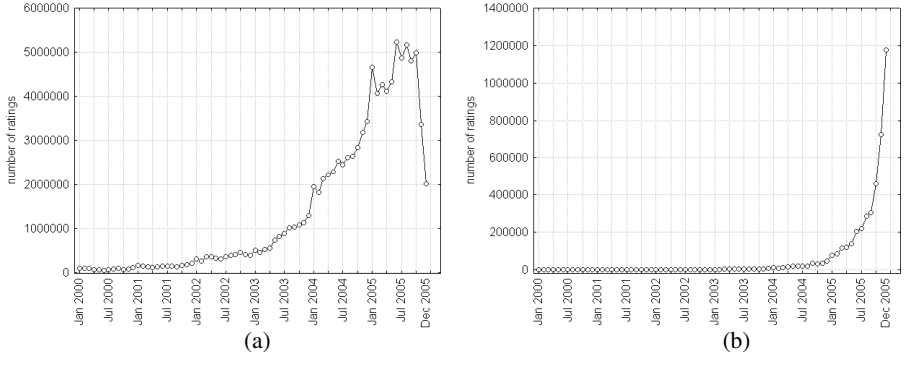


Fig. 2. Number of ratings as a function of time in the training data set without probe subset (a) and in the probe and qualifying data sets (b)

The complete data set consists of two separated subsets: the training data set and the qualifying data set. These sets were created by randomly selecting a subset of all users who provided at least 20 ratings between October 1998 and December 2005. Then the most recent ratings of these users were randomly assigned, with equal probability, to three subsets: quiz, test, and probe. The qualifying data set comprises the quiz and test subsets. The training data set was created from all remaining ratings and the probe subset [1].

Figure 2 shows the distribution of the number of ratings in the training data set (excluding the probe subset) and in the qualifying and probe data sets combined. It is clear that most of the ratings in the qualifying and the probe data sets are very recent.

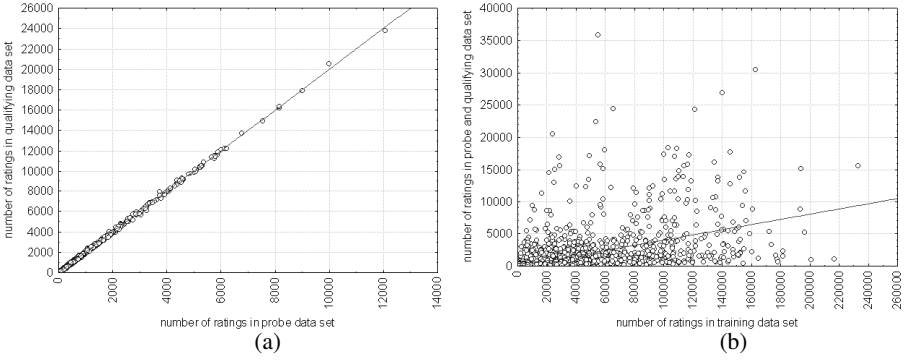


Fig. 3. The relations between the number of ratings per movie in the probe and qualifying data sets (a) and in the training data set and in the probe and qualifying data sets (b)

Please note that the number of ratings per movie in Figure 3 indicates that the ratings in the probe and qualifying subsets are not randomly selected from the training data set. In Figure 3-a, the scatterplot shows the relation between the number of ratings per movie in the probe and qualifying data sets. In Figure 3-b, it is noticeable that the relationship between the number of ratings per movie in the training data set and in the probe and qualifying data sets is much weaker, if at all.

Similarly, Figure 4 shows that the probe and the training data sets have different average ratings for both movies and users. It seems that the most recent movies (i.e. those in the qualifying and probe data sets) tend to get higher ratings. The average rating per user in the training data set is a little bit smaller and has smaller variance, which is the result of a small number of data

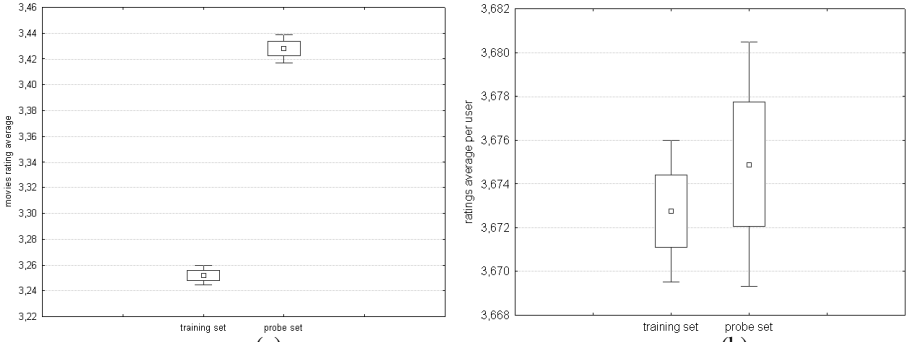


Fig. 4. Average ratings per movie (a) and per user (b) in the trainings and probe data sets

4. Movie age

We found that older movies tend to get better user ratings than new movies (Figure 5-a). This could be explained by the fact that most movies rented by Netflix are in DVD format. While there are fewer older than there are new movies, those that have been published in DVD may simply belong to masterpieces and, hence, effectively get high ratings. Virtually, all new movies are published in the DVD format and, as is usually the case, only a fraction of them are masterpieces. Hence, the average rating is lower.

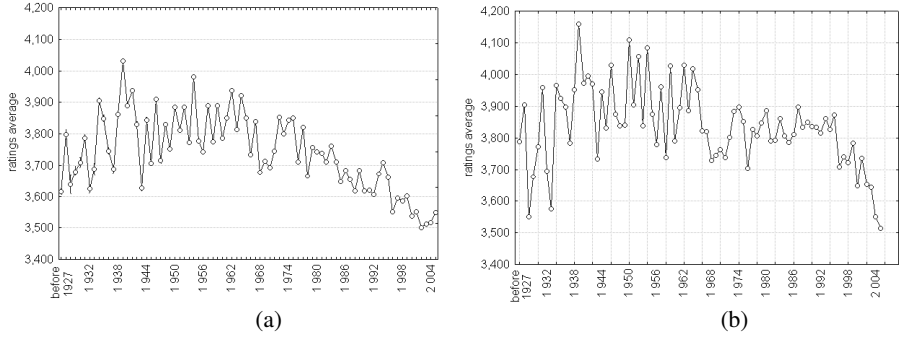


Fig. 5. Average rating as a function of movie's production year in training (a) and probe (b) data set

Comparing Figure 5-a and Figure 5-b, it is not difficult to see that this trend is still observable in the probe subset. The movies from '40s, '50s, and '60s receive higher ratings than these produced nowadays.

5. Netflix Age

Movies tend to get higher ratings as time goes by (see Figure 6). This may have to do with a varying population of Netflix customers. For example, it is well known that technology-based companies attract in the beginning so called "early adopters". Mainstream customers, typically more conservative than the "early adopters," get interested only later. The difference in average evaluation can possibly be explained by the difference between these two groups of customers.

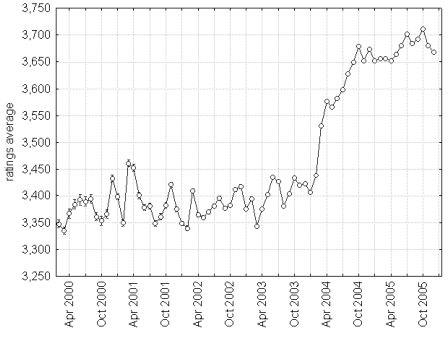


Fig. 6. Average rating as a function of Netflix's age

It is quite interesting to observe a sudden increase (about 0.15 point) in average ratings between January and April, 2004. To our knowledge, this increase was associated neither with a significant increase in the number of Netflix customers (see Figure 1), nor with the number of ratings (see Figure 2).

6. Weekly variations

We were trying to investigate why there is a difference in average rating among various days of the week. One might think that it may have to do with varying characteristics of the populations that rate on the weekends and those that rate on weekdays

– weekend customers, for example, may be more relaxed and less critical, which results in higher ratings. And in this case, confronting comeback to daily business may cause a more critical attitude and a decrease in ratings.

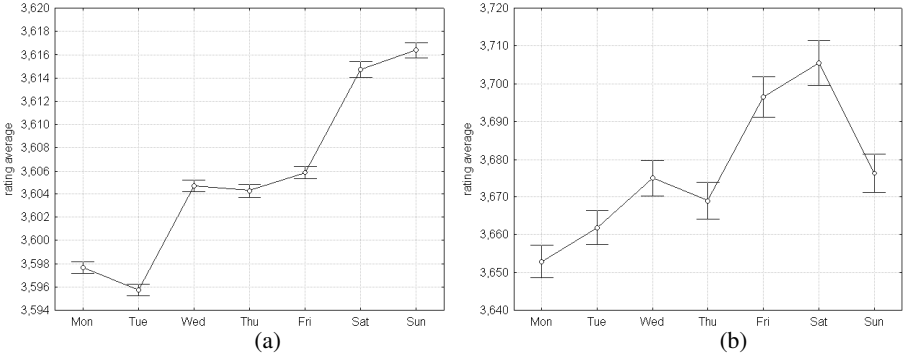


Fig. 7. Average rating as a function of day of the week in training (a) and probe (b) data set

We also looked at the number of ratings on each day of the week (see Figure 8). Almost twice as many customers rate on Mondays and Tuesdays than on weekends. This may have to do with the viewing attern – quite likely more movies are watched on weekends. Watched movies are evaluated and new movies are ordered after the weekend viewing.

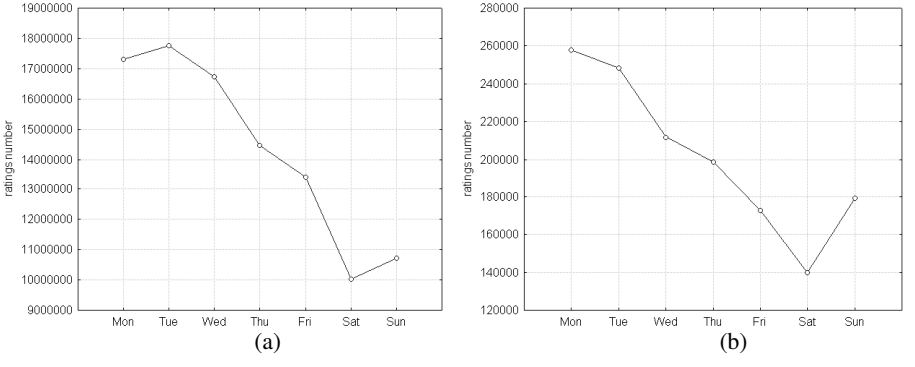


Fig. 8. Number of ratings as a function of day of the week in training (a) and probe (b) data set

We believe that while there could be a correlation between the day of the week, the customer's mood and the ratings he gives, the observed differences are rather due to individual differences among customers. Some customers typically vote on weekdays, other on weekend – probably they are the majority, as shown in Figure 8. It is also possible that those who rate on weekends, rate higher than those who rate on weekdays.

To check if there is a relationship between ratings of particular user and day of week we chose randomly a few customer with different number of ratings. Figure 9 (custmers who rated more than 16 thousand times), Figure 10 (custmers who rated about three thousand times), and Figure 11 (custmers who rated about 500 times) show their average rating and the number of ratings as a function of the day of the week. We observed that both average rating and the number of ratings falling on the day of the week is very a individual feature.

For example customer 305344 (Figure 9a, c) rates the most his movies on Fridays but gives in this day the lowest evaluations. The highest ratings he gives on Sundays and Mondays. In his case the average rating is the lower the more he votes. Customer 2439493 (Figure 9b, d) is active the most in the middle of the week, his ratings are going up and down. Customer 981753 (Figure 10a, c) rates a lot on Tuesdays but similarly to customer 305344, the more ratings the lower they are. Customer 1092521 (Figure 10b, d) is an example of a user who votes mainly once a week – on Saturday, giving at that time the highest ratings. Again, customer 912242 (Figure 11a, c) rates for the most part on Tuesdays and his ratings are then the lowest. Customer 1187552 (Figure 11b, d) rates the movies almost at the same level, independently of the day of the week. None of his ratings was given on Saturdays; Sunday, Tuesday and Wednesday are also days with almost zero activity.

It looks like there are several types of customers. One of them prefer to vote on weekdays giving then lower ratings, the second like rating on weekends and their evaluations are higher, the others give the more critical ratings the more frequently they rate, regardless of the day of the week. In case of the whole Netflix customers' population, the singular influence of these different groups can cancel each other out. Finding these groups of users, similar to each other with respect to the number of ratings and the rating average, could help to benefit from this kind temporal dependency.

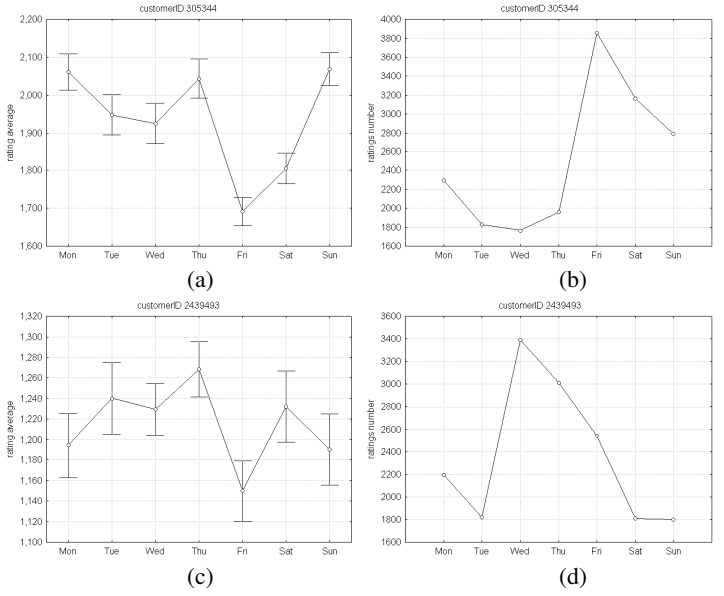


Fig. 9. Average rating (a, c) and the number of ratings (b, d) as a function of the day of the week for two customers who rated more than 16 thousand times

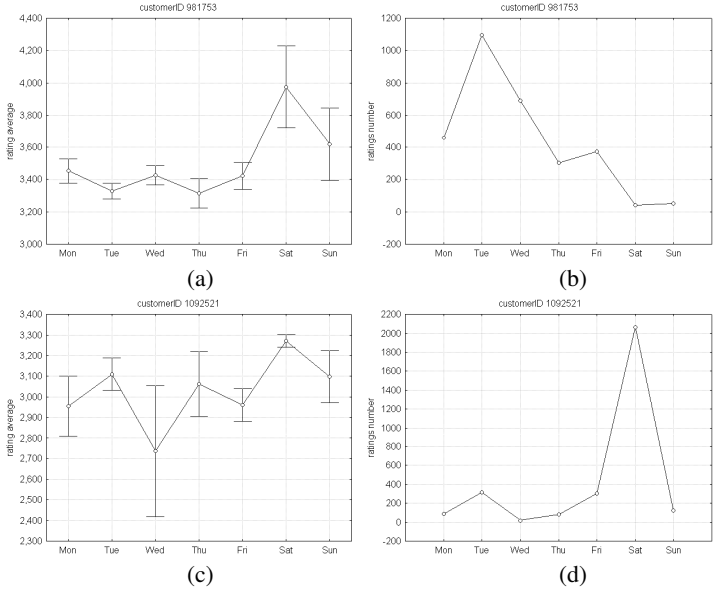


Fig. 10. Average rating (a, c) and the number of ratings (b, d) as a function of the day of the week for two customers who rated about three thousand times

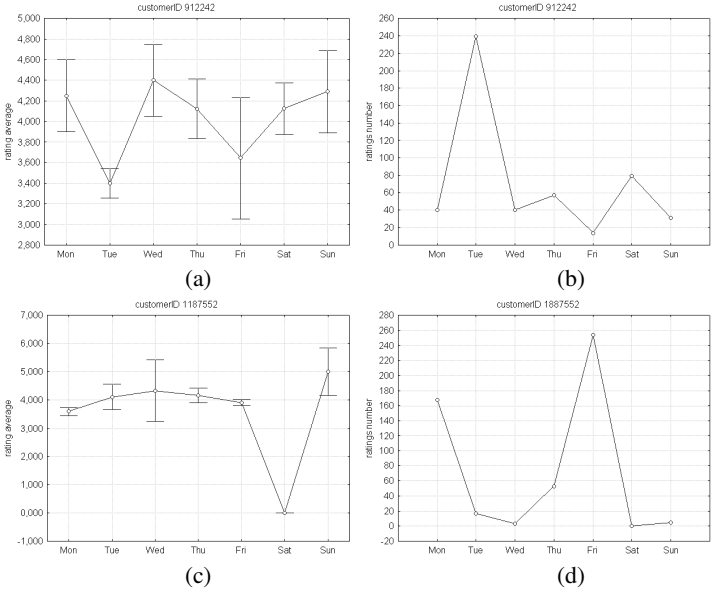


Fig. 11. Average rating (a, c) and the number of ratings (b, d) as a function of the day of the week for two customers who rated about 500 times

7. Customer behavior as a function of the length of membership

Customers tend to give higher ratings in the beginning of their membership. This goes down within the first year but picks up again somewhat after a few years (see Figure 12).

A probable cause for this trend is that in the beginning of their membership customers choose movies they have seen in movie theaters, liked, and desired to see again. As time goes by, they run out of movies that they want to watch again or that were recommended by friends or their families. Effectively, their rental may become more accidental, based on less strong recommendations and desire to watch, which may entail a decrease in average rating.

The last 2–3 years in Figure 12 show a large variation in ratings. It is quite likely the effect of a relatively small sample size – the number of customers who have been Netflix's members for more than 4 years is rather small.

To verify this, we changed the scale on the time axis and plotted the average rating as a function of lenght membership in reverse order, i.e. point 0 shows the most recent ratings, point 1 ratings from month before, etc. It seems like the average rating is going up with time (i.e., going down in reverse time scale).

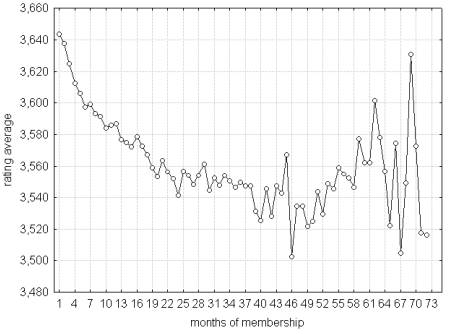


Fig. 12. Average rating as a function of a customer's membership age

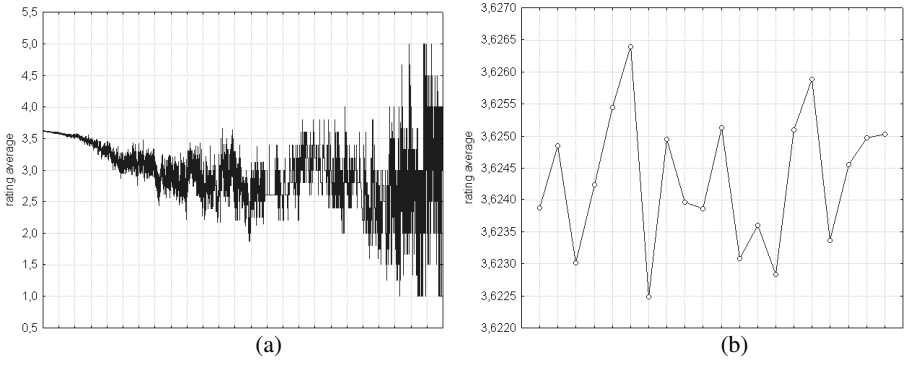


Fig. 13. Average rating as a function of rating number in order from the last rating to the first – all ratings (a) and the last 20 (b)

8. Experimental study

As mentioned before, Netflix provided two separated data sets: the training set and the qualifying set for the competition. While movie ratings for the qualifying set are not known, it is possible to test rating algorithms on the "probe" subset of the training data, identified by Netflix and statistically similar to the qualifying set. In our previous work [2], we removed the probe subset from all available ratings and were trying to guess the ratings from the "probe" subset. In this paper, we concentrated on predicting the ratings from the qualifying data set. First we made a prediction on the basis of all data in the trainig data set, then we checked if leaving only most recent data (provided in the probe subset) could improve the results.

There are many ways in which temporal dependences can be used in guessing the rating. In this paper we use a few simple techniques, such as movie average and user average, just to illustrate our point that knowledge of temporal dependences can reduce the guessing error.

8.1 Prediction based on customer's average rating from the training data set

RMSE = 1.0655

$$\text{prediction} = \text{cAVG}\,, \tag{1}$$

where the variable cAVG means average of all customer's ratings.

8.2 Prediction based on movie's average rating from the training data set

RMSE = 1.0536

$$\begin{aligned} &\text{if } (\text{cStdDev} < 0.01)\ \text{prediction} = \text{cAVG} \\ &\text{else prediction} = \text{mAVG}\,, \end{aligned} \tag{2}$$

where variable cStdDev stands for standard deviation of all customer's ratings, variable cAVG means average of all customer's ratings, variable mAVG is equal to average of all movie's ratings.

8.3 Prediction based on weekly variations of customer's average from the training data set

RMSE = 1.1266

$$\text{prediction} = 0.9\ \text{cAVG} + 0.1\ \text{cDayOfWeekAVG}\,, \tag{3}$$

where variable cDayOfWeekAVG denotes average of all customer's ratings on a given day of the week.

8.4 Prediction based on weekly variations of movie's average from the training data set

RMSE = 1.0525

$$\begin{aligned} &\text{if } (\text{cStdDev} < 0.01)\text{prediction} = \text{cAVG} \\ &\text{else prediction} = 0.9\ \text{mAVG} + 0.1\ \text{mDayOfWeekAVG}\,, \end{aligned} \tag{4}$$

where variable mDayOfWeekAVG stands for average of all ratings that this movie has been given on a specific day of the week.

8.5 Prediction based on customer's average rating from the probe data set

RMSE = 1.0672

$$\begin{aligned} &\text{if } (\text{cNoRatingsProbe} > 5) \text{ prediction} = \text{cAVGfromProbe} \\ &\text{else prediction} = \text{cAVG} , \end{aligned} \tag{5}$$

where the variable cNoRatingsProbe means the number of ratings this customer contained in the probe subset, variable cAVGfromProbe denotes average of customer's ratings from the probe subset, variable cAVG stands for average of all customer's ratings (from the training and probe data sets).

8.6 Prediction based on movie's average rating from the probe data set

RMSE = 1.0490

$$\begin{aligned} &\text{if } (\text{mNoRatingsProbe} > 7) \text{ prediction} = \text{mAVGfromProbe} \\ &\text{else if } (\text{cStdDev} < 0.01) \text{ prediction} = \text{cAVG} \\ &\text{else prediction} = \text{mAVG} , \end{aligned} \tag{6}$$

where the variable mNoRatingsProbe denotes the number of ratings this movie from the probe subset, variable mAVGfromProbe stands for average of movie's ratings contained in the probe subset, variable cStdDev is equal to standard deviation of all customer's ratings, variable mAVG stands foraverage of all movie's ratings (from the training and probe data sets).

8.7 Prediction based on weekly variations of customer's average from the probe data set

RMSE = 1.1418

$$\begin{aligned} &\text{if } (\text{cDayOfWeekNoRatingsProbe} >= 5) \\ &\text{prediction} = 0.9\,\text{cAVG} + 0.1\,\text{cDayOfWeekAVGfromProbe} \\ &\text{else if } (\text{cDayOfWeekNoRatings} > 0) \\ &\text{prediction} = 0.9\,\text{cAVG} + 0.1\,\text{cDayOfWeekAVG} \\ &\text{else prediction} = \text{cAVG} , \end{aligned} \tag{7}$$

where variables cDayOfWeekNoRatingsProbe and cDayOfWeekNoRatings denote the number of customer's ratings on a given day of the week in the probe data and

training (including probe subset) data sets respectively, variables cDayOfWeekAVGfromProbe and cDayOfWeekAVG stand for the average of all customer's ratings on a given day of the week in the probe data and training (including probe subset) data sets respectively, variable cAVG means average of all customer's ratings, independently of the day of the week.

8.8 Prediction based on weekly variations of movie's average from the probe data set

RMSE = 1.0525

$$\begin{aligned} &\text{if } (\text{cStdDev} < 0.01) \text{prediction} = \text{cAVG} \\ &\text{else if } (\text{mDayOfWeekNoRatingsProbe} >= 5) \\ &\text{prediction} = 0.9\ \text{mAVG} + 0.1\ \text{mDayOfWeekAVGfromProbe} \\ &\text{else if } (\text{mDayOfWeekNoRatings} > 0) \\ &\text{prediction} = 0.9\ \text{mAVG} + 0.1\ \text{mDayOfWeekAVG} \\ &\text{else prediction} = \text{mAVG}\ , \end{aligned} \tag{8}$$

where variable cStdDev is equal to standard deviation of all customer's ratings, variable mDayOfWeekNoRatingsProbe and mDayOfWeekNoRatings stand for the number of movie's ratings that this movie has been given on a specific day of week in the probe data and training (including probe subset) data sets respectively, variable mDayOfWeekAVGfromProbe and mDayOfWeekAVG denote the average of ratings that this movie has been given on a specific day of week in the probe data and training (including probe subset) data sets respectively, variable mAVG is equal to the average of all movie's ratings, independently of the day of the week.

9. Concluding remarks

Recommender systems are programs and algorithms that measure user interest in given items or products to provide personalized recommendations for items that will suit the user's taste. More broadly, recommender systems attempt to profile user preferences and model the interaction betwen users and products. One possible strategy for building such systems relies only on past user behaviour, without requiring creation of explicit profiles.

Netflix, an on-line movie subscription rental service, allows people to rent movies for a fixed monthly fee, maintaining a prioritized list of movies they wish to view.

The company encourages subscribers to rate the movies that they watch, expressing an opinion about how much they liked (or disliked) each movie. To predict the customer's rating of the movie, the Netflix recommendation system, Cinematch analyses the past ratings using a variant of Pearson's correlation. In October 2006, the company released a large data set of movie-ratings and challenged the data mining, machine learning, and statistical communities to develop systems that could improve the accuracy of Cinematch. At the start of the contest Cinematech's RMSE was 0.9525. One year later, in December 2007, the best team has achieved a 8.5% improvement over this score. It seems that the remaining 1.5% is fairly hard to overcome.

Given the current impasse and lack of significant progress, it is clear that every, even a weak source of information about customers' preferences can play an important role in improving the predictions. In previous paper [2], we presented certain observations that focused on temporal dependencies in Netflix' data. We have shown that there is a strong dependence of the average rating on the the day of week, Netflix' age and length of a customer's membership.

Our next step was to focus on the differences between the probe subset and the rest of the training data set. Both, the probe and the qualifying subsets were created from the most recent ratings and are very different from the training data set and, as it was shown, they have similar properties. It seems that our idea that the most recent ratings can provide much more information about future votes was good direction. When predictions are based on the movies' average ratings only from the probe subset, the RMSE is going down. Some problems appeared when we tried to guess with basis on the customers' average ratings. The RMSE was growing up when we confined to the probe subset. The possible explanation can be relatively small size of the probe subset in comparison with the number of rating contained in the training data set. When we are taking account of movies' average, the number of ratings per movie is enough to cause the decrease of RMSE. But considering customers' average one might notice that the proportion of 1.5 million of ratings in the probe subset and to 480 thousand of customers is too small to receive reliable results. We believe that enlargement the probe subset with maintenance its properties, such as number of ratings per movie and per user, should bring improvement of the accuracy of predictions based on the customer's preferences.

Bibliography

[1] Bennett, J. Lanning, S.: The Netflix Prize, Proceedings of KDD Cup and Workshop 2007, Aug. 12, 2007.

[2] Łupińska–Dubicka, A. Drużdżel, M.J. Temporal Aspects of Netflix Data, 2007.

[3] http://www.netflix.com
[4] http://www.netflixprize.com
[5] http://www.netflixprize.com/community

ANALIZA WYBRANYCH ZALEŻNOŚCI CZASOWYCH W DANYCH NETFLIX

Streszczenie: Działający w Stanach Zjednoczonych Ameryki Netflix (http://www.netflix.com/) jest jedną z największych na świecie internetowych wypożyczalni filmów. W celu uzyskania wyższej jakości proponowanych przez system ocen filmów, w październiku 2006 roku Netflix udostępnił bazę danych użytkowników oraz ich ocen i ogłosił nagrodę dla tego, kto uzyska co najmniej 10-cio procentową poprawę w stosunku do wyników Cinematch (RMSE=0.9525). W tym artykule postawiliśmy sobie za cel zbadanie, czy zależności czasowe, takie jak dzień tygodnia lub długość członkostwa, są w stanie zwiększyć jakość prognozowania.

Słowa kluczowe: Zależnosci czasowe, Netflix

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Walenty Oniszczuk[1], Maja Joanna Podobińska[2]

MODELOWANIE OBSŁUGI ZADAŃ W SERWERACH Z OSCYLACJAMI

Streszczenie: Idea przedstawianego systemu kolejek z oscylacjami jest oparta na dwóch progowych wartościach. Obsługa procesu w tym systemie jest zorganizowana, w przybliżeniu, w ten sposób, że długość kolejki utrzymuje się pomiędzy tymi wartościami. System kolejki z oscylacjami pozwala lepiej wykorzystywać dostępne zasoby i jest stosowany w wielu urządzeniach, które korzystają z obsługi pojedynczej kolejki. Jest to również uogólnianie niektórych procedur zaproponowanych dla sieci ATM (ang. Asynchronous Transfer Mode). W tej pracy rozważać będziemy systemy kolejkowe z oscylacjami w wersji ze skończonym buforem. Charakterystyki stanów systemów z procesem Poissona na wejściu (M/G-G/1/N) otrzymuje sie metodą potencjałów. To podejście daje przejrzyste i łatwe do implementacji formuły matematyczne.

Słowa kluczowe: kolejki z oscylacjami, metoda potencjałów, sieci ATM

1. Wstęp do kolejek oscylacyjnych z buforem

System kolejkowy z oscylacjami [2] jest definiowany w następujący sposób. Wiadomość przybywa w momencie $\tau_i, i = 1, ...,$ gdzie $\tau_{i+1} - \tau_i$ są niezależne oraz $P(\tau_i - \tau_{i-1} < x) = G(x)$

Niech $a, b \in \mathbb{N}$ oraz $a < b$ i $X(t)$ oznacza liczbę zadań w systemie w momencie t, włącznie z zadaniem, które jest bieżšco obsługiwane przez system. Zakładamy, że $X(0) = n$, gdzie $n \in [0, b)$. Początkowo system działa standardowo jak G / G / 1 z czasem obsługi wyznaczonym przez dystrybuantę funkcji $F_1(x)$.

Niech $\tau_b(1)$ będzie pierwszym momentem, w którym długość kolejki osiąga wartość b, np. $\tau_b(1) = \inf\{t > 0 : X(t) = b\}$. W momencie $\tau_b(1)$ dystrybuanta czasu obsługi zmienia się na $F_2(x)$. Zgłoszenie obsługiwane w momencie $\tau_b(1)$ jest obsługiwane drugi raz tylko, że z dystrybuantą $F_2(x)$. Nowa dystrybuanta czasu obsługi obowiązuje aż do momentu $\tau_a(1) = \inf\{t > \tau_b(1) : X(t) = a\}$, gdy to nastąpi, zmienia się w poprzednią i funkcjonuje do momentu $\tau_b(2) = \inf\{t > \tau_a(1) : X(t) = b\}$

[1] Wydział Informatyki, Politechnika Białostocka, Białystok
[2] Dyplomantka Wydziału Informatyki Politechniki Białostockiej

itd. Zatem zmiana dystrybuanty nast.ępuje na poziomie a i b. Załóżmy również, że bufor ma rozmiar $\mathbf{N}$ (włączając stanowisko obsługi np. serwer) i $\mathbf{N} \geq b$. Oznacza to, że jeśli zgłoszenie przychodzące zastanie system z $\mathbf{N}$ innymi zgłoszeniami, to przychodzące zgłoszenie zbedzie stracone. Jeśli $X(0) \geq b$, zmiany w tym opisie są oczywiste.

Jedna z motywacji do badania systemów z oscylacjami jest następująca: gdy projektujemy system z pojedynczą kolejką do obsługi, zwykle oczekujemy, aby kolejka nie była zbyt długa (np. aby uniknąć przepełnienia bufora i wysokiego współczynnika straty). Rozwiązaniem jest wysoka wydajność serwera, ale oznacza to wysoki koszt obsługi. Ponadto, nie możemy wykorzystać najefektywniej serwera, biorąc pod uwagę tylko jego wydajność, ponieważ bezczynne okresy są długie. Zatem jest to marnowanie zasobów. Jest to dobrze znany efekt z teorii kolejek, gdzie krótkie kolejki wymagają małego obciążenia ρ (nasilenie ruchu), podczas gdy krótkie okresy bezczynności wymagają dużego ρ. Wymagania te nawzajem się wykluczają. Problem, aby mieć krótkie kolejki oraz krótkie okresy bezczynności nie może być nareaz rozwiązany przez standardowy pojedynczy model obsługi G / G / 1.

Jednym z rozwiązań powyższego problemu jest system ze zmienną intensywnością obsługi, np. zmienna obsługa zależy od długości kolejki. Taki system jest matematycznie skomplikowany, ale i skomplikowany w praktycznej realizacji.

Inne rozwiązanie oparte jest na wartościach progowych (wiele schematów kontroli ruchu w sieciach ATM opiera się na podstawowych progach systemów kolejkowych). Na przykład w pracach [7] oraz [8] autorzy uporali się z pojedynczym systemem obsługi w następujący sposób. Zgłoszenie przybywa do kolejki z rozkładem Poissona. Jeżeli długość kolejki w momencie zapoczątkowanej usługi klienta jest mniejsza niż próg $\mathbf{L} \in \mathbb{N}$ (kolejno, większa lub równa niż próg $\mathbf{L}$), to czas obsługi klienta ma rozkład F_1 (kolejno, F_2). Te wywody były początkowo motywowane przez transmisje głosowych pakietów w sieciach ATM. Obsługa w tej sytuacji oznacza transmisję, a przez przybliżoną i dokładną transmisję głosową uzyskuje sie dwa różne czasy obsługi. Systemy z oscylacjami rozwiązują wspomniany problem jak również uogólniają idee Choi [7].

Wprowadzenie dwóch poziomów progowych w zasadzie nie komplikuje praktycznego wykonania systemu, ale dodatkowo pozwala mieć lepszą kontrolę nad długością kolejki. W modelach M / G-G / 1 można przedstawić charakterystyki systemu z oscylacjami z procesem Poissona na wejściu i nieskończonym buforem [1]. Ponieważ nieskończony bufor nie istnieje, zatem z powodów praktycznych ważniejsze jest rozpatrzenie wersji ze skończonym buforem. Wzwiązku z tym bedzie przeprowadzona wszechstronna analiza tego przypadku. Mianowicie, dystrybuante

stacjonarnš długości kolejki dla systemu z procesem Poissona na wejściu (M / G-G / 1 / N) otrzyma sie i zaprezentuje w przejrzystych formułach.

Będziemy potrzebowali tylko założyć, że wartości oczekiwane dla dystrybuant $G(x), F_1(x), F_2(x)$ są skończone $(< +\infty)$

2. System M/G-G/1/N

Zakładamy, że $G(x) = 1 - e^{-\nu x}$. Naszym celem jest znalezienie granicy

$$\lim_{t\to\infty} P_n(X(t) = l) = P_l \tag{1}$$

(indeks n przy P_n oznacza, że $X(0) = n$). Fakt, że ta granica istnieje dla każdego $l = 0, 1, \ldots N$ i że nie zależy od n może być w prosty sposób udowodniony przez odwołanie do teorii systemów kolejkowych z oscylacjami M/G-G/1. Zaprezentowane tam argumenty są oparte na fakcie, że momenty $\tau_b(i), \tau_a(i)$ są markowskie i dają wzór na P_l :

$$P_l = \frac{1}{m}(V_l(a,b) + W_l(b,a)) \tag{2}$$

gdzie

$$V_l(a,b) = \int_0^\infty P_a(X(t) = l, \tau^+(a,b) > t)dt$$

$$W_l(b,a) = \int_0^\infty P_b(X(t) = l, \tau^-(b,a) > t)dt$$

$$m = E\tau^+(a,b) + E\tau^-(b,a)$$

Tutaj $\tau^+(a,b)$ jest przedziałem czasu, w którym długość kolejki zmienia się z poziomu a do b. Innymi słowy zakładamy $X(0) = a$, mamy wtedy $\tau^+(a,b) = \tau_b(1)$. Podobnie $\tau^-(b,a)$ jest przedziałem czasu, w którym długość kolejki zmienia się z poziomu b na a, i zakładamy $X(0) = b$, wtedy $\tau^-(b,a) = \tau_a(1)$. Momenty $\tau_b(i), \tau_a(i)$ w modelu M/G-G/1/N są markowskie. W takim razie zostało nam wyznaczyć $V_l(a,b), W_l(b,a)$ oraz wartość m.

3. Obliczanie V_l

Znalezienie V_l jest łatwiejszą częścią zadania. Zauważmy, że postać tego funkcjonału zależy od procesu zachowania się długości kolejki przed osiągnięciem poziomu b. Oznacza to, że funkcjonały $V_l(a,b)$ dla systemu M/G-G/1 i M/G-G/1/N są równe. W systemie M/G-G/1 formułe dla $V_l(a,b)$ otrzymano przez użycie metody potencjałów [4]. Główną rolę w tej metodzie odgrywa następujące twierdzenie.

Twierdzenie 1. *Niech* $\{p_i\}_{i=-1}^{\infty}$ *oznacza dyskretny rozkład skoncentrowany na zbiorze* $\{-1,0,1,\ldots\}$ *taki, że* $p_{-1}>0$ *i* $\{\psi_i\}_{i=1}^{\infty}$ *dany ciąg. Głównymi rozwiązaniami tego systemu są równania*

$$\sum_{n=-1}^{k-1} p_n x_{k-n} - x_k = \psi_k \qquad k=1,2,\ldots \tag{3}$$

które mają postać

$$x_k = CR_k + \sum_{n=1}^{k} \psi_k R_{k-n} \qquad k=1,2,\ldots \tag{4}$$

gdzie C jest stałą niezależną od k oraz ciąg $\{R_i\}_{i=0}^{\infty}$ *ma rekurencyjną postać*

$$R_0 = 0 \quad R_{k+1} = \frac{1}{p_1}(\delta_{0,k} + R_k - \sum_{n=0}^{k} p_k R_{k-n}) \qquad k=0,1,\ldots \tag{5}$$

Zauważmy, że ciąg

$$\{R_i\}_{i=0}^{\infty}$$

nazywany jest potencjałem dla rozkładu $\{p_i\}_{i=-1}^{\infty}$. Używanie metody potencjałów ułatwia otrzymanie V_l:

$$V_l(a,b) = \varphi_1 \sum_{k=0}^{b-a-1} p_{k-1} R_{b-a-k} - \sum_{k=1}^{b-a-1} r_{l-b+k} R_{b-a-k} \tag{6}$$

gdzie

$$\varphi_l = \frac{\sum_{k=1}^{b-1} r_{l-b+k}(R_{b-k} - R_{b-1-k}) + \frac{I(l=0)}{\nu}}{\sum_{k=0}^{b-1} p_{k-1}(R_{b-k} - R_{b-1-k})}$$

$$r_k = I(k \geq 0) \int_0^{\infty} \frac{e^{-\nu t}(\nu t)^k}{k!}(1 - F_1(t))dt$$

$$p_k = \int_0^{\infty} \frac{e^{-\nu t}(\nu t)^{k+1}}{(k+1)!} df_1(t) \qquad k=-1,0,1,\ldots$$

i $\{R_i\}_{i=0}^{\infty}$ jest potencjałem dla rozkładu $\{p_i\}_{i=-1}^{\infty}$. Ostatecznie zauważmy, że

$$E\tau^+(a,b) = \int_0^{\infty} P_a(\tau^+(a,b) > t)dt = \sum_{l=0}^{b-1} V_l(a,b) \tag{7}$$

4. Obliczanie W_l

Oczywiście funkcjonały $W_l(b,a)$ dla systemu M/G-G/1 i M/G-G/1/N nie są równe. Dla znalezienia $W_l(b,a)$ w naszym przypadku w tej części rozważmy standardowy system kolejkowy M/G/1/K (gdzie K jest pojemnością systemu włącznie z zadaniem w systemie). Rozkład czasu przybycia dany jest przez $G(x) = 1 - e^{-\nu x}$ i rozkładem czasu obsługi $F_2(x)$ Ponieważ,

$$P_b(X(t) = l, \tau^-(b,a) > t) = P_{b-a}(X(t) = l, \tau^-(b-a,0) > t) \tag{8}$$

Możemy wyliczyć tylko $P_n(X(t) = l, \tau^-(n,0) > t)$ dla $l > 0, n > 0$ bez tracenia ogólności. Zauważmy najpierw, że $X(0) = n$ i $2 \leq n \leq K$. Używając całej formuły prawdopodobieństwa ze szczególną uwagą na pierwszy moment wyjścia (ozn. γ_1) otrzymujemy

$$P_n(X(t) = l, \tau^-(n,0) > t) = \int_0^t P_n(X(t) = l, \tau^-(n,0) > t | \gamma_1 = u) dF_2(u)$$

$$+ \int_t^\infty P_n(X(t) = l, \tau^-(n,0) > t | \gamma_1 = u) dF_2(u) \tag{9}$$

Jeśli wziąć pod uwagę dobrze znaną własność procesów Poissona oraz fakt, że moment γ_1 jest markowskim momentem

$$\int_0^t P_n(X(t) = l, \tau^-(n,0) > t | \gamma_1 = u) dF_2(u)$$

$$= \sum_{k=0}^{K-n} P_{n+k-1}(X(t-u) = l, \tau^-(n+k-1,0) > t-u) \frac{e^{-\nu u}(\nu u)^k}{k!} dF_2(u)$$

$$+ \sum_{k=K-n+1}^{\infty} \int_0^t P_{K-1}(X(t-u) = l, \tau^-(n+k-1,0) > t-u) \frac{e^{-\nu u}(\nu u)^k}{k!} dF_2(u))$$

Ponadto dla $u > t$:

$$c_{l,n}(t) =: P_n(X(t) = l, \tau^-(n,0) > t) = \begin{cases} I(l \geq n) \frac{e^{-\nu t}(\nu t)^{l-n}}{(l-n)!} & \text{dla } l < K \\ \sum_{k=K-n}^{\infty} \frac{e^{-\nu t}(\nu t)^k}{k!} & \text{dla } l = K. \end{cases} \tag{10}$$

Ustalając l i używając skróconej notacji $\Phi_n(t) = P_n(X(t) = l, \tau^-(n,0) > t)$ otrzymujemy ze wzoru (9)

$$\Phi_n(t) = \sum_{k=0}^{K-n} \int_0^t \Phi_{n+k-1}(t-u) \frac{e^{-\nu u}(\nu u)^k}{k!} dF_2(u)$$

$$+ \sum_{k=K-n+1}^{\infty} \int_0^t \Phi_{K-1}(t-u) \frac{e^{-\nu u}(\nu u)^k}{k!} dF_2(u) + c_{l,n}(t)(1-F_2(t)) \tag{11}$$

Założyliśmy, że $n \geq 2$. Łatwo zauważyć, że jeżeli dodatkowo zdefiniujemy $\Phi_0(t) = 0$, to (11) obowiązuje również dla $n = 0$. Łącząc obie strony (11) widzimy, że

$$\phi_n = \sum_{k=0}^{K-n} \phi_{n+k-1} p_{k-1} + \phi_{K-1} d_{K-n} + r_{l,n}, \tag{12}$$

gdzie

$$\phi_n = \int_0^t \Phi_n(t)dt, \quad p_k = \int_0^t \frac{e^{-\nu t}(\nu t)^{k+1}}{(k+1)!} dF_2(t), \quad k = -1, 0, 1, \ldots$$

$$r_{l,n} = \begin{cases} I(l \geq n) \int_0^t (1-F_2(t)) \frac{e^{-\nu t}(\nu t)^{l-n}}{(l-n)!} dt & \text{dla } l < K \\ m_2 - \sum_{k=0}^{K-n-1} \int_0^t (1-F_2(t)) \frac{e^{-\nu t}(\nu t)^k}{k!} dt & \text{dla } l = K \end{cases}$$

$$m_2 = \int_0^{\infty} (1-F_2(t))dt,$$

$$d_k = 1 - \sum_{i=0}^{k} p_{i-1}.$$

Przyjmując teraz $\varphi_n = \phi_{K-n}$, możemy zapisać równanie (12) jako

$$\sum_{k=-1}^{n-1} \varphi_{n-k} p_k - \varphi_n = \psi_n \tag{13}$$

z $\psi_n = -\varphi_1 d_n - r_{l,K-n}$, to pozwala nam odnosić się do twierdzenia 1. Otrzymamy

$$\varphi_n = CR_n + \sum_{k=1}^{n} R_{n-k} \psi_k \tag{14}$$

Niech $n = 1$ w (14) otrzymujemy $C = \varphi_1 / R_1$. Gdy $\psi_0 = 0$, mamy $\varphi_K = 0$, i przyjmując $n = K$ w (14) otrzymujemy

$$\varphi_1 = \varphi_1(l) = \frac{\sum_{k=1}^{K} R_{K-k} r_{l,K-k}}{R_K/R_1 - \sum_{k=1}^{K} R_{K-k} d_k}.$$

Porównując teraz wszystkie rezultaty otrzymujemy

$$\phi_n = \varphi_1(\frac{R_{K-n}}{R_1} - \sum_{k=1}^{K-n} R_{K-n-k} d_k) - \sum_{k=1}^{K-n} R_{K-n-k} r_{l,K-k}$$

Ostatecznie możemy te wyniki przystosować do oscylujących systemów. W tym celu bierzemy $n = b-a$, $K = N-a$ i $l = l'-a$. To dla systemu M/G-G/1/N otrzymujemy

$$W_l(b,a) = \varphi_1(l-a)(\frac{R_{N-b}}{R_1} - \sum_{k=1}^{N-b} R_{N-b-k} d_k) - \sum_{k=1}^{N-b} R_{N-b-k} r_{l-a,N-a-k} \tag{15}$$

Dla każdego $l = a+1,\ldots,N$. Ponadto (patrz (7) do porównania):

$$E\tau^-(b,a) = \sum_{i=a+1}^{N} W_l(b,a) \tag{16}$$

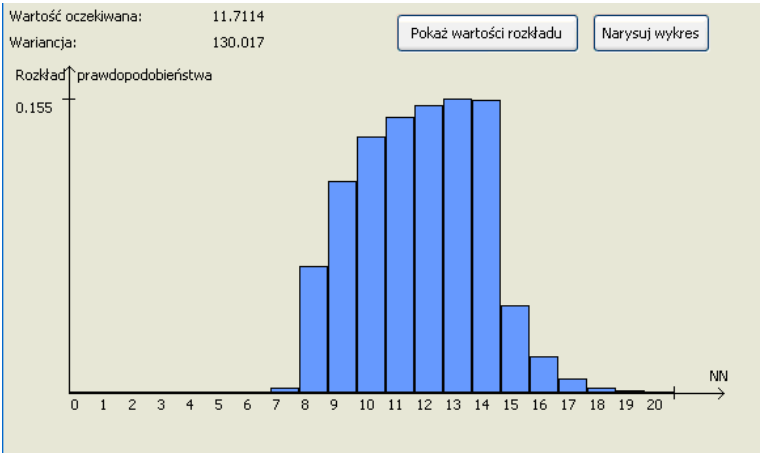
5. Program do modelowania kolejek z oscylacjami

Załóżmy, że mamy do dyspozycji dwa serwery (obsługują one zgłoszenia z różnymi rozkładami: pierwszy serwer z wykładniczym natomiast drugi serwer z rozkładem jednostajnym). Jeden z nich ma niskš wydajność, ale jest tani w eksploatacji. Drugi natomiast ma dużą wydajność, ale jest drogi w eksploatacji. Na początku strumień wejściowy jest obsługiwany przez pierwszy mniej efektywny serwer (przez co kolejka rośnie). Gdy długość kolejki osiągnie krytyczny poziom b, obsługę przejmuje drugi bardziej efektywny serwer (przez co kolejka szybko maleje). Drugi serwer pracuje do momentu, gdy długość kolejki osiągnie mniejszy dostateczny poziom a. Wtedy obsługę przejmuje mniej efektywny serwer, itd. Stąd też powstał pomysł stworzenia programu, który ułatwi projektowanie i testowanie systemu. Dzięki temu będzie można w szybki sposób przetestować wiele przykładów, które ułatwią dobór optymalnych parametrów.

Przykład 1

Dane:
Progi $a = 8$ oraz $b = 15$
Rozmiar systemu $NN = 20$
Intensywność wejściowego strumienia Poissona $\nu = 1$
Parametr obsługi pierwszego serwera $L_1 = 0.1$, gdzie L_1 oznacza parametr λ_1, który opisuje rozkład czasu obsługi pierwszego serwera ($\lambda_1 > 0$).

Nośnik f1(t) w postaci przedziału $[u_1; v_1] = [1; \infty]$, $[u_1; v_1]$ przedział czasowy, w którym pracuje urządzenie.
Nośnik f2(t) w postaci przedziału $[u_2; v_2] = [0; 1]$



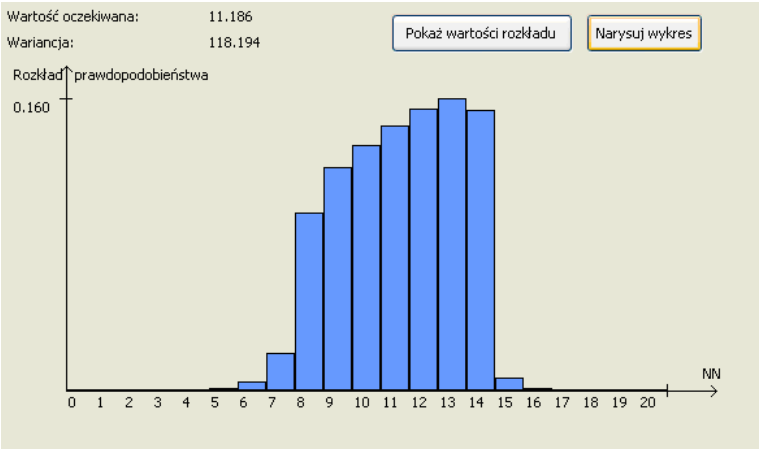
Rys. 1. Wykres rozkładu prawdopodobienstwa - Przykład 1

Przykład 2

Dane:
Progi $a = 8$ oraz $b = 15$
Rozmiar systemu $NN = 20$
Intensywność wejściowego strumienia Poissona $\nu = 0.3$
Parametr obsługi pierwszego serwera $L_1 = 0.1$
Nośnik f1(t) w postaci przedziału $[u_1; v_1] = [1; \infty]$
Nośnik f2(t) w postaci przedziału $[u_2; v_2] = [0; 1]$

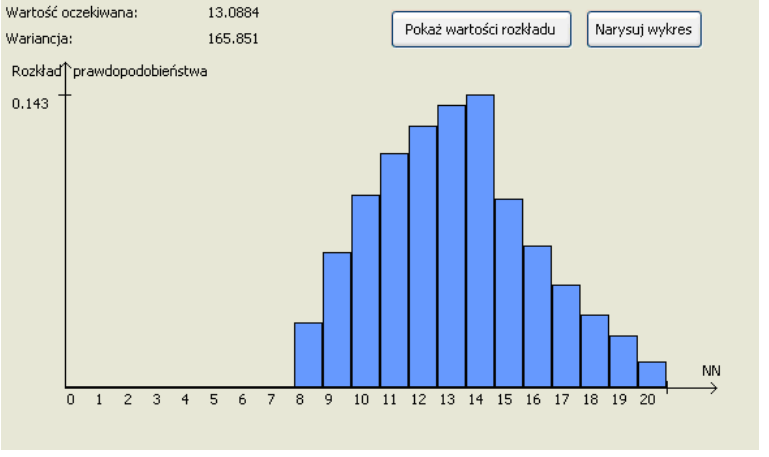
Przykład 3

Dane:
Progi $a = 8$ oraz $b = 15$
Rozmiar systemu $NN = 20$
Intensywność wejściowego strumienia Poissona $\nu = 1.6$
Parametr obsługi pierwszego serwera $L_1 = 0.1$
Nośnik f1(t) w postaci przedziału $[u_1; v_1] = [1; \infty]$



Rys. 2. Wykres rozkładu prawdopodobieństwa - Przykład 2

Nośnik f2(t) w postaci przedziału $[u_2; v_2] = [0; 1]$



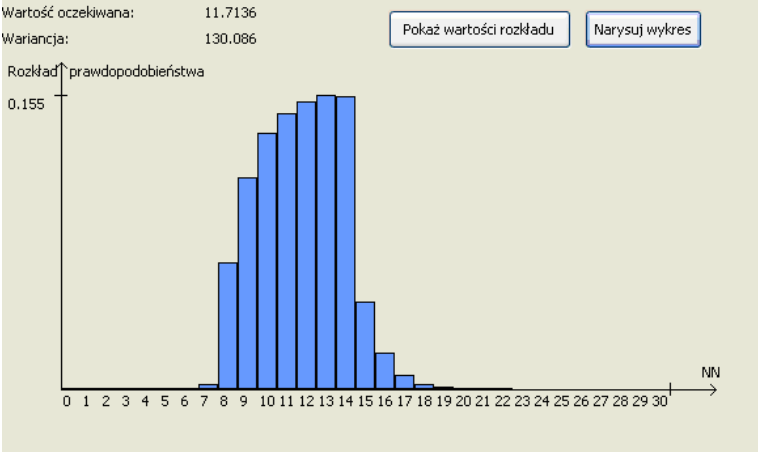
Rys. 3. Wykres rozkładu prawdopodobieństwa - Przykład 3

Przykład 4

Dane:

Progi $a = 8$ oraz $b = 15$

Rozmiar systemu $NN = 30$
Intensywność wejściowego strumienia Poissona $\nu = 1$
Parametr obsługi pierwszego serwera $L_1 = 0.1$
Nośnik f1(t) w postaci przedziału $[u_1; v_1] = [1; \infty]$
Nośnik f2(t) w postaci przedziału $[u_2; v_2] = [0; 1]$



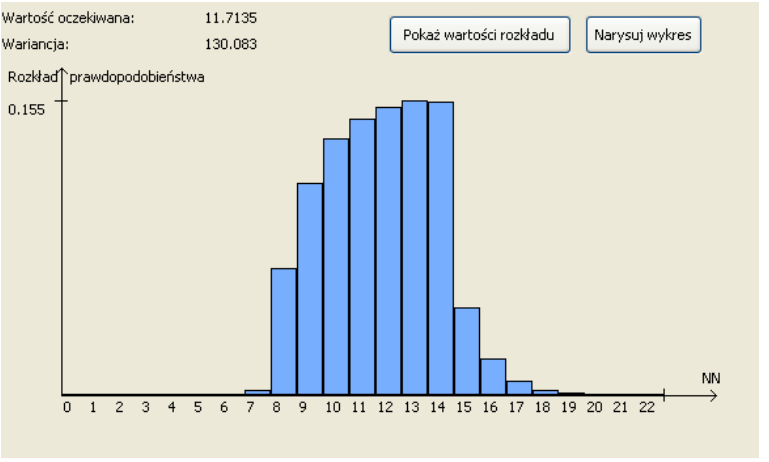
Rys. 4. Wykres rozkładu prawdopodobieństwa - Przykład 4

Przykład 5

Dane:
Progi $a = 8$ oraz $b = 15$
Rozmiar systemu $NN = 22$
Intensywność wejściowego strumienia Poissona $\nu = 1$
Parametr obsługi pierwszego serwera $L_1 = 0.1$
Nośnik f1(t) w postaci przedziału $[u_1; v_1] = [1; \infty]$
Nośnik f2(t) w postaci przedziału $[u_2; v_2] = [0; 1]$

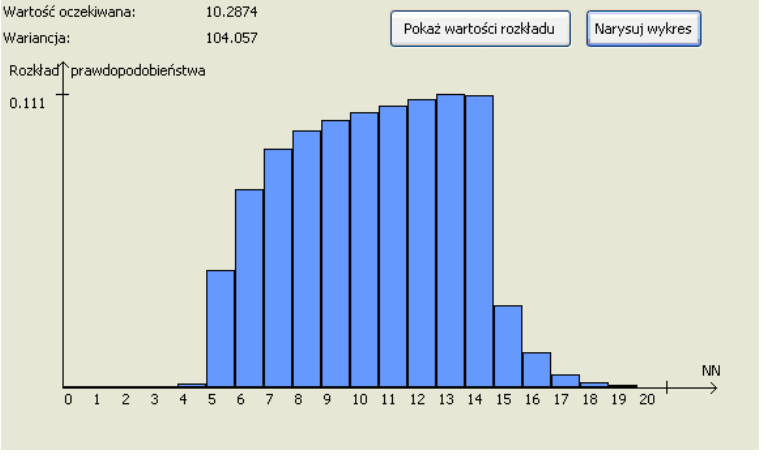
Przykład 6

Dane:
Progi $a = 5$ oraz $b = 15$
Rozmiar systemu $NN = 22$
Intensywność wejściowego strumienia Poissona $\nu = 1$



Rys. 5. Wykres rozkładu prawdopodobieństwa - Przykład 5

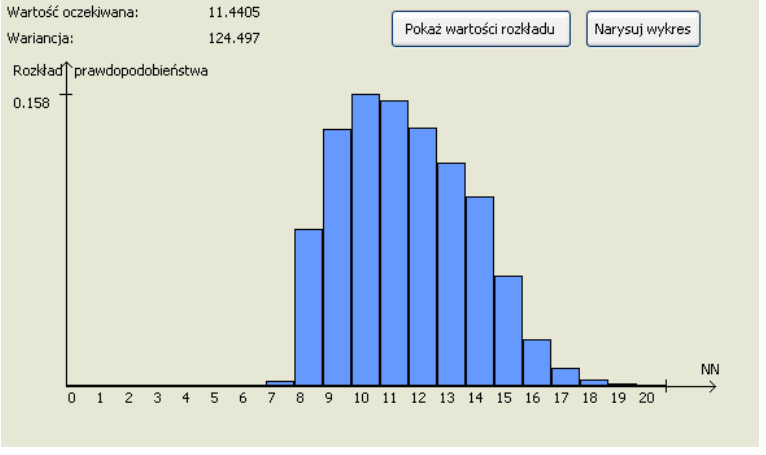
Parametr obsługi pierwszego serwera $L_1 = 1$
Nośnik f1(t) w postaci przedziału $[u_1; v_1] = [1; \infty]$
Nośnik f2(t) w postaci przedziału $[u_2; v_2] = [0; 1]$



Rys. 6. Wykres rozkładu prawdopodobieństwa - Przykład 6

Przykład 7

Dane:
Progi $a = 5$ oraz $b = 15$
Rozmiar systemu $NN = 22$
Intensywność wejściowego strumienia Poissona $\nu = 1$
Parametr obsługi pierwszego serwera $L_1 = 1$
Nośnik f1(t) w postaci przedziału $[u_1; v_1] = [1; 2]$
Nośnik f2(t) w postaci przedziału $[u_2; v_2] = [0; 1]$



Rys. 7. Wykres rozkładu prawdopodobieństwa - Przykład 7

6. Podsumowanie

Wyniki otrzymane w czasie eksperymentów przeprowadzonych na pakiecie programów zobrazowały pewne problemy i zależności związane z tym modelem. W eksperymentach zmieniano np. wartości progowe, aby sprawdzić czy czynnik ten ma wpływ na obsługę zgłoszeń. Jak zauważono, przeczucia były słuszne. Kiedy zwiększano czas obsługi na stanowisku, wartości czasu oczekiwania w kolejce, czas oczekiwania w systemie oraz długość kolejki, obciążenia stanowiska rosły. Na zachowywanie się przedstawionych systemów majś wpływ różne czynniki np. intensywność strumienia wejściowego, ustalenie progów i wiele innych. Trzeba dużo i długo manipulować danymi wejściowymi, aby otrzymać zamierzony efekt.

Literatura

[1] Chydziński, A: The M/G-G/1 oscillating queueing system, Queueing Systems 42 (3), 2001.

[2] Chydziński, A: The oscillating queue with finite buffer, Performance Evaluation 57, 2004.

[3] Oniszczuk, W.: Modele, algorytmy kolejkowe i strategie obsługi w sieciach komputerowych, Wydawnictwa Politechniki Białostockiej, Rozprawy Naukowe Nr 128,2005.

[4] Korolyuk, W: Boundary Problems for Compound Poisson Processes, Naukowa Dumka, Kiev, 1975.

[5] Jakubowski, J. & Sztencel, R.: Wstęp do teorii prawdopodobieństwa, SCRIPT, 2004.

[6] Grzech, A.: Sterowanie ruchem w sieciech teleinformatycznych, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 2002.

[7] Choi, D. & Knessl, C. & Tier, C.: A queueing system with queue length dependent service time, with applications to cell discarding in ATM networks, J. Appl. Math. Stochast. Anal. 12 (1), 1999.

[8] Sriram, K. & McKinney, R.S. & Sherif, M.H.: Voice packetization and compression in broaddand ATM networks, IEEE J.Select. Areas Commun. 9 (3), 1991.

SERVERS WITH OSCILLATING: SERVICE PATERNS MODELLING

Abstract: In this paper a finite buffer version of the oscillating queuing system is studied. The idea of the lately introduced oscillating queuing system is based on two threshold values. The service process in this system is planned in such a way that the queue length is kept between these values. The oscillating queuing system has the advantage of making improved use of the available resources and is applicable in many devices which use a single server queuing scheme. It is also a simplification of some cell disposal procedures projected for ATM networks.

Keywords: oscillating queue, potential method, ATM networks

Artykuł zrealizowano w ramach pracy badawczej S/WI/5/03

Aleksander Ostanin[1], Jerzy Wasiluk[1]

WŁAŚCIWOŚCI PROGRAMOWEJ REALIZACJI ZADANIA PROGRAMOWANIA CAŁKOWITOLICZBOWEGO

Streszczenie: Praca poświęcona jest problemom realizacji nowej optymalizacyjnej funkcji **bintprog** jako nieodłącznej części *Optimization Toolbox 3.0* programowego pakietu *MATLAB* 7. Wprowadzenie nowej funkcji istotnie poszerza skalę rozwiązywanych optymalizacyjnych zadań, ponieważ potwierdza fakt realizowania metody gałęzi i granic. Badanie tej metody wchodzi w skład programu wielu uczelnianych kursów i daje podstawę do rozwiązywania większości zadań programowania całkowitoliczbowego. Na zakończenie podano przykłady niektórych reprezentatywnych wyników uzyskanych w badaniach.

Słowa kluczowe: programowanie całkowitoliczbowe, metoda gałęzi i granic, decyzja o rozgałęzieniu

1. Wprowadzenie

Programowanie całkowitoliczbowe liniowe jest szczególnym przypadkiem zadania programowania liniowego. Oprócz założeń dotyczących liniowej postaci funkcji celu i warunków ograniczających zakładamy, że wszystkie (lub niektóre) zmienne powinny przyjmować jedynie wartości całkowite. Jeżeli te dodatkowe warunki, zwane warunkami całkowitoliczbowości nałożone są na wszystkie występujące w zadaniu zmienne, rozpatrywane zadanie jest *czystym* zadaniem programowania całkowitoliczbowego, liniowego (PCL). Jeżeli warunki te nałożono tylko na niektóre zmienne, mamy do czynienia z zadaniem *mieszanym*, liniowym (PML). Bardzo istotne praktyczne znaczenie mają również zadania programowania *binarnego*, liniowego (PBL), w którym zmienne mogą przyjmować wartości 0 lub 1.

Wśród wielu zastosowań programowania całkowitoliczbowego liniowego można wymienić zadania planowania różnego typu, np. ustalania planu wydawniczego, lokalizacji punktów usługowych, harmonogramu zatrudniania pracowników, minimalizacji odpadów itp. *Optimization Toolbox 3.0* w nowej wersji programowego

[1] Wydział Informatyki, Politechnika Białostocka, Białystok

pakietu *MATLAB 7* uzupełniono o nowy program, istotnie poszerzający jego możliwości. Wprowadzono mianowicie nową funkcję **bintprog**, przeznaczoną do rozwiązywania zadań programowania całkowitoliczbowego binarnego.

Większość uwag z poprzednich wersji pakietu *MATLAB 6* oraz starszych wersji dotyczyła programowania całkowitoliczbowego. Należy zaznaczyć, że w poprzednich wersjach istniała możliwość rozwiązywania podobnych zadań, częściowo pośrednim sposobem rozwiązania równoważnego ciągłego zagadnienia pierwotnego. W przypadku inscenizacji zadania całkowitoliczbowego zastosowano następujące podejście. Konsekwentnie z liczby niezależnych zmiennych wydzielono te wielkości, które dopuszczają możliwość korekcji. Formowanie dyskretnych wartości wykonuje się drogą zaokrągleń do najbliższego całkowitego znaczenia. Po wyodrębnieniu dyskretnych wartości dokonuje się rozwiązania przytoczonego zadania dla pozostałych wolnych zmiennych. Po znalezieniu rozwiązania zadania wyodrębnia się inne zmienne dyskretne, a obliczeniowy cykl powtarza się tak długo aż będą wybrane wszystkie. Taka inscenizacja zadań programowania całkowitoliczbowego istotnie obniżała funkcjonalne możliwości programowego pakietu *MATLAB 6*.

Wprowadzenie nowej funkcji **bintprog** istotnie poszerza skalę rozwiązywanych zadań optymalizacyjnych. Zainteresowanie funkcją **bintprog** potwierdza fakt, że realizowana jest w niej tak zwana *metoda gałęzi i granic*. Właśnie ta metoda dosyć szczegółowo jest opisana w literaturze i stanowi podstawę rozwiązania większości zadań programowania całkowitoliczbowego. Po raz pierwszy metodę gałęzi i granic przedstawili Land i Dojg [1] w 1960 roku do rozwiązania ogólnego zadania programowania liniowego całkowitoliczbowego. Zainteresowanie tą metodą i faktycznie jej "drugie narodzenie"wiąże się z pracą Little'a *(algorytm Little 'a)*, która poświęcona była rozwiązaniu zadania komiwojażera [2]. Podstawę metody zestawia pewne wielopoziomowe drzewo, na dolnym poziomie którego znajdują się elementy mnóstwa, gałęzi które prowadzą do określonego optimum.

Autorzy niniejszej pracy postawili sobie za cel opisanie sposobów realizacji nowej optymalizacyjnej funkcji bintprog jako nieodłącznej części *Optimization Toolbox 3* programowego pakietu *MATLAB 7*. Przytacza się przykłady wykorzystania funkcji **bintprog**.

2. Algorytm metody gałęzi i granic

W funkcji **bintprog** do rozwiązania zadania programowania całkowitoliczbowego wykorzystuje się algorytm programowania liniowego na podstawie metody gałęzi i granic. Podstawę metody zestawia pewne wielopoziomowe drzewo, którego stwgałęzie arzają osnowę algorytmu wyszukiwania określonego optimum. W każdym kroku

metody elementy rozgałęzienia podlegają sprawdzeniu czy zawarty podzbiór daje optymalne rozwiązanie czy też nie. Sprawdzenie dochodzi do skutku za pośrednictwem obliczenia oceny z dołu dla funkcji celu zawartego podzbioru. Jeżeli ocena z dołu nie jest mniejsza od *optimum* - najlepszego ze znalezionych rozwiązań, to podzbiór może być odrzucony. Kontrolowany podzbiór może być odrzucony jeszcze i w tym wypadku, gdy udaje się znaleźć najlepsze rozwiązanie. Jeżeli znaczenie funkcji celu w znalezionym rozwiązaniu jest mniejsze od *optimum*, to następuje jego zmiana.

W algorytmie metody gałęzi i granic realizowanej za pomocą funkcji **bintprog** przeprowadza się przegląd optymalnych rozwiązań zadania programowania całkowitoliczbowego drogą rozwiązania pewnego zbioru zadań LP - relaksacji, w których żądanie całkowitoliczbowości zmiennych zastępuje się słabszymi ograniczeniami $0 \leq x \leq 1$. Rozpatrywany algorytm zawiera:

- Przegląd całkowitoliczbowych dopuszczalnych rozwiązań.
- Korektę najlepszego całkowitoliczbowego dopuszczalnego punktu.
- Sprawdzenie niemożliwości osiągnięcia lepszych całkowitoliczbowych wyników drogą rozwiązania rzędu zadań programowania liniowego.

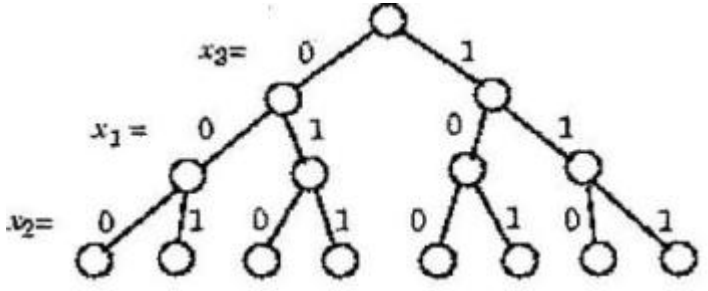
2.1 Rozgałęzienie

Zaproponowany algorytm tworzy drzewo wyszukiwania drogą wielokrotnego dodawania ograniczeń na podane zadanie, tj. rozgałęzienia. Na etapie rozgałęzienia w algorytmie będzie przeprowadzać się wybór pewnej zmiennej x, której bieżąca wartość nie jest całkowita i dalej nakładają się ograniczenia xj = 0 dla formowania jednej gałęzi i ograniczenie xj = 1 dla formowania innej gałęzi. Cały ten proces może być przedstawiony w postaci dwójkowego drzewa, którego węzły przedstawiają dodatkowo nakładane ograniczenia. Na rys. 1 przedstawiono ilustrację skończonego binarnego drzewa dla zadania z trzema zmiennymi x1, x2 i x3. Należy zaznaczyć, że w ogólnym wypadku rząd zmiennych przy obniżeniu poziomów drzewa może nie odpowiadać zwyczajnemu porządkowi indeksów zmiennych.

2.2 Decyzja o rozgałęzieniu

Na każdym węźle w algorytmie będzie wykonywane rozwiązanie zadania LP - relaksacji z uwzględnieniem ograniczeń dla danego węzła i w zależności od otrzymanego rezultatu przyjmuje się decyzje o konieczności lub rozgałęzieniu lub przejściu do innego węzła. Są trzy możliwości:

- Jeżeli zadanie LP - relaksacji dla danego bieżącego punktu jest niedopuszczalne lub jej optymalne rozwiązanie znaczenie większe niż rozwiązanie w najlepszym



Rys. 1. Binarne drzewo dla zadania z trzema zmiennymi x_1, x_2, i x_3

punkcie całych wartości, to algorytm usuwa ten punkt z drzewa, po czym nie będzie już innych przeglądów w gałęziach poniżej podanego węzła. Dalej algorytm przechodzi do analizy nowego węzła zgodnie z metodą zadaną przez opcję *NodeSearchStrategy*.

- Jeżeli algorytm określa nowy dopuszczalny całkowitoliczbowy punkt z mniejszym znaczeniem funkcji celu, zamiast wcześniej znalezionego, to koryguje on najlepszy docelowy punkt i przechodzi do analizy następującego węzła.
- Jeżeli zadanie LP- relaksacji jest optymalne, lecz nie całkowitoliczbowe, to znaczenie optymalnej funkcji celu zadania LP - relaksacji jest mniejsze niż w najlepszym punkcie, a algorytm przechodzi na nową gałąź zgodnie z metodą zadaną przez opcję *BranchStrategy*.

2.3 Granice

Rozwiązanie zadania LP - relaksacji określa dolną granicę dla zadania binarnego programowania całkowitoliczbowego. Jeżeli rozwiązanie zadania LP - relaksacji już jest binarnym całym wektorem, to takie rozwiązanie określa górną granicę zadania programowania całkowitoliczbowego binarnego.

W ciągu przeglądania węzłów w drzewie poszukiwania algorytm będą przeprowadzać korekty dolnej i górnej granicy funkcji celu z obliczeniem danych otrzymanych na etapie granic. Granica dla funkcji celu służy w charakterze progu do odcinania nadmiernych niepotrzebnych gałęzi.

2.4 Ograniczenia w algorytmie

Algorytm funkcji **bintprog** potencjalnie może przejrzeć wszystkie 2^n binarnych całych wektorów, gdzie n jest liczbą zmiennych. Ponieważ dla realizacji pełnego algo-

rytmu może być potrzebne wydłużenie czasu, należy więc nałożyć pewne ograniczenie na procedury przeglądania za pomocą następujących opcji:

- *MaxNodes* - Maksymalna liczba węzłów w algorytmie przeglądania.
- *MaxRLPIter* - Maksymalna liczba iteracji LP dla pewnego węzła przy rozwiązaniu zadania LP-relaksacji.
- *MaxTime* - Maksymalna ilość czasu w sekundach dla realizacji zadanej funkcji.

3. Opis programowej funkcji bintprog

Funkcja **bintprog** przeznaczona jest do rozwiązywania zadań programowania całkowitoliczbowego binarnego postaci

$$min_x f^T x$$

przy obecności ograniczeń

$$\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}, \mathbf{A_{eq}} \cdot \mathbf{x} = \mathbf{b_{eq}},$$

gdzie **f, b** i $\mathbf{b_{eq}}$ są wektorami, **A** i $\mathbf{A_{eq}}$ - macierzą, a **x** jest całkowitoliczbowym binarnym wektorem rozwiązania, to znaczy że jego komponenty powinny przyjmować wartości 0 lub 1.

3.1 Sposoby wywołania funkcji bintprog są następujące:

$\mathbf{x} = \mathbf{bintprog}(\mathbf{f})$ - rozwiązuje zadanie programowania całkowitoliczbowego $min_x f^T x$

$\mathbf{x} = \mathbf{bintprog}(\mathbf{f}, \mathbf{A}, \mathbf{b})$ - rozwiązuje zadanie programowania całkowitoliczbowego z warunkiem liniowej nierówności $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$.

$\mathbf{x} = \mathbf{bintprog}(\mathbf{f}, \mathbf{A}, \mathbf{b}, \mathbf{A_{eq}}, \mathbf{b_{eq}})$ - rozwiązuje poprzednie zadanie przy dodatkowych warunkach równościowych $\mathbf{A_{eq}} \cdot \mathbf{x} = \mathbf{b_{eq}}$.

$\mathbf{x} = \mathbf{bintprog}(\mathbf{f}, \mathbf{A}, \mathbf{b}, \mathbf{A_{eq}}, \mathbf{b_{eq}}, \mathbf{x0})$ - ustanawia początkowy punkt wyszukiwania **x0.**
Jeżeli punkt **x0** znajduje się w niedopuszczalnym zakresie, to polecenie **bintprog** przyjmuje dowolny punkt początkowy.

$\mathbf{x} = \mathbf{bintprog}(\mathbf{f}, \mathbf{A}, \mathbf{b}, \mathbf{A_{eq}}, \mathbf{b_{eq}}, \mathbf{x0}, \mathbf{options})$ - podczas optymalizacji wykorzystuje przyjmowaną domyślnie opcję ze struktury **options**, którą można wyznaczyć za pomocą funkcji **optimset.**

$[\mathbf{x}, \mathbf{fval}] = \mathbf{bintprog}(...)$- zwraca **fval** jako wartość docelowej funkcji w punkcie **x.**

$[\mathbf{x}, \mathbf{fval}, \mathbf{exitflag}] = \mathbf{bintprog}(...)$ - zwraca parametr **exitflag** z opisem warunków wyjściowych funkcji **bintprog.**

$[\mathbf{x}, \mathbf{fval}, \mathbf{exitflag}, \mathbf{output}] = \mathbf{bintprog}(...)$ - zwraca strukturę **output**, która zawiera informację o wynikach optymalizacji.

3.2 Wejściowe argumenty funkcji bintprog:

Wejściowe argumenty komendy **bintprog** włączają do siebie następujące zmienne:

f	Wektor ze współczynnikami liniowej funkcji celu
A	Macierz ze współczynnikami liniowych ograniczeń nierównościowych typu $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$
b	Wektor prawej części liniowych ograniczeń nierównościowych
$\mathbf{A_{eq}}$	Macierz ze współczynnikami liniowych ograniczeń równościowych typu $\mathbf{A_{eq}} \cdot \mathbf{x} = \mathbf{b_{eq}}$
$\mathbf{b_{eq}}$	Wektor prawej części liniowych ograniczeń typu równościowych
x0	Początkowy punkt w algorytmie wyszukiwania
options	Struktura opcji w algorytmie wyszukiwania

3.3 Wyjściowe dane funkcji bintprog:

Wyjściowe parametry **exitflag** i **output** posiadają szereg następujących właściwości: (patrz tabela na rysunku 2)

3.4 Struktura opcji funkcji bintprog

Sterowanie funkcjami optymalizacyjnymi (wprowadzenie lub zmiana wartości danych pola podanej struktury) odbywa się za pomocą opcji ustawianych funkcją **optimset**. Przewidywane wykorzystanie następujących parametrów opcji funkcji **bintprog**:

BranchStrategy	Typ algorytmu, wykorzystywanego przy wyborze zmiennej rozgałęzienia wprowadzanej w drzewie wyszukiwania: *'mininfeas'* - wybór zmiennej z minimalną całkowitoliczbową nieosiągalnością, tzn. wybiera się zmienne z wartościami bliskimi 0 lub 1, lecz nie równymi 0 lub 1. *'maxinfeas'* - wybór zmiennej z maksymalną całkowitoliczbową nieosiągalnością, tzn. wybiera się zmienne z wartościami bliskimi 0,5 (przyjmuje się domyślnie)
Diagnostics	Odwzorowanie diagnostycznej informacji o funkcji, podlegającej minimalizacji bądź obliczeniu
Display	Poziom odwzorowania wyników: bez wyświetlania (*'off'*), wyświetlanie wyników po każdej iteracji (*'iter'*), wyświetlanie tylko ostatecznego rozwiązania (*'final'*- wartość domyślna)
DispNodeInterval	Odstęp odwzorowań węzłów
MaxIter	Maksymalna liczba dopuszczalnych iteracji
MaxNodes	Maksymalna liczba rozwiązań (lub przeglądanych węzłów)
MaxRLPIter	Maksymalna liczba iteracji dla dowolnego węzła przy rozwiązaniu zadania relaksacji programowania liniowego (LP-relaksacji)
MaxTime	Maksymalna ilość czasu w sekundach dla realizacji zadanej funkcji
NodeSearchStrategy	Strategia algorytmu, wykorzystanego dla doboru następnego węzła przy przeglądaniu drzewa poszukiwania: *'df'* - strategia pierwszego poszukiwania w zależności od poziomu gałęzi; *'bn'* - najlepsza strategia przeglądania węzłów, przy której odbiera się węzeł z najmniejszej granicznej wartością funkcji celu
TolCon	Ostateczne dopuszczalne odchylenie przy naruszeniu przydzielonych ograniczeń
TolFun	Ostateczne dopuszczalne odchylenie dla wartości funkcji celu
TolXInteger	Dopuszczalne odchylenie dla wartości zmiennych
TolRLPFun	Ostateczne dopuszczalne odchylenie dla wartości funkcji celu zadania relaksacji programowania liniowego (LP - relaksacji)

exitflag	Pewna liczba całkowita, identyfikująca przyczynę zatrzymania algorytmu. Kolejno przytacza się wykaz wartości i odpowiednich przyczyn wstrzymania algorytmu	
	1	Funkcja zbiega się do dowolnego rozwiązania **x**
	0	Liczba iteracji przekroczyła znaczenie *options.MaxIter*
	-2	Zadanie nie ma rozwiązania
	-4	Liczba przeglądanych węzłów przewyższa znaczenie *options. MaxNodes*
	-5	Czas przeglądania przekracza znaczenie *options.MaxTime*
	-6	Liczba iteracji dla pewnego węzła przy rozwiązaniu zadania relaksacji programowania liniowego (LP – relaksacji) przekroczyła znaczenie *options.MaxRLP*
output	Struktura zawierająca podstawowe informacje o procesie optymalizacyjnym	
	iterations	Liczba wykonanych iteracji
	nodes	Liczba przeglądanych węzłów
	time	Przekroczenie czasu pracy algorytmu
	algorithm	Wykorzystany algorytm
	message	Przyczyna zatrzymania algorytmu

Rys. 2. Właściwości wyjściowych parametrów **exitflag** i **output**

4. Realizacja zadania programowania całkowitoliczbowego

Przykład 1.

Należy minimalizować funkcję

$$f(x) = -9x_1 - 5x_2 - 6x_3 - 4x_4$$

przy obecności ograniczeń

$$\begin{bmatrix} 6 & 3 & 5 & 2 \\ 0 & 0 & 1 & 1 \\ -1 & 0 & 1 & 1 \\ 0 & -1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \leq \begin{bmatrix} 9 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

gdzie x_1, x_2, x_3 i x_4 są liczbami binarnymi. Wykonamy następujące polecenie:

$$f = [-9; -5; -6; -4;]$$

$$A = [6 \quad 3 \quad 5 \quad 2; 0 \quad 0 \quad 1 \quad 1; -1 \quad 0 \quad 1 \quad 0; 0 \quad -1 \quad 0 \quad 1;]$$

```
b = [9; 1; 0; 0;]

[x; fval; exitflag; output;] = bintprog(f,A,b);
```

Po realizacji wskazanych komend otrzymamy następujący wynik:

Optimization terminated successfully.

$$X = \begin{matrix} 1 \\ 1 \\ 0 \\ 0 \end{matrix}$$

$$\texttt{fval} = -14$$

$$\texttt{exitflag} = 1$$

```
output =

     iterations: 12
          nodes: 5
           time: 0.1402
      algorithm: 'LP-base branch-and-bound'
  branchStrategy: 'maximum integer infeasibility'
nodeSrchStrategy: 'best node search'
        message: 'Optimization terminated.'
```

Przykład 2.
Rozważmy następujące zadanie PBL:

$$4x_1 + 3x_2 \longrightarrow max,$$

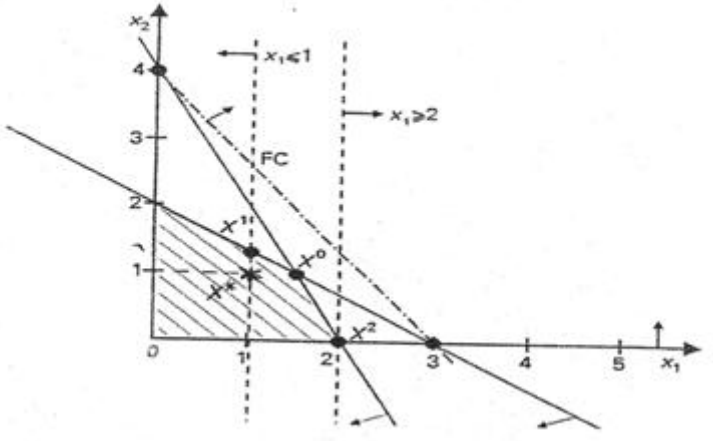
$$2x_1 + 3x_2 \leq 6,$$

$$x_1 + 2x_2 \leq 8,$$

$x_1, x_2 \geq 0,$

gdzie x_1, x_2- całkowite.

Jest to zadanie programowania całkowitoliczbowego, liniowe (PCL) z dwoma zmiennymi, więc można je rozwiązać bezpośrednio metodą graficzną. Pokażemy jednak jak uzyskać rozwiązanie metodą gałęzi i granic. Metodą graficzną będziemy rozwiązywać tylko pomocnicze ZPL. Zastosujmy najpierw metodę graficzną w odniesieniu do wyjściowego ZPL (1). Zbiór rozwiązań tego zadania to obszar zakreskowany na rys. 2. Sam podział zbioru D rozwiązań dopuszczalnych PCL przedstawiono w postaci drzewa na rys. 3.



Rys. 3. Zbiór rozwiązań ZPL

Z dowolnym l-tym węzłem (wierzchołkiem) drzewa są związane: zbiór D_l, ZPL^l, jego rozwiązanie optymalne x^l oraz wartość funkcji ograniczającej w dla górnej (dolnej) granicy. Przygotujmy dane w pliku **cl_ 1:**

```
% Plik cl_ 1
% Dane do ZPL

f = [4; 3];

A = [2   3; 4   2];

b = [6; 8];
```

po wywołaniu którego zwrócimy się do funkcji *linprog:*

```
→ [x; fval; exitflag; output;] = linprog(-f,A,b)
```

Optimization terminated.

$$X = \begin{matrix} 1.5000 \\ 1.0000 \end{matrix}$$

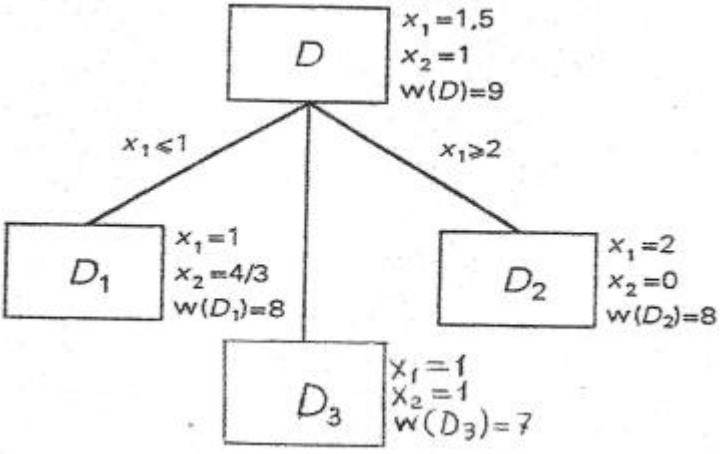
$$\texttt{fval} = \begin{matrix} -9.0000 \end{matrix}$$

$$\texttt{exitflag} = \begin{matrix} 1 \end{matrix}$$

```
output =

 iterations: 4
  algorithm: 'large-scale: interior point'
cgiterations: 0
    message: 'Optimization terminated.'
```

Rozwiązaniem optymalnym ZPL jest punkt X^0 =(1.5; l) z wartością funkcji celu $C(X^0) = 9$. Stąd wartość funkcji ograniczającej w dla górnej granicy $w(D) = 9$. Ponieważ rozwiązanie X^0 nie spełnia warunku całkowitoliczbowości, więc dzielimy zbiór



Rys. 4. Drzewo podziału dla zadania z dwoma zmiennymi x_1 i x_2

D na dwa podzbioryD_1 i D_2; $D_1 = x \mid x \in D \wedge X_1 \leq l, D_2 = x \mid x \in D \wedge X_1 \geq 2$, z którymi są związane dwa pomocnicze zadania ZPL^1 i ZPL^2. Podstawą rozgałęzienia jest zmienna x_1. Rozwiązaniem optymalnym ZPL^1 jest punkt $X^1 = (l; 4/3)$ z wartością funkcji celu $C(X^1) = 8$ i wartością funkcji ograniczającej *w(D1)* = 8. Dla zadania ZPL^2 rozwiązaniem optymalnym jest punkt $X^2 = (2;0)$, dla którego wartość funkcji celu $C(X^2) = 8$ i wartość funkcji ograniczającej $w(D_2) = 8$. Następnie wybieramy zbiór perspektywiczny, tzn. zbiór o maksymalnej (minimalnej) wartości funkcji ograniczającej oraz dokonujemy jego podziału, o ile jest to możliwe. Jeżeli jednak zbiorów o maksymalnej wartości funkcji ograniczającej jest więcej, podobnie jak w naszym przypadku, to wybieramy ten, który jest zamknięty (zbiór D_1 jest zamknięty, jeżeli: a) rozwiązanie optymalne ZPL^1 jest całkowitoliczbowe, wtedy $w(D_1) = \mathbf{cx^l}$, gdzie **c** - wektor wag funkcji celu; b) zbiór rozwiązań dopuszczalnych ZPL^1 jest pusty, wtedy przyjmujemy, że $w(D_1) = \infty$). Będzie to podzbiór D_2. Dlatego rozwiązanie X^2 jest także rozwiązaniem całego zadania PCL.

Dla zadania na minimum wyznaczamy zbiory nie o maksymalnej wartości funkcji ograniczającej, lecz o minimalnej jej wartości. Gdy zadanie PCL będzie miało więcej niż dwie zmienne, to rozwiązując poszczególne pomocnicze ZPL należy stosować metodę *SIMPLEKS*. Wywołanie pliku **cl_ 1** i zwrócenie się do funkcji *bintprog*; pozwala otrzymać nie tylko wektor rozwiązania zadania PBL, ale i minimalną wartość funkcji celu:

```
→» [x; fval; exitflag; output;] = bintprog(-f,A,b)
```

Optimization terminated.

$$X = \begin{matrix} 1 \\ 1 \end{matrix}$$

$$\texttt{fval} = \begin{matrix} -7 \end{matrix}$$

$$\texttt{exitflag} = \begin{matrix} 1 \end{matrix}$$

```
output =

     iterations: 2
          nodes: 1
           time: 0.0100
      algorithm: 'LP-base branch-and-bound'
  branchStrategy:'maximum integer infeasibility'
nodeSrchStrategy:'best node search'
        message: 'Optimization terminated.'
```

Dla zadania programowania binarnego, liniowego, rozwiązaniem optymalnym jest punkt $X^* = (1;1)$ z wartością funkcji celu $C(X^*) = 7$ i wartością funkcji ograniczającej $w(D_3) = 7$.

5. Wnioski

W pracy przedstawiono opis i sposób wykorzystania w *Optimization Toolbox 3.0* pakietu *MATLAB* 7 nowej funkcji **bintprog** istotnie poszerzającej skalę rozwiązywanych optymalizacyjnych zadań. Zastosowanie funkcji **bintprog** potwierdza fakt realizowania metody gałęzi i granic do rozwiązywania większości trudnych zadań programowania całkowitoliczbowego. Przytacza się przykłady zastosowania funkcji **bintprog**.

Przedstawiony materiał może być wykorzystany do praktycznego zastosowania przez specjalistów, którzy w swojej działalności wykorzystują pakiet *MATLAB* i rozwiązują zadania programowania liniowego, całkowitoliczbowego oraz binarnego.

Literatura

[1] Land A.H., Doig A.G.: An automatic method of solving discrete programming problems, Econometrica. v28, 1960, pp 497-520.
[2] Little J.D.C., Murty K.G., Sweeney D.W., Karel C: An algorithm for the traveling salesman problem, Operations Research, v11, 1963, pp 972-989.
[3] Wolsey, Laurence A.: Integer Programming, John Wiley & Sons, 1998.
[4] Nemhauser, George L. and Laurence A. Wolsey: Integer and Combinatorial Optimization, John Wiley & Sons, 1998.
[5] Hillier, Frederick S. and Lieberman Gerald J.: Introduction to Operations Research, McGraw-Hill, 2001.
[6] Optimization Toolbox. User 's Guide, Version 3.0., The MathWorks, Inc., 2004.
[7] Anbil R., Gelman E., Patty B., Tanga R.: Recent Advances in Crew-Pairing Optimization at American Airlines, Interfaces, 21, no. 1 (1990): 62-74.
[8] Spenser T., Brigandi A., Dargon D., Sheehan M.: ATT's Telemarketing Site Selection System Offers' Customer Support, Interface, 20, no. 1 (Jan.-Feb., 1990).

LOOK-AND-FEEL REALIZATION OF INTEGER PROGRAMMING PROBLEMS

Abstract: The paper is dedicated to the problems of realization a new optimization function **bintprog** as inseparable part of *Optimization Toolbox 3.0*, pack *MATLAB* 7. Introduction of the new function essentially extends the scale of optimization assignments that should be solved, because it confirms the fact of realizing the branch-and-bound method. Investigation of this methods is included in many of educational courses and gives base to solving most of integer programming problems. Some representative results of tests are given at the end of the paper.

Keywords: integer programming, branch-and-bound method, deciding whether to branch

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Tomasz Rybak[1]

USING TEMPORAL PROCESS MODEL TO RECOVER LOST DATA

Abstract: Article describes data gathered during 23rd Chaos Communication Congress held in Berlin in December 2006. It presents characteristics of data set describing movements of participants in conference venue and errors present in it. The main part of article is description of attempts of recovering lost data, problems with it, and how different information present in data set can help with restoring lost parts. To recover lost data spatial dependencies and temporal model of Sputnik system were used.

Keywords: temporal data, spatial data, data mining, data recovery

1. Introduction to Sputnik system

Analysing human interactions is in center of interest of social sciences. It is usually done by employing questionnaires ([1]), whether filled in on paper or by using computers. This requires, however, that participants remember details of their behaviours, which is not always possible with required details.

Sputnik is Radio Frequency Identifier (RFID) system intended to trace people in small areas and buildings. Each person is wearing tag that transmits its identifier in regular time intervals to allow to store this persons position at those precise moments. System was used during 23rd Chaos Communication Conference (23C3) held in December 2006 in Berlin. This article describes analysis performed on data from 23C3.

Unlike ordinary RFID systems, Sputnik uses active tags; they are equipped in battery and transmit data whatever there is reader listening to it or not. Tag range in buildings is up to the 10m even through dry walls. Concrete walls tend to block the signal. Because transmission occurs at 2.4GHz, human body decreases power of signal by about 50%; transmissions from WiFi and Bluetooth devices using this frequency range can disturb occurring communication.

Rag has control over transmission power and can send signals varying in strength because of having independent power source. This allows for estimating distance

[1] Faculty of Computer Science, Bialystok Technical University, Bialystok

from reader. During conference 25 readers were placed in conference center in such a way that in most cases more than one reader saw tag. This, because of possibility of estimating distance from reader, allows for estimation of position of tag.

Sputnik system is not the only one that is intended for tracing and tracking people. Because of reasons mentioned earlier and development in electronics similar systems are created and used. Vassili Kostakos placed Bluetooth readers in city of Leeds in late 2006, and recorded all identifiers of pedestrians ([6]). Nathan Eagle equipped 100 cellular phones in special software which recorded cell towers and nearby Bluetooth devices seen by those phones ([3]). Josua Smith et. al. ([9]) were using ordinary passive RFID to get activities of human in house. Their setup required putting RFID tag on each of items in home, and giving reader to each human in form of wrist bracelet.

Setup similar to described in article was presented in [5], where museum items was equipped in tags, and visitors could get readers which took role of tour guides.

Privacy concerns present when human movements is traced were described by Miako Okhubo et. al. in [8].

2. Data gathered during conference

Data gathered during 23C3 was made available as both XML and binary files.

XML file contains very small portion of gathered data; it has only 357974 entries, while full data set contains 11.1 million of observations. It does not contain information about readers used to calculate positions of tags. This omission is important, as about 1/3rd of observations has no meaningful position calculated, probably because in those cases there was not enough data to calculate them. Also XML file contains data from only few hours for each day of Congress.

XML file consists of "observation" tags with attributes

id ID of tag
time time of receiving radio transmission
position position of tag; (0, 0, 0) if unknown

XML file was not used in analysis because it lacked sequence numbers and identifiers of reading stations used to calculate positions of tags.

Binary file contained all data packets received from tags; it consisted of 1144232 values. Each packet contained time of reception, IP address of reading station, strength of signal and sequence number. Because of error in server software, identifiers of tags were not saved. It made analysis of behaviour impossible, so first concern was to recover sequences of packets produced by individual tags.

Entire data set was stored in relational database so reading and parsing file with data is needed only once, as those operations take long time. SQL offered by database allows for writing analysis algorithms in much simpler way than it would be written when parsing would be required. Basic table used to store received packets consists of columns storing identifier of tag (initially empty column), time of reception, sequence value, strength of received signal, station that received it and additional information (pressed button, etc.).

Created database can be seen as temporal, and when looking at XML data containing position of tags also as spatial one. Such databases store information about presence of phenomenas in space and time. This database stores information about presence of tags (and probably persons wearing them) at the place at the moment. Activities performed using tags, like pressing button, are also stored in it. Additional spatial data, like geometry of building and rooms where events were held, and temporal data (schedule of Congress) can be used later for more sophisticated analysis.

Table containing data from 23C3 occupies about 700MB on hard drive. Data types used to store sequence and time values occupy 8 bytes each; index for each of those columns takes 250MB. Sequence identifier is stored as 4 byte integer and its index takes about 130MB. Creation of indexes is necessary for performance reasons, because size of database is larger than available RAM and analysis speed depends on disk performance.

3. Analysis of data

Data was stored and worked upon by using relational database. Article [2] describes basic algorithms and techniques used for data mining in relational databases, like regression (including linear) and decision trees.

To understand further operations one needs to understand how tags work. In each transmission tag sends its ID and strength of signal it uses to transmit. Each transmission is encrypted using XXTEA. To avoid replay attacks it is necessary that each send packet differs from previous ones — so simple ever-increasing counter was added to the tag. Base station discards all packages with counter numbers less than the one seen previously. To avoid problems with reset of tag (i.e. removing battery) which sets counter value to 0, counter was split into two parts. Higher word (16 bits) is saved during reset, and lower (also 16 bits) is not. After reset tag increases higher word and lower is set to 0 so counter value always grows. This feature means that gaps may occur in counter values sequences when tag battery is removed. To avoid problems with transmitting packets from two tags at the same time each tag transmits and sleeps for random time, from 2 to 4 seconds.

Figure 1 shows number of packets seen in entire system in each minute. It can be seen that during day there is high activity, and during night hours activity is very low, because most of attendees left conference venue.

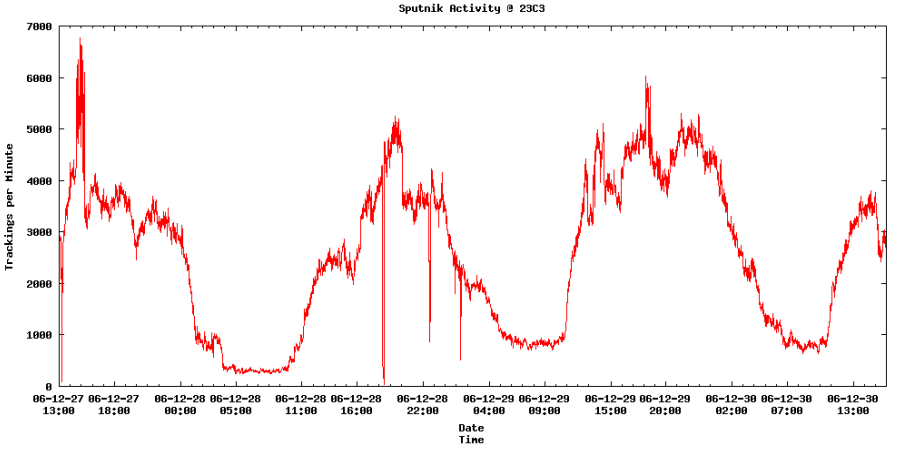


Fig. 1. Number of packets read during one minute

Table shows number of packets with particular strength of signal. Most of received signals were the strongest ones; only about 1.5% were from the weakest packets.

Strength	count
0	182874
85	568413
170	1167287
255	9225658

3.1 Rebuilding sequences

To be able to analyse data and gain some knowledge from it, sequences need to be restored. It requires joining single packets into sequences and then attaching unique number into each found sequence. Unfortunately original tag identifiers are lost and it is impossible to recover them; but even without them restoring sequences will allow for analysis of data.

Technique used for testing data mining algorithms, where part of data with known parameters is used to check correctness of proposed program, cannot be used

here, as no part of main data set contains identifiers. XML data set cannot be used to test because it contains neither reader identifier nor sequence number, which does not allow for joining those two sets.

There is not single parameter that decides whether sequence is correctly rebuild. But knowledge coming from observations of participants during Congress and analysis of source code used in tags can help with creation of heuristics and numerical parameters (like time periods) which should decide whether results are correct or not. Slope of created lines must lie in correct range and observations made using data must be correct according to physical reality. It means that one tag cannot be seen in two places at the same time, nor it may move faster than walking (running) human — move in very short time for few dozens of meters. Sequences also should be long, because tags were working all the time and most of places in BCC had coverage of readers.

3.2 Local algorithms

Local search for short sequences was used initially because global searching requires large amounts of processing time, memory and disk resources.

According to Sputnik behaviour model, one should be able to take first packet and then be able to find next one, that has next value of counter, and is 1 or 2 seconds from previous one. This ideal situation does not take into consideration gaps in sequences because of person leaving conference venue, or because one is not in the range of any readers, or when tag is transmitting too weak signal to be received by any of readers. However this is idea of finding local sequences; all functions described in this subsection are using this approach and add code dealing with gaps and choosing one packet that can be added to sequence when there is more than one.

The basic idea of algorithm for searching local sequences is to assume that packets are send once per 1.5s. Program needs to take all points from chosen period of few dozens seconds. Starting from the lowest counter value it tries to find the next value. In case of very close values of counter, difference of time is 1 or 2 seconds. In case of longer time distances, difference should be closer to 1.5s for every packet.

When more than one packet can be chosen to extend sequence conflict occurs and this problem must be resolved. Conflict may arise because either at the same time there are two different counter values, or the same value occurs at different moments. In case of either conflict we must choose only one packet to include in sequence, and discard another one. It needs to be noticed that conflict occurs when more than one packet can be added; it means that more than two packets may be involved in that conflict. The general case is presence of more than one sub-sequence (consisting of

one or more packets) that can extend existing sequence. Only one of them must be chosen, as adding all sub-sequences will destroy existing sequence by introducing decreases in either time or counter values. Sub-sequence may be chosen by taking into consideration length or resemblance to already existing sequence.

To be able to recreate sequences it is necessary to create all alternatives and then choose the best ones. All values of time and sequence counter are read in increasing order, and all points are treated as potential extension of sequences. If considered point can be added to sequence, it is. If not, conflict is detected. Previous value is removed from sequence, and both points are added to special list of alternatives. In such case each subsequent point is treated as extension not of main sequence, but alternative sub-sequences. If it can be added to all of them, alternatives are stored, and this point is added to main sequence. If it can be added to only some of sub-sequences, conflict still remains. If it cannot be added to any of sub-sequences, it is added as another alternative sub-sequence.

All found alternative sub-sequences are used to build optimal sequence. Each step of algorithm tries to build the longest and the smoothest sequence, with the smallest number of missing points. Slope of line is used to choose the best possible sub-sequence to add. Line with slope closest to 1.5 is chosen by using minimal square difference.

First program used to find sequences had time cost of $O(N^3)$, where N is number of packets. For each point it was finding whether any of other points can be added to the sequence by checking if equation $\Delta s = a\Delta t, 1.0 \leq a \leq 2.0$ was met. After finding all possible points it was generating all possible alternatives from all those chosen points. Because for every point from given interval it was checking all other points, time cost of this operation was $O(N^2)$. If any sequence was found, points belonging to it were removed from data set, and entire process was started from the beginning, giving time cost $O(N^3)$.

Improving speed of this algorithm came from observation that the longest sequences are made when starting from the lowest time and lowest counter values. Query was changed to return sorted result. Algorithm was changed to take first tuple, and try to find all other packets that can make sequence with the first one. If sequence was found, it was removed from data set; if not, only the first tuple was removed. Each packet was considered once as start of the sequence and all other points were used to build sequence with it; this gives time cost $O(N^2)$.

Local algorithms were not able to find long enough sequences. Although few found sequences were rather long (up to 20 packets for 1 minute), the most found were only consisting of only 2 or 3 packets. Attempts of joining sequences coming

from different time intervals (consecutive minutes) were not successful because of large gaps between end of one sequence and beginning of the next one.

3.3 Global algorithms

Because local searching did not lead to useful sequences, global algorithms were used. All calculations were done inside blocks of counter values of size 65536 to avoid problems with gaps caused by counter reset.

Starting from the point with lowest values of time and sequence counter program draws lines through all other points in range. Line that includes the most points is chosen; all those points are put into one sequence, and removed from the original set of points. In each case algorithm chooses line consisting of the largest number of points; this means that it is greedy algorithm.

Histogram of all slopes with bucket of size 0.1 is used to choose the best line coefficient. The largest count of points belonging to one class (bucket) reveals coefficient of line that, when drawn, will include the most points. This line should be chosen as the next sequence. This approach can be seen as using "naive Bayes" classification, as the most populated class is used — but on the other hand not all points are selected for this sequence. To be sure that no point is left without sequence because of rounding errors, range of slopes is used instead of simple comparison: all points that are on lines with slopes differing less than ± 0.3 from chosen slope are included into created sequence.

Because for each point all other points are used to calculate slopes and then all points that are in right coefficient range are chosen, time cost of this algorithm is $O(N^2)$.

Program was running for about 72h on AMD Duron 1.3GHz with 768MB RAM and single HDD IDE 7200RPM. It was disk-constraint, probably because of database size larger than available RAM; processor was not much used. Update of packets to include them in the sequence was done by one SQL query for one sequence. This required loading large part of table, but on the other hand allowed for database to optimise access to this table.

Many long sequences were found; only 4000 points (out of 11.1 million) were not included in sequence. However generated lines had rather strange coefficients; besides ordinary 1.4 or 2.5, one could find values 0.1, 0.4, 0.5, 9.9, 10.0, 8.1, ... Amongst sequences generated by this algorithm some were the proper ones, but many other were wrong. Those incorrect ones had line coefficient that could not be generated by firmware installed in tags, or they had points that came from two differ-

ent lines, similarly to Figure 6 or Figure 7; the most noticeable example is shown in Figure 2.

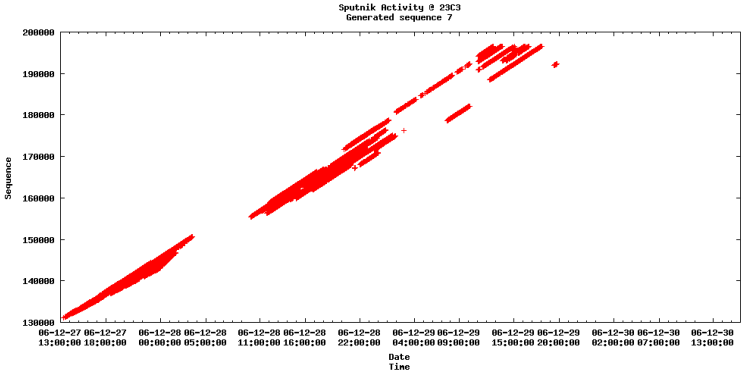


Fig. 2. Generated sequence; first set, number 7

Figure 2 shows sequence that looks like collage of many sequences. This shows the main problem of algorithm: range of allowed coefficients is too wide so too many points are added to sequence. The farther away from the first point, the more obvious it is — Figure 3 shows sequence that in the beginning is correct and gets incorrect in the end. The first part of this sequence should be preserved, and after it sequence should end; remaining points should be used to create another sequences.

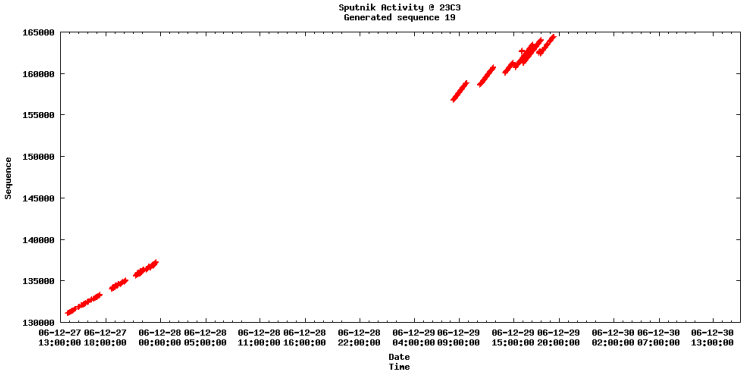


Fig. 3. Generated sequence; first set, number 19

Sequences that contain point that should belong to many different sequences unveil the main problem with algorithm, which is caused by to wide range of possible coefficient values and incorrect slopes of generated lines.

Two histograms of line coefficients were generated; one with buckets of size of 0.1 (Figure 4), and another with buckets of size of 0.001 (Figure 5). As can be seen, first histogram presents false situation; number of points in many lines that consist of small number of points but have close coefficient values is able to outnumber one line with high number of points. This leads to situation where instead of long line the short one is chosen.

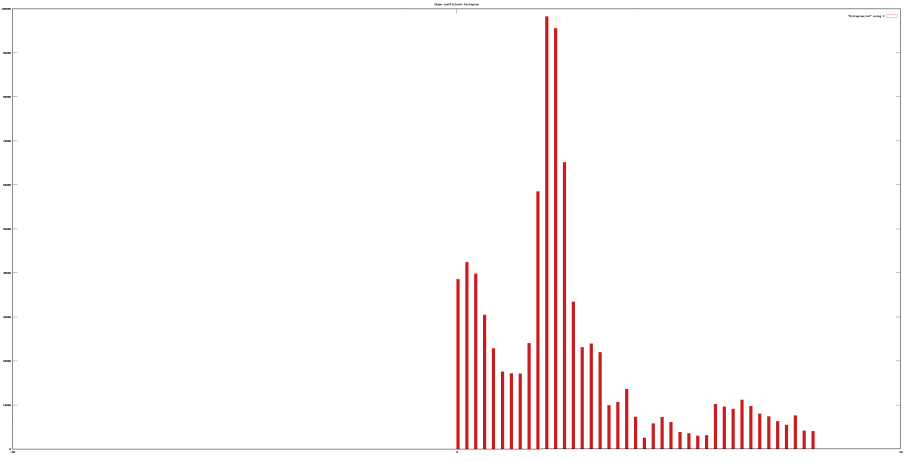


Fig. 4. Coefficients histogram for 10 buckets

Figure 4 and Figure 5 show that experiments are needed to find proper values of parameters, as incorrect values lead to wrong results.

Improvements of algorithm were necessary to obtain better results. Size of histogram class was changed to 0.001 to avoid problems with many lines joining into one. Histogram was calculated for slopes from range 1.0 to 5.0, instead of from 0.0 to 10.0. Range of coefficients of points that were used to build lines was changed from ± 0.3 to ± 0.001. This however caused gap at the beginning of each sequence, as rounding errors in the first few minutes of sequence caused slope of those initial points to be not close enough to the ideal to be included in chosen range.

SQL aggregate function was used to choose point which was included in sequence in case of presence of more than one counter value at the same time. It re-

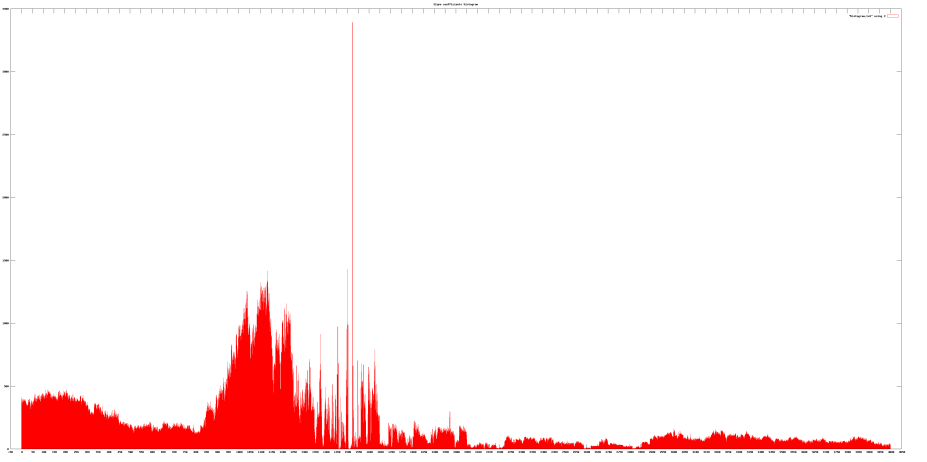


Fig. 5. Coefficients histogram for 1000 buckets

quired grouping by time in SQL query. Point which distance from the chosen slope was the smallest was chosen.

Program was running on the same machine, but it was very slow. It was running for about one week before it was stopped. It could not finish calculating sequences belonging to the first large data set ($counter \in < 2 * 65536; 3 * 65536 >$) so it was stopped and run for later counter values. It did not leave the next counter values block for another week. Its usage of disk subsystem and processor time was more balanced that for previous one. Its long working time may come from performing more calculations, using custom aggregate function, and updating information about sequences as many individual queries instead of one bulk query.

First few generated sequences were big, but later ones were getting smaller and smaller, down to dozen points. Sequences had some points missing; probably some points that should be included into those sequences were not added because of rounding errors and too narrow range of allowed coefficients.

Algorithm was joining sequences in spite of aggregate function which was used to guard against it. Data analysis shown that some sequences had errors (Figure 6, Figure 7), but they were more subtle and could not easily be seen on the graphs.

Figure 6 shows two distinct sequences that are joined. Their points are in allowed slope range, and they are interlaced, so even aggregate function can not remove them.

Figure 7 shows three distinct sequences joined into one. They have similar slope and their points lie in allowed range, so they are joined together, even though that points should create distinct lines.

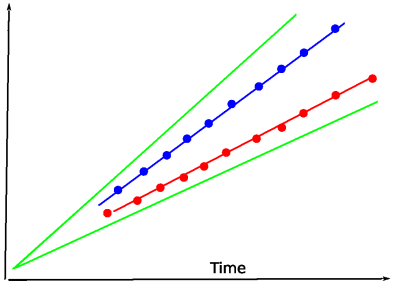


Fig. 6. Interlaced sequences

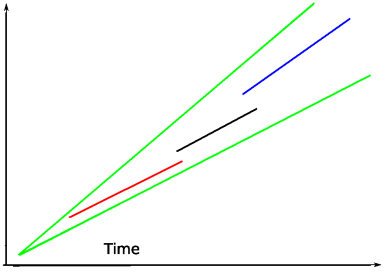


Fig. 7. Collinear sequences

Sequences that are joined but have different slopes can be detected by calculating difference of slopes between consecutive points, similarly to differentiating. The long sequence of differences of the same sign followed by long sequence of differences of another sign may suggest coupling of different sequences. To avoid splitting one sequence into many short ones number of points that have the same sign of difference between slopes and absolute difference between those slopes can be used. If both of those parameters are small, all points belong to one sequence.

This is similar approach to shown in [4], where authors were trying to find event that separates two lines. Unlike in [4] here splitting event comes not from experiment, but is interpreted by us as coupling of two lines. So "event" is just point where two lines that are joined and need to be separated.

Basing on experiments, correct values of parameters were found. While value 0.001 as size of histogram classes was found to be correct, the same value was to small as border of line coefficient. Experiments shown that value 0.01 was giving much better results. Solution to problem of points belonging to other sequences that might be included by larger border will be described later.

As mentioned earlier, because of rounding errors at the beginning of sequence coefficients do not have the same values as coefficients for further points. It is necessary to have wider allowed range of slopes in the beginning and more narrow near the end. This can be accomplished by sigmoid function. Function $0.01 + \frac{0.09}{1+e^{(x-500)/100}}$ was used in program. At the distance 0 it generates border of 0.1; its value was getting smaller to reach 0.01 for argument of 1000. Because of very large exponential values, mathematical exception is generated for arguments greater than about 70000.

Using only time and counter values gives us choice of either having too wide range and having joined sequences, or having too narrow range and leaving some points out, without guarantee that appropriate points are included in sequence. Additional data must be used in searching for good sequences.

First choice of additional parameter is strength of signal. Each tag changes strength of sent signal in sequence of values 0x00, 0xff, 0x55, 0xff, 0xaa, 0xff, 0xff, 0xff. First problem is that 5 out of 8 values are the same 0xff, so it is difficult to determine where in sequence of signal strengths particular point is. However analysing of source code and Sputnik data revealed that strength of signal is not distinctive between tags. Each tag starts at the same strength sequence point, so there is no variability between sequences. If more than one point has the same counter value, they also have the same strength of signal. It can not be used to distinguish between different sequences.

Because strength of signal could not be used to help to generate sequences, stations that received signal from tag were used. The main assumption was that set of

seen readers did not change from one point to another if those points were close in time. To keep algorithm simple only list of seen stations was considered, not their distribution in space. Similarity was defined as number of stations in both sets, divided by size of joined sets; it is known as Jaccard coefficient and defined as $c(A,B) = \frac{|A \cap B|}{|A \cup B|}$.

To avoid errors shown in Figure 6 algorithm was changed to retrieve all potential points that could be added to generated sequence and choose the best one. This approach is return to the idea of generating alternative sub-sequences used in local algorithm. Points that are part of one sequence have condition $(T_1 > T_0 \wedge S_1 > S_0)$ met. Program creates all possible sub-sequence from point that have not met this condition and then chooses the best one. To choose the best it locally compares lengths, slopes of sub-sequences and reading stations seen by all sub-sequences and chooses one that is the most similar to main sequence (pseudo-code is shown in Figure 8).

Last version of global algorithm differs from previous ones in more than only numerical parameters. Instead of using constant range, sigmoid function is used to include more points in the beginning of sequence. All points are read from database, and program builds alternative sequences from them. Instead of using custom aggregate function to choose only one point, standard function aggregating all seen readers into array is used. This array is later used to choose the best points to include into sequence. The last change is breaking line if it is discovered that created line has high probability of being two different lines.

The line that has the most points laying on it is chosen using histogram. Then all points that are on lines that differ less than border range are read from database; range is not constant, but sigmoid function is used to get it narrower in the end of block. All those points are used to build sequence; alternative sub-sequences are generated if needed. To find which sub-sequence should be added to main sequence, similarity of slopes is used. Special function was created to return similarity of sets of seen stations; it returns number from range $< 0.0; 1.0 >$:

1. If strength of signals are the same for two points, similarity is calculated by dividing number of stations seen by both points by number of all seen stations (Jaccard coefficient)
2. If first of points has more power than second, similarity is calculated as number of stations seen by latter by number of all readers seen by former (stronger)

Sub-sequence which is more similar to preceding point is chosen.

The very important part of function choosing sequences is condition allowing only sub-sequences which time and counter values are greater than already existing in sequence to be considered as alternatives. This protects from the problem of having improper sequence in case when one alternative proceeds another.

The final difference is new function `Break` which determines whether generated sequence should end. This protects from situation seen in Figures 3, and 7. This function (deciding whether to break one sequence into two) takes into consideration six parameters:

1. Similarity of sets of seen stations for both lines
2. Slope of first line
3. Slope of second line
4. Whether they differ too much to avoid stair-step-case line; from experiments 10 times difference is too much
5. Whether there is much difference in time of consecutive points
6. Whether slope of any of lines differs too much from global scope of line we are currently building

If more than 50% of those conditions is met, line is split.

```
for s in "SELECT DISTINCT sequence FROM sputnik WHERE id IS NULL":
        for t in "SELECT DISTINCT time FROM sputnik WHERE id IS NULL AND sequence = i
                Calculate histogram of slopes of all lines starting at s, t
                slope = max(histogram)
                points = find all x, y that slope-delta <= (y-t)/(x-s) <= slope+delta
                lines = CreateAllPossibleLines(points)
                ResultLine = []
# Choose the best line:
                for l0, l1 in lines:
                        if similarity(l0, ResultLine) >  similarity(l1, ResultLine):
                                ResultLine.append(l0)
                        else:
                                ResultLine.append(l1)
                for point in ResultLine:
                        if ShouldBreak(ResultLine[0]-point, point-ResultLine[-1]):
                                Split ResultLine into two at point
```

Fig. 8. Final algorithm for finding lines

Program was run on different machine than previous ones. It was running for about 100h on 64 bit AMD 3400+ with 1GB of RAM and one SATA HDD 7200RPM. It was stopped by FPU error in sigmoid function for large values of counter. 10.6 million points was used in generated sequences. Over 1600 sequences were made from more than 1000 points.

Because some of generated sequences were short, the next step should be joining them. One solution is to try to join existing sequences, another could be trying to extend sequences by points not belonging to any sequence. But problem with joining is choosing which sequence to join with each another. No experiments with joining sequences were performed.

4. Final remarks

Regenerating sequences was started with assumption that each tag sends packet every 1.5s. This lead to setting coefficient range from 1.0 to 2.0. Because this was not giving good results in local algorithms, and by observing scatter plots, global algorithms with ragne 0.0 to 10.0 was used; and later, basing on analysing source code of Sputnik firmware, range of coefficients was changed to 1.0 to 5.0. Source code of firmware contains two calls of sleep function. One sleeps for 2s, and another for random period from 0s to 2s. This gives range of line slopes from 2s to 4s. But because second sleep function parameter is random value, there should be no straight line! However scatter plots reveals many lines. So either Sputnik data contains so many points that one can draw any line, or function rand() returns numbers that are not random. Basing on analysing packets generated by single tag, second possibility is believed to be true.

No physical (or geometrical) model was taken into consideration during generating sequences. No distance between stations or speed of movement was analysed. This could give better results in sequences, by limiting point to only those that are in range to reach from previous point. On the other hand this approach would require calculating position of each tag in every moment.

XML data set proves that it is possible to calculate position of tag. Tags send packets with different signal strength to allow for estimation of distance from reader. This estimation bases on negative knowledge. If reader is unable to read signal with small strength it means that tag is far away from it. So having few packets it is possible to calculate minimal and maximal distance tag is from reader. Power of signal was set so next level of power increases radius two times. This gives two spheres with small and large radius; person is between them. When data from few readers is known, it is possible to calculate common fragment of space where all those spheres intersect — this is position of tag. This approach requires knowing exact positions of readers.

Human body decreases strength of signal. This decreases precision of estimating position of tag, but could be used to calculate direction person has, assuming that tag is worn in the front. Range would not be sphere, but two hemispheres, larger in the front and smaller in the back. This would require performing more calculations (two times for each reader), but as there is no situation when all readers see one tag,

it would not be impossible. Direction could be proven when person moves in this direction, again with assumption that person walks forwards, not backwards.

The most interesting analysis is looking for connections and similarities between attendees. This can be done by looking for people that attended similar talks. Those people may not even know each other but have common interests.

Another research area is looking for friends. Friends can be defined as people that stay together; they tend to be together not only during talks, but also and especially during breaks. If two people are close during most breaks, they are close friends. If they are close for some times, and not close for other moments, they may be colleagues. Or they may just stay in the same queue for pizza. However here the most important is relative position (distance between people), not exact position of tags.

Article by Kumar et al. ([7]) describes analysis done on data gathered from online messaging system. Authors describe finding friends and fellows in online communities, and process of creation of groups and cliques. In my opinion it is desirable to use Sputnik in similar way.

The other possibility is to try to join internet and real life friends. But this assumes that it is possible to gather online data; companies do not make it available, as it is source of advertisement revenue.

Although observing movements of people in real time is interesting, it also raises many privacy concerns. Article shows that issues raised in [8] are real; it shows that there is even more risks to privacy than previously were thought. Authors of [8] do not even raise question of restoring identity of tags worn by people. Presented here approach is similar to calculating identity of person from "cloud" described by Kostakos ([6]), where we can compute identity of person basing on some (changing) set of tags (RFID of watch, computer, suitcase). While Kostakos presents is as theoretical danger, this article shows that this is real concern and must be taken into consideration.

References

[1] P.S. Bearman, J. Moody, and K. Stovel. Chains of affection: The structure of adolescent romantic and sexual networks. *American Journal of Sociology*, 110(1):44–91, 2004.

[2] Sašo Džeroski. Multi-relational data mining: an introduction. *SIGKDD Explorations Newsletter*, 5(1):1–16, 2003.

[3] Nathan Eagle and Alex (Sandy) Pentland. Reality mining: sensing complex social systems. *Personal Ubiquitous Comput.*, 10(4):255–268, 2006.

[4] Valery Guralnik and Jaideep Srivastava. Event detection from time series data. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 33–42, New York, NY, USA, 1999. ACM.

[5] Sherry Hsi and Holly Fait. Rfid enhances visitors' museum experience at the exploratorium. *Communications of ACM*, 48(9):60–65, 2005.

[6] Vassilis Kostakos. The privacy implications of bluetooth. 2007.

[7] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 611–617, New York, NY, USA, 2006. ACM Press.

[8] Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Rfid privacy issues and technical challenges. *Communications of ACM*, 48(9):66–71, 2005.

[9] Joshua R. Smith, Kenneth P. Fishkin, Bing Jiang, Alexander Mamishev, Matthai Philipose, Adam D. Rea, Sumit Roy, and Kishore Sundara-Rajan. Rfid-based techniques for human-activity detection. *Communications of ACM*, 48(9):39–44, 2005.

UŻYCIE MODELU TEMPORALNEGO W CELU ODZYSKANIA UTRACONYCH DANYCH

Streszczenie: Artykuł przedstawia dane zgromadzone podczas konferencji Chaos Communication Congress która odbyła się w grudniu 2006r. w Berlinie. Dane pochodzą z systemu Sputnik który monitorował ruch uczestników konferencji. Pierwszy rozdział to któtka prezentacja danych oraz obecnych w nich błędów. Główną część artykułu stanowi opis prób odzyskania danych, które zostały utracone na skutek błędu w oprogramowaniu systemu Sputnik. Opisane są sposoby odzyskiwania tych danych oraz ich rezultaty. Do odzyskania został użyty temporalny model działania systemu oraz zależności przestrzenne obecne w danych.

Słowa kluczowe: dane temporalne, dane przestrzenne, analiza danych, odzyskiwanie danych

Artykul zrealizowano w ramach pracy badawczej S/WI/2/03.

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Bartosz Sokół[1]

TECHNIKI WYKRYWANIA USZKODZEŃ PAMIĘCI Z WYKORZYSTANIEM STOPNI SWOBODY TESTÓW KROKOWYCH

Streszczenie: Publikacja zawiera opis wybranych metod i technik wykrywania uszkodzeń pamięci z wykorzystaniem stopni swobody transparentnych testów krokowych. Główna uwaga została skupiona na uszkodzeniach uwarunkowanych zawartością typu Pattern Sensitive Faults (*PSF*) jako najtrudniejszych do wykrycia. Zaproponowane wykorzystanie stopni swobody testów krokowych przejawia się możliwością efektywnego przeprowadzenia transparentnego testowania i wykrywania uszkodzeń typu *PSF* przez proste testy krokowe i bazuje na możliwości wielokrotnego uruchomienia testu przy zmianach warunków początkowych (porządku adresowania) dla każdego uruchomienia. Wykorzystane i zaproponowane metody umożliwiają generowanie pełnych sekwencji adresowych oraz pozwalają na optymalny wybór adresów startowych przy wielokrotnym uruchomieniu testów krokowych. W pierwszej części pracy przedstawiona została problematyka testowania pamięci oraz stopnie swobody testów krokowych. Druga część pracy zawiera opis zaproponowanych rozwiązań wraz z wynikami wybranych eksperymentów.

Słowa kluczowe: testowanie pamięci, testy krokowe, uszkodzenia *PSF*, stopnie swobody

1. Wstęp

Pamięci półprzewodnikowe (RAM) zawsze stanowiły jedną z ważniejszych części systemów cyfrowych. Znajdują one szerokie zastosowanie w układach obliczeniowych i sterujących, a także w systemach przetwarzania i przechowywania informacji. Współczesne technologie wytwarzania modułów RAM idą w kierunku zwiększenia stopnia integracji, zmniejszania rozmiarów elementów i zwiększenia gęstości ich rozmieszczenia. W tych warunkach wzrasta prawdopodobieństwo wystąpienia przypadkowych defektów w trakcie procesu produkcji i intensywność pojawiania się uszkodzeń podczas procesu eksploatacji układów RAM, co prowadzi do istotnego obniżenia niezawodności pracy całego systemu. Analiza dotychczasowych prac i badań, poświęconych metodom niedestrukcyjnego testowania modułów RAM ([2]-[4],[7,12]), pozwala zauważyć tendencję do wykorzystywania w konkretnym celu

[1] Wydział Informatyki, Politechnika Białostocka, Białystok

metod opartych na wykorzystywaniu testów krokowych, co uzasadnione jest wysoką skutecznością wykrywania tymi metodami uszkodzeń ze złożonością *O(N)*, gdzie *N* jest to rozmiar pamięci, i małymi narzutami sprzętowymi i programowymi na realizację.

Test krokowy składa się ze skończonej liczby faz. Każda faza testu krokowego składa się ze skończonej liczby poleceń odczytu i zapisu, z których wszystkie oddziałują na określoną komórkę przed przejściem do następnej komórki pamięci, określonej przez przyjęty sposób adresowania. Wysokie wymagania co do niezawodności i czasu pracy urządzeń powodują, że coraz szerzej rozpowszechnione są niedestrukcyjne modyfikacje testów krokowych [7], pozwalające na zachowanie zawartości testowanych komórek pamięci podczas przeprowadzania okresowego testowania w przerwach między normalnym funkcjonowaniem systemu pamięci. Podstawowym wymaganiem, stawianym testerom niedestrukcyjnym, jest obowiązkowy powrót obiektu do stanu wyjściowego po fazie testowania. Jako przykład testu krokowego może posłużyć szeroko wykorzystywany test *March C-*, którego transparentna wersja znajduje się poniżej.

$$\Uparrow (ra,wa^*);\Uparrow (ra^*,wa);\Downarrow (ra,wa^*);\Downarrow (ra^*,wa);\Updownarrow (ra)$$

$$\leftarrow P0 \rightarrow \leftarrow P1 \rightarrow \leftarrow P2 \rightarrow \leftarrow P3 \rightarrow \leftarrow P4 \rightarrow$$

gdzie: *P0...P4* – elementy krokowe / fazy

$\Updownarrow, \Uparrow, \Downarrow$ – kierunek adresacji

(ra,wa)...(ra)* - operacje w komórkach pamięci

$a \in \{0,1\}, a^*$ - wartość przeciwna do a

Przedstawiony algorytm składa się z pięciu elementów krokowych - zbiorów operacji odczytu *(r)* lub zapisu *(w)*, stosowanych do wszystkich komórek pamięci w określonym porządku. Argumenty operacji zapisu określone są przez binarne wartości zapisywane w komórkach pamięci. Argumenty operacji odczytu określają wartości, które powinny być odczytane ze sprawnych komórek pamięci.

Łącząc niedestrukcyjne testowanie z wykorzystaniem stopni swobody właściwych testom krokowym, możemy w łatwy sposób efektywnie testować pamięć i wykrywać złożone uszkodzenia pamięci ([9]-[11],[13]-[15]).

2. Funkcjonalne modele uszkodzeń pamięci RAM

Idealny test dla cyfrowych układów scalonych RAM powinien wykrywać wszystkie możliwe uszkodzenia w strukturze testowanego schematu. Opracowujący testy spotykają się jednak z niemożnością sprawdzenia układu na obecność ogromnej

liczby różnorodnych uszkodzeń w rozsądnym okresie czasu. Dlatego też w praktyce przyjęto budowanie testów na podstawie pewnych analitycznych modeli uszkodzeń [2,5], w ten czy inny sposób odzwierciedlających realne uszkodzenia układów RAM. Zaletą funkcjonalnego testowania jest to, że testy funkcjonalne i odpowiadające im modele uszkodzeń są zaprojektowane z wykorzystaniem ustalonych i zautomatyzowanych reguł [2].

Głównym rodzajem uszkodzeń rozważanych w niniejszej publikacji będą uszkodzenia uwarunkowane zawartością (*Pattern Sensitive Fault - PSF*) [1,2], które można rozpatrywać jako uogólnienie uszkodzeń sprzężeniowych. Niech uszkodzenie uwarunkowane zawartością zawiera *k* komórek agresorów (lub komórek sąsiednich). Wtedy, gdy *k-1* komórek agresorów zawiera określony kod, a zmiana stanu w *k*-tej komórce agresorze wprowadza w komórce ofierze (lub komórce bazowej) pewną określoną wartość, to mówi się o obecności aktywnego uszkodzenia *PSF* (*Active Pattern Sensitive Fault - APSF*). W przypadku pasywnego uszkodzenia *PSF* (*Passive Pattern Sensitive Fault - PPSF*) przy istnieniu określonego kodu w komórkach agresorach komórka ofiara nie może zmienić swego stanu. Przy statycznym uszkodzeniu *PSF* (*Static Pattern Sensitive Fault - SPSF*) określony kod, zawarty w komórkach agresorach, wprowadza w komórce ofierze określoną wartość, która nie może być zmieniona bez uprzedniej zmiany kodu w komórkach agresorach [2]. W praktyce wykorzystuje się uszkodzenia typu *PSF* z ograniczoną liczbą komórek agresorów. Z reguły są to: trzy, pięć lub dziewięć komórek sąsiednich (*PSF3, PSF5, PSF9*) [6].

3. Stopnie swobody testów krokowych

Każdy algorytm testów krokowych może zostać wykorzystany na różne sposoby i nadal będzie skuteczny przy wykrywaniu jego uszkodzeń docelowych. Stopnie swobody (*ang. Degrees of Freedom - DOF*) właściwe dla testów krokowych mogą być wykorzystane do skonstruowania testów i metod, które umożliwią zwiększenie ich skuteczności zarówno przy wykrywaniu ich uszkodzeń docelowych, jak i przy innych uszkodzeniach dotychczas niewykrywanych.

Jak wiemy, testy krokowe używają sekwencji adresów nazywanych wzrastającymi i malejącymi, oznaczanych odpowiednio przez: $\Uparrow$ oraz $\Downarrow$, które nie muszą być obowiązkowo sekwencjami licznikowymi. Stąd można zdefiniować następujące stopnie swobody [8]:

DOF I: Każda przypadkowa sekwencja adresów może być definiowana jako sekwencja wzrastająca, pod warunkiem, że wszystkie możliwe adresy wystąpią dokładnie tylko raz.

DOF II: Sekwencja adresów dla fazy inicjalizacji może być dowolnie wybrana, pod warunkiem, że wszystkie adresy wystąpią przynajmniej raz.

Jak to zostało opisane powyżej, testy krokowe składają się z wielu krokowych faz testowych, w których zbiory operacji odczytu (r) i zapisu (w) przeprowadzane są na wszystkich komórkach pamięci podczas pełnej wzrastającej lub malejącej sekwencji adresów. Pozostałe stopnie swobody właściwe testom krokowym zostały opisane w pracy [8].

W kolejnym rozdziale zaproponowane będą metody wykorzystania stopni swobody w transparentnym testowaniu pamięci do wykrywania złożonych uszkodzeń pamięci, w szczególności typu *PSF*. Zmiana porządku adresów to wykorzystanie *DOF I* i *II*, a równoczesna zmiana zawartości i porządku adresów to wykorzystanie pierwszych czterech stopni swobody.

4. Wykrywanie uszkodzeń pamięci z wykorzystaniem stopni swobody

Poniżej zaprezentowane będą nowe metody i techniki wykorzystywane przy transparentnym testowaniu pamięci do wykrywania złożonych uszkodzeń pamięci.

4.1 Porównanie sekwencji adresowych

Główna idea zaproponowanego rozwiązania polega na zastosowaniu tego samego testu krokowego okresowo z różnymi sekwencjami adresów.

Dla najprostszego przypadku niech będzie to dowolny test krokowy uruchomiony dwukrotnie z różną sekwencją adresów. Pojawia się pytanie, które z dwóch sekwencji *A1* i *A2* muszą zostać użyte, aby osiągnąć najwyższe pokrycie uszkodzeń?

Porównajmy dwie sekwencje adresów *A1* i *A2* w celu wyciągnięcia wniosków co do przykładowych najlepszych zbiorów sekwencji adresów, dzięki którym osiągnięte będzie wysokie pokrycie uszkodzeń. Kandydaci do tego rodzaju zbioru muszą być jak najbardziej różni. Oznacza to, że nie powinno być żadnych wspólnych podzbiorów adresów na tych samych pozycjach w obu sekwencjach oraz wszystkie adresy na tych samych pozycjach w obu przypadkach muszą mieć jak największą różnicę pomiędzy dwoma adresami [11].

W wypadku standardowej sekwencji licznikowej, przykładem mogą być dwie sekwencje adresów, pierwsza ze wzrastającym porządkiem adresów, zaczynając od adresu z samymi zerami 000…00, 000…01, …, 111…11, druga z malejącym porządkiem adresów, zaczynając od adresu z samymi jedynkami 111…11, 111…10, …, 000…00. Dla $m = 2$ mamy $S_{ap}\#1$ = 00,01,10,11 i $S_{ap}\#24$ = 11,10,01,00.

Równocześnie S_{ap}#1 i S_{ap}#7 = 01,00,10,11 nie są całkowicie różne, ponieważ na trzeciej i czwartej pozycji występują te same adresy, mianowicie 10 i 11 [13].

Pozwólmy sobie na zaproponowanie arytmetycznej odległości $AD(A_k,A_j)$ dwóch sekwencji A_k i A_j, jako charakterystyki numerycznej do oszacowania jak różne są dwie sekwencje adresów:

$$AD(A_k,A_j) = \sum_{i=0}^{2^m-1} |A_k(i) - A_j(i)| \tag{1}$$

Dla dwóch określonych sekwencji adresów: $A_k = 2^m-1, 2^m-2, 2^m-3, \ldots, 0$ i $A_j = 0,1,2,\ldots,2^m-1$, ostatnie równanie przyjmuje postać:

$$AD(A_k,A_j) = \sum_{i=0}^{2^m-1} |2^m - 2i - 1| = 2^{2m-1} \tag{2}$$

Na podstawie proponowanej odległości arytmetycznej $AD(A_k,A_j)$ ostatnia wartość 2^{2m-1} wygląda na maksymalną możliwą wartość $AD_{max}(A_k,A_j) = 2^{2m-1}$, natomiast minimalna wartość wynosi $AD_{min}(A_k,A_j) = 2$. I rzeczywiście dla $m = 1$ $AD_{max}(A_k,A_j) = 2^{22-1} = 8$ (patrz S_{ap}#1 i S_{ap}#24) oraz $AD_{min}(A_k,A_j) = 2$ dla S_{ap}#1 i S_{ap}#7.

Dla realistycznego algorytmu opisanego w [13] przez:

$$A_{md} = a_1^{\lambda_1} a_2^{\lambda_2} a_3^{\lambda_3} \ldots a_m^{\lambda_m} \tag{3}$$

zapiszmy dwa następujące twierdzenia, których dowody można znaleźć w [11]. Aby uprościć poniższe rozumowanie, pozwólmy na zdefiniowanie standardowej sekwencji licznikowej (0,1,2,...,2^m-1) jako $A_{st} = a_1 a_2 a_3 \ldots a_m$.

Twierdzenie 1 *Odległość arytmetyczna AD(A_{st},A) gdzie* $A = a_1 a_2 a_3 \ldots a_{j-1} a_j^* a_{j+1} \ldots a_{m-1} a_m$, *z* $\lambda_k = 0$ *dla* $k \neq j$ *i* $\lambda_j = 1$ *jest obliczana jako:*

$$AD(A_{st},A) = 2^{2m-j} \tag{4}$$

Twierdzenie 2 *Odległość arytmetyczna AD(A_{st},A) gdzie* $A = a_1 a_2 a_3 \ldots a_{j-1} a_j^* a_{j+1}^{\lambda_{j+1}} \ldots a_{m-1}^{\lambda_{m-1}} a_m^{\lambda_m}$, *z* $\lambda_k = 0$ *dla* $k < j$, $\lambda_j = 1$ *i* $\lambda_k \in \{0,1\}$ *dla* $k > j$ *jest obliczana jako:*

$$AD(A_{st},A) = 2^{2m-j} \tag{5}$$

Jako przykład, obliczmy odległość arytmetyczną dla przypadku, gdy m = 4.
Niech $A = a_1^* a_2 a_3^* a_4^*$, wtedy $AD(A_{st},A) = |0-11| + |1-10| + |2-9| + |3-8| + |4-15| + |5-14| + |6-13| + |7-12| + |8-3| + |9-2| + |10-1| + |11-0| + |12-7| +$

$|13-6|+|14-5|+|15-4| = 11+9+7+5+11+9+7+5+5+7+9+11+$
$+5+7+9+11 = 128 = 2^{2*4-1}$.

Pod względem zmodyfikowanych sekwencji adresów A_{st} może być rozważana jako zmodyfikowana sekwencja A_{md0} z $\lambda_1\lambda_2\lambda_3\ldots\lambda_m = 000\ldots0$ wygenerowana zgodnie z równaniem (3). Dla ogólnego przypadku A_{md0} może być dowolną sekwencją licznikową ze wszystkimi binarnymi wartościami $a_1a_2a_3\ldots a_m$, gdzie $a_i \in \{0,1\}$. Ostatnie dwa twierdzenia, wspólnie z powyższym przykładem pozwalają sformułować ogólne metody szacowania arytmetycznych odległości pomiędzy dwiema zmodyfikowanymi sekwencjami adresów, wygenerowanymi zgodnie z algorytmem (3).

Odległość arytmetyczna pomiędzy dwiema sekwencjami adresów $A_{md0} = a_1a_2a_3\ldots a_m$ i $A_{md1} = a_1a_2a_3\ldots a_{j-1}a_j^*a_{j+1}^{\lambda_{j+1}}\ldots a_{m-1}^{\lambda_{m-1}}a_m^{\lambda_m}$, gdzie A_{md0} jest dowolną sekwencją licznikową, jest definiowana jako:

$$AD(A_{md0},A_{md1}) = 2^{2m-j} \tag{6}$$

gdzie j jest to indeks najbardziej znaczącego bitu w ciągu a_j odwróconego w A_{md1} w porównaniu z sekwencją A_{md0}. Przykład umieszczony w poniższej tabeli potwierdza ostatnie stwierdzenie.

Tabela 1. Odległości pomiędzy różnymi sekwencjami adresowymi

i	$A_{md1}(i) = a_1a_2a_3$	$A_{md2}(i) = a_1^*a_2a_3^*$	$\lvert A_{md1}(i) - A_{md2}(i)\rvert$
0	101 (5)	000(0)	$\lvert5-0\rvert = 5$
1	010 (2)	111(7)	$\lvert2-7\rvert = 5$
2	000 (0)	101(5)	$\lvert0-5\rvert = 5$
3	100 (4)	001(1)	$\lvert4-1\rvert = 3$
4	110 (6)	011(3)	$\lvert6-3\rvert = 3$
5	001 (1)	100(4)	$\lvert1-4\rvert = 3$
6	011 (3)	110(6)	$\lvert3-6\rvert = 3$
7	111 (7)	010(2)	$\lvert7-2\rvert = 5$

AD(A_{md1},A_{md2}) = 32=2^{2*3-1}

Jak można zauważyć, odległość arytmetyczna *AD(A_{md1},A_{md2})* zależy tylko od liczby j najbardziej znaczącego odwróconego bitu w A_{md2} w porównaniu z A_{md1}.

Spróbujmy wykryć uszkodzenie typu Passive Pattern Sensitive Fault z pięcioma komórkami sąsiednimi *(PPSF5)*, używając różnych sekwencji adresowych wygenerowanych zgodnie z równaniem (3). Sprawdzone zostały wszystkie możliwe ułożenia

zarówno komórki bazowej jak też komórek sąsiednich. Do testowania wykorzystany został prosty test krokowy *MATS++*, który został uruchomiony dwukrotnie, za każdym razem z różnym porządkiem adresów. W pierwszej kolejności użyto sekwencji licznikowej, jako drugiej sekwencji użyto sekwencji z odwróconymi bitami na różnych pozycjach (gwiazdka oznacza odwrócony bit). Otrzymane wyniki są zaprezentowane w poniższej tablicy.

Tabela 2. Procent wykrycia uszkodzeń *PPSF5* z różnymi sekwencjami adresów i negacją bitów

i	Sekwencja adresowa	$\lvert A_{md1}(i) - A_{md2}(i)\rvert$	*PPSF5* (%)
1	$A_{md0}(i) = a_1a_2a_3$;		
	$A_{md1}(i) = a_1^*a_2a_3$	$32 = 2^{2*3-1}$	12,5
2	$A_{md0}(i) = a_1a_2a_3$;		
	$A_{md2}(i) = a_1a_2^*a_3$	$16 = 2^{2*3-2}$	11,61
3	$A_{md0}(i) = a_1a_2a_3$;		
	$A_{md3}(i) = a_1a_2a_3^*$	$8 = 2^{2*3-3}$	9,82
4	$A_{md0}(i) = a_1a_2a_3$;		
	$A_{md4}(i) = a_1a_2^*a_3^*$	$16 = 2^{2*3-2}$	12,32
5	$A_{md0}(i) = a_1a_2a_3$;		
	$A_{md5}(i) = a_1^*a_2^*a_3$	$32 = 2^{2*3-1}$	12,5
6	$A_{md0}(i) = a_1a_2a_3$;		
	$A_{md6}(i) = a_1^*a_2a_3^*$	$32 = 2^{2*3-1}$	12,5
7	$A_{md0}(i) = a_1a_2a_3$;		
	$A_{md7}(i) = a_1^*a_2^*a_3^*$	$32 = 2^{2*3-1}$	12,5

Najlepsze wyniki otrzymano w przypadku, gdy odwrócony został najbardziej znaczący bit, bez względu na inne odwrócone bity, i gdy odległość pomiędzy sekwencjami była największa.

Przedstawione powyżej wyniki ugruntowały twierdzenia 1 i 2, które mówiły, że odległość arytmetyczna pomiędzy dwiema kolejnymi sekwencjami adresów musi być duża, by można było wykryć uszkodzenia pamięci z dużym prawdopodobieństwem, i zależy od pozycji najbardziej znaczącego odwróconego bitu w drugiej sekwencji dla drugiego uruchomienia testu (patrz równanie (6)).

4.2 Optymalne adresowanie pamięci, dobór adresów początkowych

W rozdziale tym rozważane będą uszkodzenia *PPSF*. Każda komórka pamięci, spośród N komórek może być zaangażowana w uszkodzenie *PPSFk*. Notacja ta oznacza, że k przypadkowych komórek pamięci jest zaangażowanych w uszkodzenie *PPSF* a jedną z tych komórek jest komórka bazowa (*base cell - b*),

pozostałe *k-1* komórek to komórki wiążące (*neighbors - n*). Istnieje k klas różnych uszkodzeń *PPSFk*. Klasyfikacja ta zależy od uporządkowania w przestrzeni adresów oraz od pozycji wszystkich tych komórek w tej przestrzeni. Niech adresy pamięci $i_0, i_1, i_2, \ldots, i_{k-1}$, dla konkretnego uszkodzenia *PPSFk* będą posortowane w porządku wzrastającym w sposób następujący $i_0 < i_1 < i_2 < \ldots < i_{k-1}$. Wtedy każde uszkodzenie *PPSFk* będzie reprezentowane przez elementy $a_{i0}, a_{i1}, a_{i2}, \ldots, a_{i(k-1)}$ uporządkowane w przestrzeni adresów zgodnie ze wzrastającym porządkiem adresów komórek pamięci. Jedna spośród k komórek jest komórką bazową, więc istnieje k różnych uszkodzeń *PPSFk* zależnych od pozycji komórki bazowej. Oznacza to, że istnieje k klas uszkodzeń *PPSFk* zależnych od pozycji komórki bazowej. Na przykład dla $k = 5$ mamy pięć następujących klas: $b_{i0}, n_{i1}, n_{i2}, n_{i3}, n_{i4}; n_{i0}, b_{i1}, n_{i2}, n_{i3}, n_{i4}; n_{i0}, n_{i1}, b_{i2}, n_{i3}, n_{i4}; n_{i0}, n_{i1}, n_{i2}, b_{i3}, n_{i4}; n_{i0}, n_{i1}, n_{i2}, n_{i3}, b_{i4}$. Istnieje 2^{k-1} różnych kombinacji bitów dla zbioru bitów w komórkach wiążących. Dokładna liczba uszkodzeń *PPSFk* określona jest zgodnie z równaniem:

$$L(PPSFk) = k2^{k-1} * \binom{N}{k} \tag{7}$$

Ten sam rezultat otrzymujemy na podstawie następującej obserwacji. Komórka bazowa może przyjąć każdą spośród N możliwych pozycji wśród komórek pamięci, a dla dowolnej kombinacji bitów w komórkach wiążących istnieje 2^{k-1} różnych zbiorów bitów. Mamy więc:

$$L(PPSFk) = N * 2^{k-1} * \binom{N-1}{k-1} = k2^{k-1} * \binom{N}{k} \tag{8}$$

Ostatnie równanie pozwala na sformułowanie wniosku, że wśród wszystkich k klas danego uszkodzenia istnieje taka sama liczba uszkodzeń *PPSFk*. Przypuśćmy, że używamy testu *MATS+* $\{\Uparrow (ra, wa^*); \Downarrow (ra^*, wa);\}$ do testowania ośmio bitowej pamięci, w której zawartość początkowa jest równa samym zerom $A = a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7$
$= 00000000$. Wtedy kolejne stany testowanej pamięci przy wykorzystaniu testu *MATS+* będą wyglądały tak jak przedstawione w tablicy 3.

Należy zauważyć, że sekwencja adresów pamięci została wybrana na podstawie sekwencji licznikowej, a adresem startowym był adres $i_0 = 0$. Jak widać w powyższej tabeli, tylko jeden zbiór bitów, spośród wszystkich możliwych dla każdej aktywnej komórki, jest zaznaczony pochyloną czcionką. W rzeczywistości sprawdzamy (odczyt 0 zapis 1 podczas pierwszej fazy i odczyt 1 zapis 0 podczas drugiej fazy) komórkę a_0 w obu fazach dla tej samej zawartości w pozostałych komórkach. Aktywacja uszkodzenia *PPSFk* może wystąpić tylko w momencie sprawdzania

Tabela 3. Zawartość pamięci podczas testowania testem *MATS+* z różnym adresem startowym

Fazy testu *MATS+*	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	Fazy testu *MATS+*	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
	0	0	0	0	0	0	0	0									
	1	0	0	0	0	0	0	0		1	1	1	1	1	1	1	*0*
	1	*1*	0	0	0	0	0	0		1	1	1	1	1	1	*0*	0
	1	1	*1*	0	0	0	0	0		1	1	1	1	1	*0*	0	0
	1	1	1	*1*	0	0	0	0		1	1	1	1	*0*	0	0	0
$\Uparrow (ra, wa^*)$	1	1	1	1	*1*	0	0	0	$\Downarrow (ra^*, wa)$	1	1	1	*0*	0	0	0	0
	1	1	1	1	1	*1*	0	0		1	1	*0*	0	0	0	0	0
	1	1	1	1	1	1	*1*	0		1	*0*	0	0	0	0	0	0
	1	1	1	1	1	1	1	*1*		*0*	0	0	0	0	0	0	0
Fazy testu *MATS+*	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	Fazy testu *MATS+*	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
	0	0	0	0	0	0	0	0									
	0	*1*	0	0	0	0	0	0		*0*	1	1	1	1	1	1	1
	0	1	*1*	0	0	0	0	0		0	1	1	1	1	1	1	*0*
	0	1	1	*1*	0	0	0	0		0	1	1	1	1	1	*0*	0
	0	1	1	1	*1*	0	0	0		0	1	1	1	1	*0*	0	0
$\Uparrow (ra, wa^*)$	0	1	1	1	1	*1*	0	0	$\Downarrow (ra^*, wa)$	0	1	1	1	*0*	0	0	0
	0	1	1	1	1	1	*1*	0		0	1	1	*0*	0	0	0	0
	0	1	1	1	1	1	1	*1*		0	1	*0*	0	0	0	0	0
	1	1	1	1	1	1	1	1		0	*0*	0	0	0	0	0	0

komórki bazowej. Podsumowując widzimy, że wykrycie uszkodzenia *PPSFk* jest możliwe tylko dla jednej kombinacji bitów w pozostałych *k-1* komórkach wiążących spośród 2^{k-1} możliwych kombinacji. Dlatego też liczba *Q1(PPSFk(i_0))* uszkodzeń możliwych do wykrycia podczas pierwszego uruchomienia testu *MATS+* wynosi:

$$Q1(PPSFk(i_0)) = k * \binom{N}{k} \tag{9}$$

A pokrycie uszkodzeń (*ang. fault coverage FC*) dla testu *MATS+* wynosi:

$$FC1\_MATS+(PPSFk(i_0)) = \frac{Q1(PPSFk(i_0))}{L(PPSFk)}100\% = \frac{1}{2^{k-1}}100\% \tag{10}$$

Na przykład: *FC_MATS+(PPSF5)* = $(1/2^4)100\%$ = 6,25%. Pokrycie uszkodzeń opisane wzorem (10) jest poprawne dla każdego testu pamięci, z kolejnymi fazami ustawionymi tak jak w przypadku testu *MATS+*, mianowicie $\ldots(ra, \ldots, wa^*); (ra^*, \ldots, wa); \ldots$. W celu podniesienia możliwości wykrycia uszkodzeń *PSF* musimy stosować test z czteroma kolejnymi fazami $\ldots(ra^*, \ldots, wa); (ra, \ldots); \ldots (ra^*, \ldots, wa); (ra, \ldots)\ldots$ (na przykład test *March C*). W praktyce wystarczą mniej niż cztery fazy.

Dla testu *March C*, liczba *Q2(PPSFk(i_0))* wykrywanych uszkodzeń podczas pierwszego uruchomienia testu wynosi:

$$Q2(PPSFk(i_0)) = 2k * \binom{N}{k} \tag{11}$$

A pokrycie uszkodzeń *(FC)* dla testu *March C* wynosi:

$$FC2\_MarchC(PPSFk(i_0)) = \frac{Q2(PPSFk(i_0))}{L(PPSFk)}100\% = \frac{1}{2^{k-2}}100\% \tag{12}$$

W przypadku uszkodzenia *PPSF5* i testu *March C* mamy: FC_March C(*PPSF5*) = $(1/2^3)100\%$ = 12,5%. Oczywiście, dla takiej samej zawartości pamięci każdy test pamięci będzie wykrywał takie same uszkodzenia pamięci, a także taką samą ich liczbę. Bazując na dwóch powyższych przykładach, możemy rozważyć dwa różne przypadki. Przypadek pierwszy, gdy komórka bazowa ma stałą pozycję, a komórki wiążące są na przypadkowych pozycjach w ramach *N-1* komórek pamięci RAM. Drugi przypadek jest bardziej realistyczny, gdy zarówno komórka bazowa jak też komórki wiążące znajdują się na losowych pozycjach w ramach *N*-bitowej pamięci. Dla powyższych przypadków możemy sformułować kolejny problem.

Jak zwiększyć pokrycie uszkodzeń, bazując na krokowych testach pamięci i stałej zawartości?

Odpowiadając na postawione powyżej pytanie, wiemy, że dla stałych dwóch elementów: wybranego testu krokowego oraz zawartości pamięci podczas procedury testowania, jedyną możliwością zwiększenia pokrycia uszkodzeń jest wykorzystanie procedury generowania adresów. Zacznijmy od optymalizacji wartości adresu startowego. Rozważmy przypadek procedury testowej z wielokrotnym uruchomieniem testu krokowego. Bazując na stopniach swobody i stwierdzeniu, że liczba uszkodzeń wykrywanych przez test krokowy nie zależy od porządku adresów, przy kolejnym uruchomieniu testu możemy użyć nowego adresu startowego. Jako przykład rozważmy drugie uruchomienie testu *MATS+* z adresem początkowym równym i_1 = 1. Kolejne stany we wszystkich komórkach pamięci są zaprezentowane w drugiej części tablicy 3.

Ponownie jak poprzednio otrzymaliśmy tylko jedną kombinację bitów w ramach całej pamięci dla każdej aktywnej komórki bazowej *b*. Porównując powyższe wyniki z przypadkiem opisanym w pierwszej części tablicy 3, widzimy, że zostały wygenerowane nowe kombinacje bitów. Dla każdej komórki bazowej *b*, z wyjątkiem przypadku, gdy $b = a_0$, istnieją różne wartości w komórce a_0. Dla przypadku, gdy $b = a_0$, mamy *N-1* różnych wartości w zbiorze bitów w komórkach wiążących. Widzimy, że liczba wykrywanych uszkodzeń *PPSFk* jest taka sama w

obu przypadkach, istnieje jednak grupa uszkodzeń wykrywanych podczas pierwszego uruchomienia z adresem i_0 oraz nowe uszkodzenia *PPSFk* wykrywane tylko podczas drugiego uruchomienia. Na przykład, gdy komórka bazowa $b = a_1$, a adres początkowy $i_0 = 0$ (patrz górna część tablicy 3), mamy dwie klasy uszkodzeń *PPSF5*. Jest to 15 uszkodzeń $b_{i0}, n_{i1}, n_{i2}, n_{i3}, n_{i4} = b0000$ oraz 20 uszkodzeń $n_{i0}, b_{i1}, n_{i2}, n_{i3}, n_{i4} = 1b000$. Dla nowego adresu początkowego $i_1 = 1$ (patrz dolna część tablicy 3) mamy 15 uszkodzeń $b_{i0}, n_{i1}, n_{i2}, n_{i3}, n_{i4} = b0000$ oraz 20 nowych uszkodzeń $n_{i0}, b_{i1}, n_{i2}, n_{i3}, n_{i4} = 0b000$ niewykrytych dotychczas podczas pierwszego uruchomienia testu *MATS+*. Jeżeli komórka bazowa będzie miała inną pozycję, na przykład $b = a_3$, to dla adresu początkowego $i_0 = 0$ mamy cztery klasy uszkodzeń *PPSF5*, mianowicie jedno uszkodzenie $b_{i0}, n_{i1}, n_{i2}, n_{i3}, n_{i4} = b0000$, 12 uszkodzeń $n_{i0}, b_{i1}, n_{i2}, n_{i3}, n_{i4} = 1b000$, 18 uszkodzeń $n_{i0}, n_{i1}, b_{i2}, n_{i3}, n_{i4} = 11b00$ oraz 4 uszkodzenia $n_{i0}, n_{i1}, n_{i2}, b_{i3}, n_{i4} = 111b0$. Z nowym adresem początkowym $i_1 = 1$ podział uszkodzeń będzie inny.

Całkowicie inna sytuacja występuję w przypadku, gdy komórka bazowa jest równa $b = a_0$, a adres początkowy wynosi $i_1 = 1$. W tym przypadku wszystkie uszkodzenia wykrywane podczas drugiego uruchomienia testu będą inne w porównaniu z uszkodzeniami wykrytymi podczas pierwszego uruchomienia testu *MATS+* dla przypadku $i_0 = 0$. Jak widać, podczas pierwszego uruchomienia wykrywane są uszkodzenia *b0000* a podczas drugiego uruchomienia wykrywane są tylko uszkodzenia *b1111*. Jeśli wziąć pod uwagę wszystkie możliwe pozycje komórki bazowej, całkowita liczba nowych uszkodzeń *PPSF5* z nową adresacją i_1 wynosi 175. Jak łatwo pokazać, dla ogólnego przypadku, liczba dodatkowych uszkodzeń *PPSFk* wykrywanych podczas drugiego uruchomienia testu *MATS+* z adresacją $i_1 = 1$ może być oszacowana jako:

$$(N-1)\binom{N-2}{k-2} + \binom{N-1}{k-1} \tag{13}$$

Dla $i_1 = 2$ łatwo wykazać, że:

$$(N-2) * \left[\binom{2}{1} * \binom{N-3}{k-2} + \binom{2}{2} * \binom{N-3}{k-3}\right] + 2 * \left[\binom{N-2}{k-1} + \binom{N-2}{k-2}\right] \tag{14}$$

Należy zaznaczyć, że nowa adresacja pamięci i_1 dzieli wszystkie możliwe (N) komórki bazowe na dwie grupy. Podział jest wykonywany zgodnie z odległością $s = i_1 - i_0$ pomiędzy dwoma adresami startowymi (należy pamiętać, że $i_1 > i_0$). Zgodnie z tymi obliczeniami, całkowity zbiór wszystkich możliwych stałych pozycji komórki bazowej będzie podzielony na dwie grupy. Pierwsza grupa będzie miała s różnych bitów pośród *N-1* komórek pamięci (z wyjątkiem pozycji komórki bazowej) wygenerowanych podczas drugiego uruchomienia testu *MATS+* w porównaniu

z kombinacjami bitów otrzymanymi podczas pierwszego uruchomienia z adresem początkowym $i_0 = 0$. Liczba takich komórek bazowych jest określana przez *N-s*. Reszta komórek bazowych należy do drugiego zbioru z *N-s* różnymi bitami.

Jako przykład weźmy $i_1 = 3$. Dla tego przypadku mamy $s = i_1 - i_0$ = 3-0 = 3. Więc istnieje *N-s* = 8-3 = 5 komórek bazowych z s = 3 różnymi bitami w ramach wszystkich *N-1* = 8-1 = 7 komórek pamięci, jak to zostało pokazane w tablicy 3.

Nowe zbiory bitów będą generowane w komórkach wiążących dla każdej możliwej komórki bazowej, dzięki czemu powinny być wykrywane nowe uszkodzenia *PPSFk*. Liczba nowych uszkodzeń *PPSFk* może być określona przy pomocy odległości Hamminga pomiędzy zawartościami pamięci dla dwóch różnych adresów początkowych. W naszym przypadku odległość $s = i_1 - i_0$ pomiędzy dwoma adresami startowymi oraz wartość $N - s = (i_1 - i_0)$ równa się odległości Hamminga dla zbiorów bitów w komórkach wiążących dla tej samej komórki bazowej przy dwóch uruchomieniach testu *MATS+*. Na przykład, jeżeli odległość $s = i_1 - i_0$ = 2, oznacza to, że dla i_0 = 0, i_1 = 2, oraz dla każdego i_0 i i_1, gdzie $i_0 - i_1$ = 2, mamy 6 (*N-s* = 8-2 = 6) par zbiorów bitów: *((11b00000, 00b00000), (111b0000, 001b0000), (1111b000, 0011b000), (11111b00, 00111b00), (111111b0, 001111b0), (1111111b, 0011111b))*, dla których odległość Hamminga wynosi s = 2, oraz mamy s = 2 pary zbiorów bitów *((b0000000, b0111111), (1b000000, 1b011111))* z odległością *N*-2 = 6.

Dokładna liczba nowych uszkodzeń *PPSFk* wykrytych podczas drugiego uruchomienia testu *MATS+* dla $s > 1$ będzie wyliczona zgodnie z równaniem:

Q1(PPSFk(i_1))=

$$(N-2) * \sum_{i=1}^{min(s,(k-1))} \binom{s}{i} * \binom{N-s-1}{k-i-1} + s * \sum_{i}^{min(s,(k-1))} \binom{s-1}{i-1} * \binom{N-s}{k-i} \quad (15)$$

Analiza powyższego równania pozwala sformułować następujące wnioski. Aby osiągnąć wysokie pokrycie uszkodzeń podczas drugiego uruchomienia testu *MATS+*, musimy użyć porządku adresów i_1:

1. Pierwszy przypadek (stała pozycja komórki bazowej) - musimy otrzymać maksymalną odległość Hamminga pomiędzy zbiorami bitów w komórkach wiążących dla stałej pozycji komórki bazowej. Aby osiągnąć ten cel, nowy adres początkowy, przy drugim uruchomienia testu pamięci, powinien spełniać następujące równanie $i_1 - i_0 = b + 1$, gdzie b jest to numer komórki bazowej.
2. W drugim przypadku (komórka bazowa może przyjąć dowolną pozycję losową) musimy otrzymać, o ile to możliwe, maksymalną sumę *S_HD* odległości Hamminga

dla wszystkich pozycji komórki bazowej. Liczba ta może być policzona jako

$$S\_HD = s(N-s) + (N-s)s \tag{16}$$

Maksimum dla funkcji S_HD może być wyliczone z równania:

$$\frac{\delta(S\_HD)}{\delta s} = \frac{\delta(s(N-s)+(N-s)s)}{\delta s} = (N - \frac{s}{2}) = 0 \tag{17}$$

gdzie wynikiem jest $s = N/2$.

Wracając do naszego przykładu, dla wersji pierwszej mamy $i_1 - i_0 = s = b$. Oczywiście jest to najlepsze rozwiązanie, pozwalające zwiększy pokrycie uszkodzeń. Na przykład, jeżeli ustaloną pozycją komórki bazowej jest 3, to dla i_0 = 0 mamy *111b0000*. Wtedy z równania $i_1 - i_0 = b + 1$=3+1=4 możemy wziąć i_1=4. Łatwo można wykazać, że zaproponowane techniki mogą być rozszerzone na wielokrotne użycie testu pamięci z wykorzystaniem różnych adresów początkowych: $i_0, i_1, i_2, i_3, \ldots$.

5. Podsumowanie

Jak widać z przedstawionych powyżej informacji, wykorzystanie stopni swobody w transparentnym testowaniu pamięci pozwala zwiększyć efektywność testów krokowych przy wykrywaniu złożonych uszkodzeń pamięci. Odpowiednio dobierając sekwencję adresową i adres początkowy jesteśmy w stanie wykrywać z dużym prawdopodobieństwem uszkodzenia uwarunkowane zawartością (*PSF*) przy wykorzystaniu prostych testów krokowych. Dobór sekwencji adresowych i wybór adresu startowego był przedmiotem badań przedstawionych między innymi w pracach ([13] - [15]).

6. Podstawowe pojęcia

PSF - Uszkodzenie uwarunkowane zawartością (*ang. pattern sensitive fault - PSF*) - wartość (lub możliwość zmiany wartości) komórki *i* (komórki bazowej) zależy od wartości (lub ich zmian) wszystkich pozostałych komórek pamięci (komórek wiążących) składających się na to uszkodzenie.
błąd - (*ang. error*) niepoprawna wartość logiczna spowodowana występującym uszkodzeniem.
uszkodzenie - (*ang. fault*) fizyczny defekt powodujący, że wartości logiczne w układzie przyjmują niepoprawne wartości.

defekt - fizyczna zmiana struktury układu prowadząca do jego nieprawidłowego działania.
test krokowy - (*ang. march test*) test pamięci składający się ze skończonej liczby faz. Każda faza testu krokowego składa się ze skończonej liczby poleceń, z których wszystkie oddziałują na określoną komórkę przed przejściem do następnej komórki pamięci. Komórka następna określona jest poprzez sposób adresowania.
test transparentny - (*ang. transparent test*) test gwarantujący (przy poprawnie działającej pamięci), że zawartość pamięci po zakończeniu procesu testowania będzie identyczna z zawartością pamięci w momencie bezpośrednio poprzedzającym rozpoczęcie testu.

Literatura

[1] Cheng K.L., Wu C.W.: Neighbourhood Pattern Sensitive Fault Testing for Semiconductor Memories, Proc. VLSI Design/CAD, Pingtung, Aug. 2000, pp.401-404.

[2] Goor A.J. van de: Testing Semiconductor Memories: Theory and Practice, Chichester: John Wiley and Sons Ltd., 1991.

[3] Goor A.J. van de, Smit B.: Generating March Tests Automatically, IEEE International Test Conference, IEEE Computer Society, Washington, DC, USA, 1994, pp. 870-878.

[4] Goor A.J. van de, Gaydadjiev G.N., Yarmolik V.N., Mikitjuk V.G.: Memory Tests and their Fault Coverage into a New Perspective, Resulting into a New Test, SEMICON, Seul, Korea, Jan. 1996.

[5] Goor A.J. van de, Al-Ars Z.: Functional Memory Faults: A Formal Notation and a Taxonomy, 18th IEEE VLSI Test Symposium (VTS'00), IEEE Computer Society, Montreal, Canada, 2000, pp. 281-289.

[6] Mrozek I., Yarmolik V.N.: Detection of Pattern Sensitive Faults by Multiple Transparent March Tests, Mixed Design of Integrated Circuits and Systems, proc. 10th International Conference, MIXDES 03, Lodz 26-28 June 2003, pp. 542-545.

[7] Nicolaidis M.: Transparent BIST for RAMs, IEEE International Test Conference, IEEE Computer Society, Baltimore, MD, USA, 1992, pp. 596-607.

[8] Niggemeyer D., Otterstedt J., Redeker M.: Detection of Non classical Memory Faults using Degrees of Freedom in March Testing, Rec. 11th Workshop "Testmethods and Reliability of Circuits and Systems", Potsdam, Feb. 1999.

[9] Sokol B., Mrozek I., Yarmolik V.N.: Transparent March Tests to Effective Pattern Sensitive Faults Detection, Proceedings of the IEEE East-West Design and

Test International Workshop (EWDTW 2004), Crimea, September 23-26, 2004, pp.: 166-171.

[10] Sokol B., Yarmolik V.N.: Memory Faults Detection Techniques with Use of Degrees of Freedom in March Tests, Proceedings of the IEEE East-West Design and Test International Workshop (EWDTW 2005), Odessa, Ukraine, September 15-19, 2005, pp.: 96-101.

[11] Sokol B., Yarmolik S.V.: Address Sequences for March Test to Detect Pattern Sensitive Faults, Proceedings of Third IEEE International Workshop on Electronic Design Test and Applications (DELTA'06), Kuala Lumpur, Malaysia, January 17-19, 2006, pp.:354-357.

[12] Yarmolik V.N., Hellebrand S., Wunderlich H.J.: Symetric transparent BIST for RAMs, Design and Test in Europe DATE'99, Munich 1999.

[13] Yarmolik V.N., Sokol B., Yarmolik S.V.: Counter Sequences for Memory Test Address Generation, Mixed Design of Integrated Circuits and Systems, proc. 12th International Conference MIXDES 2005, Krakow, Poland 22-25 June, 2005, pp.413-418.

[14] Yarmolik S.V., Sokol B.: Optimal Memory Address Seeds for Pattern Sensitive Faults Detection, Proceedings of the IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'2006), Prague, Czech Republic, April 18-21, 2006 - pp.220-221.

[15] Yarmolik S.V., Mrozek I., Sokol B.: Address Sequences Generation for Multiple Run Memory Testing, Proceedings of the 6th Internation Conference Computer Information Systems and Industrial Management Applications (CISIM 07), Elk, Poland, June 28-30, 2007 - pp.341-344.

MEMORY FAULTS DETECTION TECHNIQUES WITH USE OF DEGREES OF FREEDOM IN MARCH TESTS

Abstract: Publication shows the description of selected methods and techniques for memory faults detection with use of degrees of freedom inherent to transparent March tests. This paper deals with Pattern Sensitive Faults (*PSF*) as the most difficult to detect. Proposed techniques of use of degrees of freedom in March testing manifest itself in effective transparent memory testing and *PSF* faults detection with use of simple March tests, based on the possibility of multiple run of test with different initial conditions like address order for each run. Proposed methods allow us to choose in an optimal way starting addresses for multiple March tests run. In the first part of this publication, memory testing problems, testing

principles and degrees of freedom were presented. Second part of this publication shows the description of proposed solutions with selected results.

Keywords: memory testing, march tests, *PSF* faults, degrees of freedom

Artykuł zrealizowano w ramach pracy badawczej W/WI/05/06

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Magdalena Topczewska[1]

RANGOWA KLASYFIKACJA OBIEKTÓW ZA POMOCĄ KUL W RÓŻNYCH NORMACH

Streszczenie: Mając dany zbiór uczący, który zawiera obiekty należące do dwóch lub większej liczby klas, można zbudować najmniejsze kule otaczające obiekty z wybranej klasy rozwiązując zadanie programowania kwadratowego. Ze względu na to, że najmniejsze kule są konstruowane oddzielnie dla każdej klasy, problem może być w prosty sposób rozszerzony do przypadków wieloklasowych. W pracy przedstawiamy propozycje klasyfikatorów opartych na kulach w normie euklidesowej oraz w normie l_1. Przedstawione eksperymenty zostały przeprowadzone tak na syntetycznych jak i rzeczywistych zbiorach danych.

Słowa kluczowe: klasyfikacja, kule, metody nieliniowe

1. Wstęp

Analiza dyskryminacyjna jest jedną z metod eksploracyjnej analizy danych, która służy budowaniu obszarów decyzyjnych pozwalających przydzielać obiekty do odpowiednich klas. W przypadku, gdy nie są znane warunkowe gęstości prawdopodobieństwa w klasach możliwe jest ich szacowanie za pomocą metod nieparametrycznych lub założenie dotyczące ogólnej klasy modelu klasyfikatora. Wybór klasyfikatora wiąże się z decyzją o stopniu złożoności linii decyzyjnej.

Popularne i powszechnie stosowane są klasyfikatory w postaci liniowych funkcji dyskryminacyjnych, dla których znane są uogólnienia za pomocą różnych metod [2,8]. Można również założyć inne kształty funkcji decyzyjnych. W literaturze znane są przykłady zastosowania kul, sześcianów, wielościanów wypukłych lub innych kształtów do podziału przestrzeni.

Pierwszy raz dyskryminacja za pomocą kul zastosował do klasyfikacji obrazów w 1962 roku P.W. Cooper. W kolejnych latach była rozwijana przez wielu naukowców [7]. Znanym algorytmem opartym na kulach jest sieć *RCE* (ang. *Restricted Coulomb Energy*) zaproponowana przez D.L. Reilly, L.N. Coopera i C. Elbauma [4,5]. Każda klasa reprezentowana jest jako zbiór prototypowych obszarów, które

[1] Wydział Informatyki, Politechnika Białostocka, Białystok

mogą być przedstawione w postaci kul. Pojedyncze kule dobierane są w ten sposób, aby zawierały obiekty należące tylko do jednej klasy i nieobjęte przez inne kule. Po zakończeniu budowania obszarów decyzyjnych w postaci zbioru kul nowy obiekt jest klasyfikowany zgodnie z etykietą klasy, jaką miały obiekty zawarte w odpowiedniej kuli.

Innym znanym algorytmem opartym na klasyfikacji za pomocą kul jest *SCM* (ang. *Set Covering Machine*), zaproponowany przez M. Marchanda oraz J. Shawe-Taylora. Głównym celem jest znalezienie najmniejszej liczby kul, które idealnie podzielą przestrzeń na obszary odpowiadające odpowiednim klasom. Klasyfikator wynikowy jest koniunkcją lub dysjunkcją zbioru klasyfikatorów w postaci kul, a każda z nich dzieli przestrzeń wejściową na obszary przynależności do dwóch klas [3].

W cytowanych pracach klasyfikacja za pomocą kul opiera się na takim podziale przestrzeni, by poszczególne kule zawierały jedynie obiekty należące do danej klasy. Jeśli dane z obu klas bardzo mocno zachodzą na siebie, to liczba kul potrzebnych do idealnego podzielenia przestrzeni na obszary decyzyjne może być bardzo duża.

W niniejszej pracy prezentowane są nowe funkcje dyskryminacyjne, które pozwalają na to, by pewna liczba obiektów z jednej klasy znalazła się w obszarze decyzyjnym innej klasy. Funkcje te mają postać kul w różnych normach, więc kształty obszarów decyzyjnych można lepiej dobrać do analizowanych danych.

Najmniejsza kula otaczająca zbiór obiektów z k-tej klasy, scharakteryzowana przez środek $\mathbf{s}$ oraz promień R, może być wyznaczona poprzez rozwiązanie zadania programowania kwadratowego [6]. Modyfikując podejście można uzyskać funkcję celu służącą klasyfikacji obiektów.

W pracy proponujemy klasyfikację punktów należących do kilku klas przy użyciu kryterium nieliniowego opartego na aproksymacji funkcji *signum*, obliczającej znak argumentu, funkcją gładką o własnościach opisanych w dalszej części pracy. Pojedynczy klasyfikator stanowi rozsądny kompromis między ilością prawidłowo i nieprawidłowo sklasyfikowanych przypadków, dopuszczając, aby w obrębie kuli znalazły się także obiekty z innych klas. Takie podejście pozwala zachować jedną z najważniejszych cech klasyfikatorów – *generalizację* przy zachowaniu wysokiej *jakości klasyfikacji*. Wpływa to również na efektywność algorytmu, gdyż zdecydowanie redukuje liczbę kul potrzebnych do podziału przestrzeni wejściowej.

Dodatkowo jako inny typ klasyfikatora proponowane jest zastosowanie kul w normie l_1, dla których optymalne położenie znajdowane jest także przy użyciu kryterium nieliniowego. Zarówno dla kul w normie l_1 jak i l_2, pojedyncze klasyfikatory składane są w warstwy rangowe dzieląc przestrzeń na sekwencję aktywnych pól, pozwalającą na wyjściu uzyskać informację, do której klasy należy przyporządkować dany obiekt.

Struktura pracy jest następująca: w rozdziale drugim opisany jest sposób znajdowania najmniejszej kuli otaczającej obiekty, które należą do jednej klasy. Rozdział trzeci rozszerza ideę najmniejszej kuli otaczającej dane do postaci klasyfikatora, służącego do podziału przestrzeni na obszary decyzyjne dwóch klas. Zaprezentowany jest przykład klasyfikacji oraz wizualizacji przedstawionej funkcji celu. W rozdziale czwartym przedstawiona jest idea klasyfikacji rangowej, czyli strategii wykorzystywanej w sytuacjach, gdy dla prawidłowej klasyfikacji potrzebny jest wielokrotny podział przestrzeni za pomocą wybranego klasyfikatora. Rozdział piąty zawiera opis nowego klasyfikatora mającego postać kuli w normie l_1, a także przykład zastosowania tego typu klasyfikacji. Ostatni, szósty rozdział jest podsumowaniem oraz wytyczeniem dalszych kierunków badań.

2. Najmniejsze kule

Dany jest zbiór obiektów zgromadzonych w zbiorze uczącym składającym się z par $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, gdzie $\mathbf{x}_i$ $(i = 1, \ldots, m)$ jest wektorem zawierającym wartości atrybutów i-tego obiektu, y_i zawiera etykietę klasy, do której należy obiekt $\mathbf{x}_i$. Atrybuty mogą przyjmować wartości ze zbioru liczb rzeczywistych, mogą to być również cechy mające wartości binarne ($\mathbf{x}_i \in R^n$, $\mathbf{x}_i \in \{0,1\}^n$). W przypadku dwóch klas dla etykiety preferowana w pracy jest reprezentacja bipolarna ($y_i \in \{-1,+1\}$), w przypadku wielu klas mogą to być kolejne wartości ze zbioru liczb naturalnych.

Niech k-ta klasa będzie oznaczona jako $C_k = \{\mathbf{x}_i : \ i \in I_k\}$, gdzie I_k jest zbiorem indeksów obiektów należących do tej klasy. Najmniejsza kula otaczająca ten zbiór obiektów, scharakteryzowana przez środek $\mathbf{s}$ oraz promień R, może być wyznaczona poprzez rozwiązanie zadania programowania kwadratowego

$$\min_{\mathbf{s},R} R^2 \tag{1}$$

z ograniczeniami

$$\forall i \quad \|\mathbf{x}_i - \mathbf{s}\|^2 \leqslant R^2, \tag{2}$$

$$R \geqslant 0. \tag{3}$$

gdzie $\|\cdot\|$ oznacza normę euklidesową. Ujmując inaczej $C_k \subseteq B_{\mathbf{s},R}$.

Ze względu na to, że tak sformułowane zadanie jest bardzo wrażliwe na obserwacje odstające, można wprowadzić zmienne rozluźniajce, pozwalając, by część obiektów znalazła się poza kulą. Uzyskuje się wówczas sformułowanie zawierające w funcji celu dodatkowo sumę zmiennych rozluźniających. Poprzez dobór odpowiednich parametrów możemy wpływać na wielkość kuli, a zatem na liczbę obiektów,

które znajdą się poza nią [6]. Ponieważ problem jest wypukły, można zastosować metodę mnożników Lagrange'a i doprowadzić funkcję celu do postaci, w której obliczane są wartości iloczynów skalarnych między wektorami danych, a następnie, w celu znalezienia rozwiązania, zastosować standardowe algorytmy programowania kwadratowego [7].

Należy nadmienić, że rozwiązując (1) z ograniczeniami (2) oraz (3) poszukuje się najmniejszej kuli otaczającej obiekty z wybranej klasy biorąc pod uwagę wyłącznie obiekty z tej klasy. Przypadki należące do innych klas nie są w ogóle rozpatrywane. Jeśli dwie klasy będą się pokrywać może wystąpić sytuacja, że wewnątrz znalezionej kuli znajdą się wszystkie obiekty z obydwu klas. Należy więc tak sformułować zadanie, aby znalezione parametry kuli były tak dobrane, by odpowiednio dużo obiektów z wybranej klasy znalazło się wewnątrz kuli przy jednoczesnym zapewnieniu, że minimalna liczba obiektów z innych klas znajdzie się w jej obrębie. Dlatego w pracy proponowane są modyfikacje (1) pozwalające na dobrą klasyfikację nowych obiektów, dla których informacja, do której klasy powinny zostać przyporządkowane nie jest jeszcze znana.

3. Klasyfikacja oparta na kuli

3.1 Funkcja celu

W tej pracy proponujemy klasyfikację punktów należących do dwóch klas przy użyciu kryterium nieliniowego opartego na aproksymacji funkcji *signum*, obliczającej znak argumentu, dogodną funkcją gładką ς. Modyfikując (1) przy ograniczeniach (2) oraz (3), badanie czy pojedynczy obiekt $\mathbf{x}_i$ znajduje się wewnątrz kuli czy poza nią odbywa się poprzez obliczenie wartości funkcji

$$\phi(\mathbf{x}_i;\mathbf{s},R) = \varsigma(R^2 - (\mathbf{x}_i - \mathbf{s})^2). \tag{4}$$

Funkcja celu przyjmie następującą postać

$$\Phi(\mathbf{x},\mathbf{y};\mathbf{s},R) = R^2 + \frac{1}{4}\sum_{i=1}^{m}(\phi(\mathbf{x}_i;\mathbf{s},R) - y_i)^2. \tag{5}$$

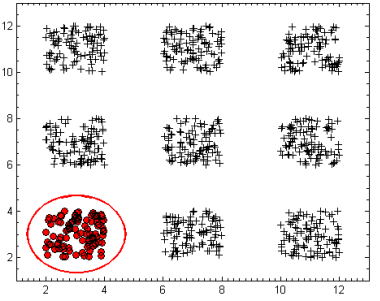
Pierwszy wyraz funkcji celu odpowiada poszukiwaniu najmniejszej kuli wokół zbioru obiektów, drugi natomiast jest regułą minimalizacji błędu kwadratowego przy klasyfikacji obiektów należących do dwóch klas. Jeśli i-ty obiekt posiadający etykietę 1 będzie usytuowany wewnątrz kuli, to wartość funkcji $\phi(\mathbf{x}_i;\mathbf{s},R)$ wyniesie 1, więc wartość różnicy $(\phi(\mathbf{x}_i;\mathbf{s},R) - y_i)$ (błędu klasyfikacji) wyniesie 0. Analogicznie,

jeśli obiekt $\mathbf{x}_i$ o etykiecie -1 będzie usytuowany poza kulą wartość błędu również wyniesie 0. W pozostałych przypadkach wartość różnicy będzie wynosić $+2$ lub -2, co po podniesieniu do kwadratu da wartość 4. Liczbę błędnie sklasyfikowanych obiektów, zarówno tych z etykietą $+1$ leżących poza kulą, jak i obiektów z etykietą -1, usytuowanych wewnątrz kuli, otrzymamy poprzez podzielenie wartości błędów opisanych powyżej przez wartość 4.

Ze względu na ciągłość i różniczkowalność (5) do znalezienia optymalnych wartości $\mathbf{s}$ oraz R można stosować metody gradientowe.

3.2 Przykład z wizualizacją funkcji celu

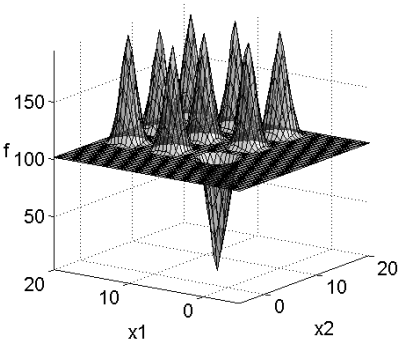
Jako przykład poszukiwania pojedynczej kuli oddzielającej obiekty z wybranej klasy od pozostałych, został wygenerowany zbiór zawierający punkty opisane w R^2, należące do dziewięciu klas. Każda klasa zawiera 100 obiektów. Obiekty jednej klasy można oddzielić od pozostałych stosując klasyfikator w postaci kuli (rys. 1). Zauważmy, że kule w R^2 to koła.



Rys. 1. Zbior – *szach.txt*

Biorąc pod uwagę przykład z przestrzeni dwuwymiarowej, znalezienie optymalnego podziału w pojedynczym kroku algorytmu wiąże się ze znalezieniem trzech parametrów - dwóch współrzędnych opisujących środek kuli oraz jej promienia. Poniżej znajduje się wykres funkcji celu do klasyfikacji skrajnej klasy w przypadku danych *szach.txt* przy ustalonej (optymalnej) wartości promienia.

Jeśli wartość promienia kuli jest ustalona, poszukiwanie minimum funkcji celu (1) ogranicza się do poszukiwania położenia środka kuli. Na rysunku można za-



Rys. 2. Wykres funkcji celu dla zbioru *szach.txt* przy ustalonej (optymalnej) wartości promienia

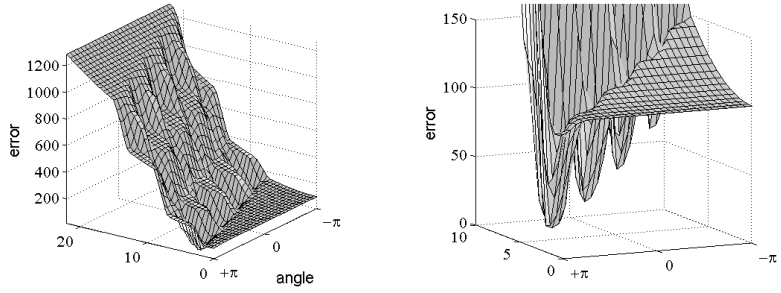
uważyć, że mimimum funkcji znajduje się w środku wyróżnionej klasy. Przesuwanie środka kuli powoduje wzrost wartości funkcji celu, czyli gorszą klasyfikację obiektów. Usytuowanie środka kuli w miejscu, gdzie żaden obiekt nie jest wewnątrz kuli, powoduje, że wartość funkcji celu wynosi około 100 (wszystkie obiekty z wyróżnionej klasy są źle sklasyfikowane). Jeśli zaś kula zawiera inną niż wyróżnioną klasę, to wartość funkcji celu wzrośnie do 200 (zliczone są źle sklasyfikowane obiekty z wyróżnionej klasy oraz obiekty z innych klas, które nieprawidłowo znalazły się wewnątrz kuli).

W celu graficznego przedstawienia funkcji celu z wszystkimi parametrami należy posłużyć się opisem w przestrzeni parametrów. Jako pierwszy parametr wzięty jest kąt między wektorem **s** i osią OX, jako drugi – parametr R.

Zarówno na rysunku 2 jak i 3 funkcja jest gładka i tym samym dogodna do minimalizacji.

4. Klasyfikacja rangowa

Przedstawiona w poprzednim paragrafie funkcja celu dotyczy klasyfikacji obiektów należących do dwóch klas – obiekty z jednej klasy powinny znaleźć się wewnątrz kuli, podczas gdy obiekty z drugiej klasy powinny pozostać na zewnątrz. W przypadku, gdy obiekty należące do wybranej klasy rozmieszone są w przestrzeni danych w kilku rozłącznych klastrach lub gdy w zbiorze danych znajduje się liczba klas większa niż 2, można zastosować wielokrotne podziały zgodnie z wybraną strategią. W pracy proponujemy zastosowanie strategii rangowej oraz wykorzystanie pól aktywnych [1].



Rys. 3. Wykres funkcji celu w przestrzeni parametrów (z prawej strony – powiększenie ukazujące minimum globalne funkcji)

Niech będzie dany zbiór elementarnych klasyfikatorów Q_k, $(i = 1, \ldots, k)$, każdy zdefiniowany na podstawie obiektów $\mathbf{x}_i$, należących do zbioru uczącego, oraz przyjmujący jedną z wartości bipolarnych q_k ($q_k \in \{-1, +1\}$):

$$q_k = q_k(\mathbf{p}, \mathbf{x}_i) \tag{6}$$

gdzie $\mathbf{p}$ jest wektorem parametrów, w tej pracy $\mathbf{p} = [\mathbf{s}_k, R_k]$.

Idea klasyfikacji rangowej polega na wielkokrotnym stosowaniu klasyfikacji danych w celu uzyskania podziału całej przestrzeni na obszary odpowiadające odpowiednim klasom.

Definition 1. *Klasyfikator Q_k jest nadrzędny w stosunku do klasyfikatora Q_r wtedy i tylko wtedy, gdy $k < r$.*

Definition 2. *Pole aktywne P_k jest to zbiór takich obiektów $\mathbf{x}_i$, które uaktywniają klasyfikator Q_k, jednocześnie nie uaktywniając żadnego nadrzędnego klasyfikatora.*

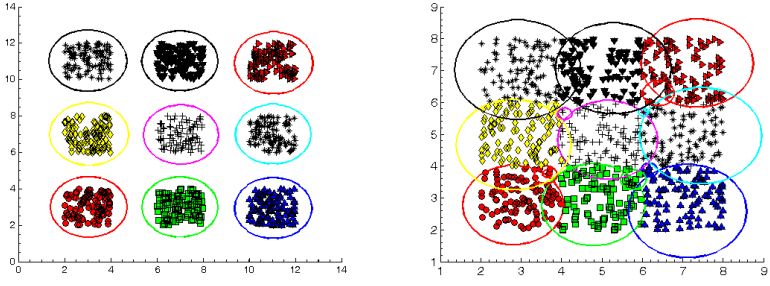
W pierwszym kroku poszukiwana jest kula najlepiej separująca obiekty z wyróżnionej klasy. Wnętrze znalezionej kuli stanowi *pole aktywne*. Zapamiętywana jest dla niego etykieta klasy oraz ranga. Klasyfikator znaleziony w taki sposób stanowi pierwszą warstwę ogólnego rangowego modelu klasyfikacji. Dla warstwy pierwszej ranga jest równa 1. W kolejnym kroku, wśród pozostałych obiektów poszukiwana jest następna kula, która możliwie dobrze oddziela obiekty z wyróżnionej klasy od obiektów z innych klas. Wyróżnioną klasą na tym etapie może być klasa z etapu pierwszego, dla której nie wszystkie obiekty zostały prawidłowo sklasyfikowane lub dowolna inna klasa spośród pozostałych, wybrana według dowolnego kryterium. Algorytm jest powtarzany, a dla znalezionych w kolejnych iteracjach kul zapamiętywana

jest etykieta klasy oraz ranga. Algorytm jest zatrzymany, gdy wszystkie obiekty zostaną podzielone na sekwencję aktywnych pól, pozwalając na wyjściu uzyskać informację, do której klasy należy przyporządkować dany obiekt.

Dzieląc przestrzeń na pola aktywne można posłużyć się ich dwoma rodzajami [1]. Pierwszy rodzaj to pola aktywne, które są deterministycznie dopuszczalne, czyli takie, które zawierają wyłacznie obiekty należące do jednej klasy. Są to pola jednorodne pod względem przynależności obiektów do klas. Drugi typ stanowią pola aktywne, które są statystycznie dopuszczalne. Są to pola, które z pewnym założonym współczynnikiem $\alpha \in (0; 0.5)$ dopuszczają występowanie wewnątrz nich obiektów należących do innych klas. W pracy rozpatrujemy obydwa rodzaje pól aktywnych.

4.1 Przykład

Rozwijając przykład podany w poprzednim paragrafie prezentujemy zbiory danych zawierające obiekty, które należą do dziewięciu klas ułożonych w postaci pól szachownicy. Pierwsza część wykresu przedstawia zbiór, w którym obiekty z różnych klas są separowalne, natomiast druga część dotyczy zbioru, gdy klasy przylegają do siebie.



Rys. 4. Zbiory *szach.txt* z zaznaczonymi granicami decyzyjnymi

Stosując klasyfikację rangową uzyskujemy podział przestrzeni na obszary przynależności do poszczególnych klas. W pierwszym przypadku każda klasa może być odseparowana poprzez jedno cięcie, czyli znalezienie parametrów dla pojedynczej kuli, wewnątrz której znajdą się wszystkie obiekty z odpowiedniej klasy. W drugim – do odseparowania pojedynczej klasy potrzeba jest kilku cięć. W pojedynczym

kroku algorytmu poszukiwana jest najbardziej liczna klasa, której obiekty mogą być sklasyfikowane prawidłowo.

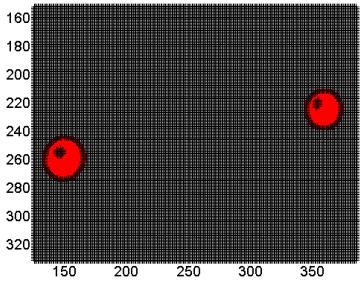
W przypadku danych separowalnych dziewięć iteracji potrzebowano do znalezienia dziewięciu kul całkowicie separujących obiekty, które należały do dziewięciu klas. Każda kula jest jednorodnym polem aktywnym i zawiera obserwacje należące wyłącznie do jednej klasy. W przypadku danych, w których klasy przylegały do siebie algorytm znalazł 22 kule potrzebne do całkowitej separacji obiektów.

4.2 Przykład

Jako przykład drugi został wybrany zbiór zawierający kolorowe zdjęcie, na którym wystąpił efekt "*czerwonych oczu*". Efekt ten może wystąpić, gdy zdjęcie robione jest przy użyciu lampy błyskowej. Światło lampy odbija się od siatkówki dając zabarwienie czerwone na skutek przefiltrowania światła przez naczynia krwionośne.

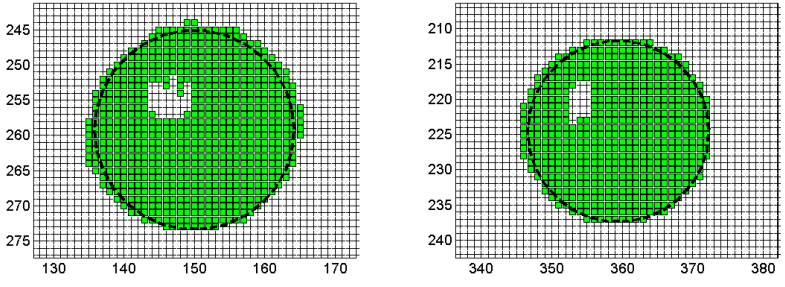
Rys. 5. Obraz oryginalny – Gabi

W tym przypadku jednokrotne zastosowanie klasyfikacji za pomocą kul nie przyniosłoby oczekiwanych rezultatów, gdyż obydwoje oczu powinno się sklasyfikować jako obiekty leżące wewnątrz znalezionych kul. Niezbędna w takim przypadku jest klasyfikacja rangowa. Ze względu na fakt, że oryginalny obraz jest w przestrzeni barw RGB24, poprzez badanie poziomów poszczególnych kanałów została stworzona bipolarna zmienna klasyfikująca, która przyjmuje wartość $+1$ w przypadku gdy kolor piksela jest czerwony oraz -1 dla pozostałych kolorów. Należy tu zaznaczyć, że nie wszystkie obiekty należące do klastra opisującego pojedyncze oko są w odcieniach koloru czerwonego. Odbłysk światła sprawia, że część



Rys. 6. Wyróżniona klasa – powiększenie

pikseli wewnątrz oka ma kolor biały lub zbliżony do białego (rys. 6). Z tego powodu parametry algorytmu powinny być dobrane tak, aby pozwalały umiejscowić wewnątrz kuli obiekty nienależące do wyróżnionej klasy.



Rys. 7. Znalezione kule otaczające obiekty opisujące lewe i prawe oko

W dwóch krokach algorytmu znaleziono optymalne parametry kul otaczających obiekty z danej klasy (rys. 7). Punkty znajdujące się wewnątrz znalezionych kul mogą być następnie poddane obróbce niwelującej efekt czerwonych oczu.

5. Klasyfikacja za pomocą kul w normie l_1

5.1 Funkcja celu

Drugim rodzajem klasyfikatora opisanym w pracy jest kula w normie l_1, opisana równaniem

$$\|\mathbf{x}-\mathbf{s}\|_{l_1} \leqslant R, \tag{7}$$

gdzie $\mathbf{s}$ jest środkiem kuli, R jej promieniem a $\|\cdot\|_{l_1}$ oznacza normę taksówkową.

Analogicznie jak w przypadku normy euklidesowej, najmniejsza kula w normie l_1 otaczająca zbiór obiektów z k-tej klasy, scharakteryzowana przez środek $\mathbf{s}$ oraz promień R, może być wyznaczona poprzez rozwiązanie zadania

$$\min_{\mathbf{s},R} R^2 \tag{8}$$

z ograniczeniami

$$\forall i \quad \|\mathbf{x}_i-\mathbf{s}\|_{l_1} \leqslant R \tag{9}$$

$$R \geqslant 0. \tag{10}$$

W celu znalezienia takiej linii decyzyjnej, by obiekty z etykietą 1 znalazły się wewnątrz kuli, a obiekty z etykietą -1 poza nią, poszukujemy funkcji, która będzie zwracać wartość 1 wewnątrz kuli i wartość -1 poza kulą. Posłużymy się do tego funkcją charakterystyczną, która dla obiektów w kuli zwróci wartość jeden, a poza nią wartość zero. Następnie zmodyfikujemy funkcję, by na wyjściu uzyskiwać wartości ± 1.

W przypadku jednowymiarowym możemy badać usytuowanie obiektu x_i poprzez obliczenie następującej funkcji

$$\phi_{1D}(x_i;s,R) = \frac{1}{2}(\mathrm{sgn}(x_i-s+R)+\mathrm{sgn}(s-x_i+R)). \tag{11}$$

Jeśli obiekt x jest usytuowany poza kulą, wartość jednego wyrazu w sumie będzie dodatnia, drugiego ujemna, więc suma wyniesie zero. Jeżeli natomiast x jest wewnątrz kuli, to wartość sumy wyniesie 2. Mnożąc wynik przez $\frac{1}{2}$ uzyskujemy funkcję, która zwraca wartość 1, gdy obiekt jest usytuowany wewnątrz kuli oraz wartość 0, gdy jest poza nią.

Jeżeli obiekty należą do przestrzeni wielowymiarowej, można wszystkie funkcje (11) dla każdego wymiaru przemnożyć przez siebie. Dodatkowo jako aproksymacja funkcji *sgn* wprowadzona została odpowiednia funkcja gładka ς

$$\phi_{smooth}(\mathbf{x}_i;\mathbf{s},R) = \prod_{j=1}^{n} \frac{1}{2}(\varsigma(x_{ij}-s_j+R)+\varsigma(s_j-x_{ij}+R)). \tag{12}$$

Jeśli obiekt jest umieszczony wewnątrz kuli, to dla każdego wymiaru wartość gładkiej aproksymacji funkcji (11) będzie równa około 1. Przemnożenie przez siebie wszystkich funkcji jednowymiarowych dla każdego wymiaru będzie również wynosić 1. Jeśli natomiast obiekt jest usytuowany poza kulą, wtedy dla wybranych wymiarów funkcja (11) wyniesie 0. Mnożąc wszystkie funkcje jednowymiarowe przez siebie uzyskamy ostatecznie wartość 0.

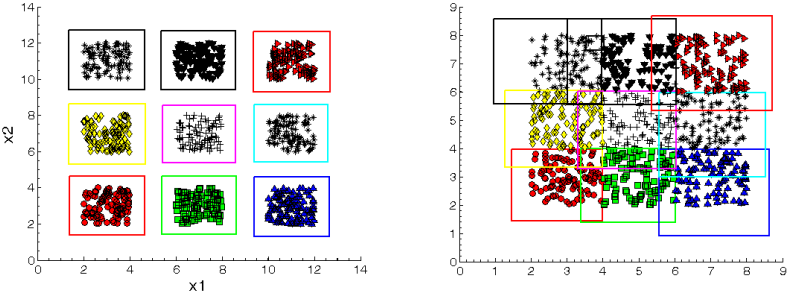
Ze względu na to, że zmienna klasyfikująca w danych ma reprezentację bipolarną, przekształcamy funkcję (12), by na wyjściu otrzymywać wartości 1 dla obiektów znajdujących się wewnątrz kuli oraz -1 dla obiektów rozmieszczonych poza nią

$$\Phi(\mathbf{x},\mathbf{y};\mathbf{s},R) = R_k^2 + \frac{1}{4}\sum_{i=1}^{m}((2\phi_{smooth}(\mathbf{x}_i;\mathbf{s},R)-1)-y_i)^2. \tag{13}$$

Do funkcji celu dodajemy również człon odpowiedzialny za poszukiwanie jak najmniejszej kuli.

5.2 Przykład

Poniżej przedstawiono klasyfikację danych opisanych w *przykładzie 1* przy użyciu kul w normie l_1.



Rys. 8. Zbiory *szach.txt* z zaznaczonymi granicami decyzyjnymi

W przypadku danych separowalnych w dziewięciu kolejnych krokach uzyskano podział przestrzeni na pola aktywne, z których każde było deterministycznie dopuszczalne, czyli zawierało obiekty należące do jednej klasy. Także dane, w których klasy przylegały do siebie, zostały całkowicie odseparowane w dziewięciu cięciach.

6. Podsumowanie

W pracy zaprezentowano klasyfikatory oparte na kulach w normie euklidesowej oraz w normie l_1. Przedstawiono dwie funkcje celu, do minimalizacji których można zastosować metody gradientowe. Niewątpliwą trudność stanowić może zależność metod nieliniowych od wartości wektora początkowego. Można je wyznaczyć stosując metodę k–średnich lub k–medoidów, obliczając oraz zapamiętując wartości średnie lub mediany dla znalezionych klastrów, czyli skupisk obiektów.

Obecnie badany jest przypadek modyfikacji przedstawionych funkcji celu, aby umożliwić niesymetryczne karanie obiektów należących do różnych klas. Eksperymenty będą przeprowadzone na dużych rzeczywistych zbiorach danych.

Literatura

[1] Bobrowski, L.: Ranked Patterns and Structural Linearization of Data Sets, In Poster Proceedings: Petra Perner (Ed.), Industrial Conference on Data Mining, ICDM 2006, IBaI CD-Report, ISSN 1617-2671, July 2006, pp. 30-36.

[2] Duda, O.R., Heart, P.E., Stork, D.G.: Pattern Classification, Wydanie drugie, zmienione, John Wiley & Sons, 2001.

[3] M. Marchand, M., Shawe-Taylor, J.: The set covering machine, Journal of Machine Research, 3, pp. 723-746, 2002.

[4] Reilly, D.L., Cooper, L.N., Elbaum, C.: A neural model for category learning, Biological Cybernetics, 45, pp. 35-41, 1982.

[5] Scofield, C.L., Reilly, D.L., Elbaum, C., Cooper, L.N.: Pattern class degeneracy in an unrestricted storage density memory. In Anderson, D.Z., ed.: Neural Information Processing Systems. American Institute of Physics, Denver, CO, pp. 674-682, 1987.

[6] Tax, D.M.J., Duin, R.P.W.: Support vector domain description, Pattern Recognition Letters, vol. 20, no. 11-13, pp. 1191-1199, 1999.

[7] Wang, J., Neskovic, P., Cooper, L. N.: Pattern classification via single spheres. Lecture Notes in Computer Science: Discovery Science (DS), A. Hoffmann, H. Motoda, and T. Scheffer (Eds.), Springer-Verlag, Vol. 3735. pp. 241-252, 2005.

[8] Johnson, R. A., Wichern, D. W.: Applied Multivariate Statistical Analysis, Prentice-Hall, Inc., Englewood Cliffs, New York, 1991.

RANKED CLASSIFICATION OF DATA USING BOUNDING SPHERES IN DIFFERENT NORMS

Abstract: If a training set containing objects from two or more classes is given, minimum bounding spheres enclosing objects belonging to a marked class can be built by solving a quadratic programming task. Because the minimum spheres are constructed separately for each class the problem can be easily extended to the multi-class cases.
In the paper classifiers both in l_1 and l_2 norms are proposed. Experiments were performed on artificial and on real data sets.

Keywords: classification, spheres, nonlinear methods

Praca naukowa finansowana ze środków na naukę w latach 2007-2008 jako projekt badawczy.

ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Jerzy Wasiluk[1], Aleksander Ostanin[1]

INTERFEJS ŁĄCZNOŚCI EXCEL – MATLAB DO ROZWĄZYWANIA ZAGADNIEŃ FINANSOWYCH

Streszczenie: Artykuł poświęcony jest problemom wymiany danych między *Excelem* a językiem wysokiego poziomu *MATLAB* firmy MathWorks. Opisany został interfejs łączności **IL Excel Link**, który łączy obszary robocze *MS Excela* a *Matlaba* i pozwala użytkownikowi zwracać się do bibliotek dodatkowych *Matlab Toolboxes* przy rozwiązywaniu złożonych zagadnień finansowych, do graficznego interfejsu oraz wywoływać programy w językach C, Java, C++ i in. bezpośrednio ze środowiska *MS Excel*.

Słowa kluczowe: interfejs łączności, wymiana danych, komórki arkusza kalkulacyjnego

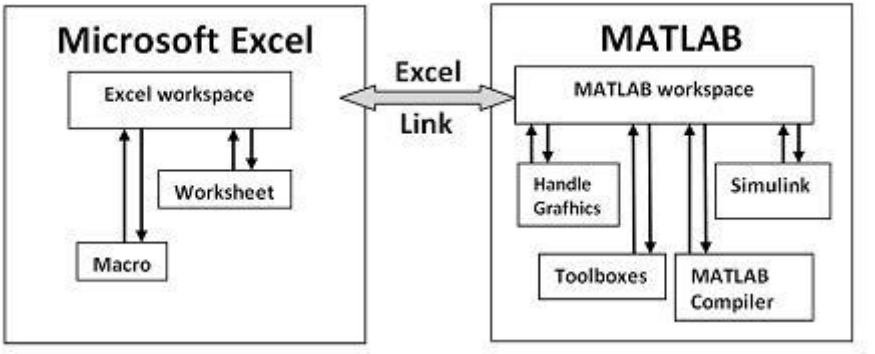
1. Wprowadzenie

Elektroniczne arkusze kalkulacyjne były pierwotnie projektowane z przeznaczeniem dla księgowych, pracowników banków, biznesmenów. Jednak obecnie przy rozwiązywaniu złożonych zagadnień ekonomicznych, takich jak: prognozowanie trendów czasowych (kursów akcji, walut), dynamiczna klasyfikacja (stopień determinacji pewności klienta, atrakcyjność obiektu do inwestycji, rozpoznawanie podpisu), analizy i znajdowanie anomalii w zachowaniu obiektu w czasie itd., możliwości Excela czasami stają się już prawie wyczerpane. W tej sytuacji zachodzi konieczność zwrócenia uwagi na zastosowanie nowych, bardziej zaawansowanych aplikacji matematycznych, takich jak: Mathematica, MatCAD, Maple, Matlab, Sieci Neuronowe itd. Przy tym dla użytkownika niezbędne jest przechowywanie wejściowych danych i wyników obliczeń w tradycyjnej postaci tablic elektronicznych.

Autorzy artykułu stawiają sobie za cel opisanie możliwości używania interfejsu ułatwiającego dostęp do środków programistycznych, np. Matlaba ze strony MS Excela. Taki interfejs łączności **(IL Excel Link)** został opracowany przez firmę Math Works i wchodzi w zestaw narzędzi środowiska MATLAB. Zapełnia on połączenie do wymiany danych między obszarami roboczymi omawianych systemów oraz pozwala realizować polecenia Matlaba nie opuszczając środowiska MS Excel przy rozwiązywaniu złożonych zagadnień ekonomicznych.

[1] Wydział Informatyki, Politechnika Białostocka, Białystok

Na rysunku 1 pokazany jest schemat blokowy łącza MS Excel i środowiska Matlab. Jak wynika z analizy przedstawionego schematu, **IL Excel Link** łączy obszary robocze MS Excela i Matlaba i pozwala użytkownikowi zwracać się do bibliotek dodatkowych Matlaba Toolboxes oraz do graficznego interfejsu bezpośrednio ze środowiska MS Excel.



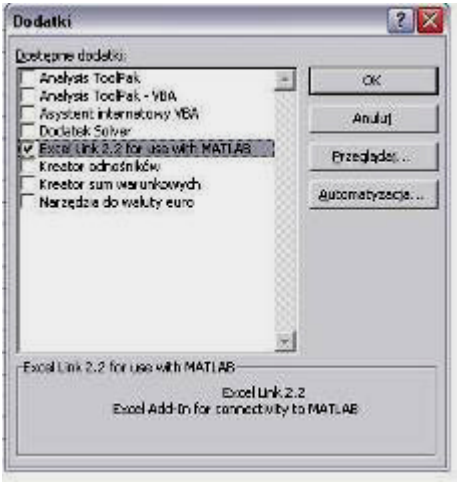
Rys. 1. Schemat blokowy łącza MS Excela z Matlabem

W części drugiej i trzeciej niniejszego artykułu pokazano, że rolę wymiany danych między Excelem a MATLABem spełnia program wspomagający implementację zależności matematycznych o nazwie Excel Link. Zadania zawarte w części czwartej ilustrują zasady pracy między aplikacją MATLAB a MS Excel.

2. Uruchomienie interfejsu łączności (IL)

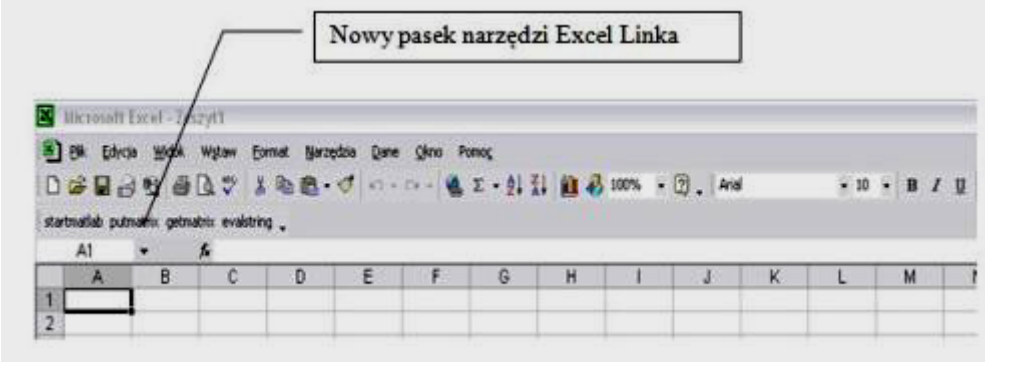
Daną procedurę (rys. 1) przeprowadza się tylko raz:

1. Otworzyć program MS Excel.
2. Wybrać z menu **Narzędzia, Dodatki..., Przeglądaj.**
3. W polu: **Szukaj w** ... - wybrać **MATLAB, TOOLBOX, EXLINK** i dalej drogę do pliku **excllink.xla.**
4. Po wciśnięciu przycisku **OK** powrócimy do okna **Dodatki** (rys. 2), w którym pojawił się nowy dodatek: **"Excel Link 2.2 for use with MATLAB".**
5. Wybrać przycisk **OK**, co zakończy proces.



Rys. 2. Okno dialogowe "Dodatki"

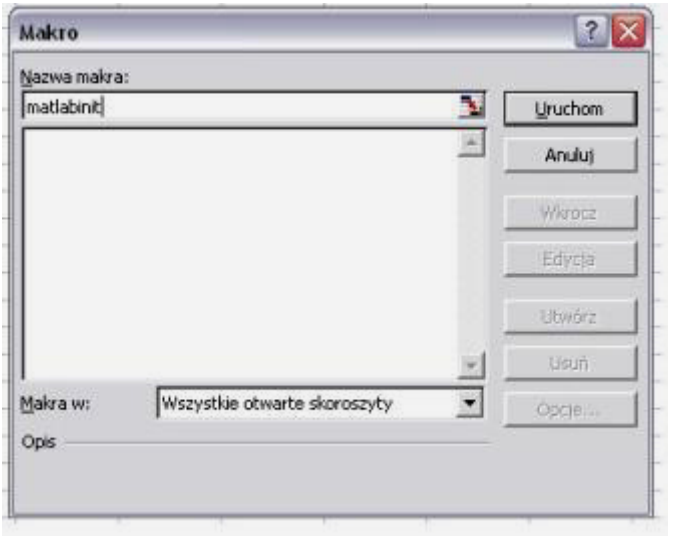
W panelu aktywnych okien MS Windows pojawiło się okno **MATLAB Command Window**, a w oknie MS Excel - nowy pasek narzędzi **Excel Link: startmatlab, putmatrix, getmatrix i evalstring**, który można włączać i wyłączać jak wszystkie inne paski narzędziowe w programach MS Office (rys. 3).



Rys. 3. Nowy pasek narzędzi

Po wykonaniu powyższych kroków, dodatek **Excel Link 2.2 for use with MATLAB** będzie włączał się automatycznie przy każdym wywołaniu MS Excel. Jeśli

tego nie chcemy, należy w dowolną komórkę otwartego, roboczego arkusza wpisać polecenie =MLAutoStart(**"no"**), co wyłączy automatyczny start program MATLAB przy wywołaniu programu MS Excel. Dla ręcznego uruchomienia **IL Excel Link 2.2 for use with MATLAB** należy z menu **Narzędzia-Makro** wybrać podmenu **Makra** - w pole: **Nazwa makra** wpisać **matlabinit** i nacisnąć przycisk: **Uruchom** (rys. 4). W panelu aktywnych okien MS Windows pojawi się okno Matlab Command Window.



Rys. 4. Okno dialogowe "Makro"

Żeby przerwać połączenie programu MATLAB z MS Excel i zakończyć pracę z Matlabem, należy, będąc w Excelu, w dowolną komórkę otwartego arkusza wpisać: =MLClose(), co poinformuje Excela o tym, że praca z Matlabem zostaje zawieszona. Przywrócić połączenie można za pomocą funkcji =MLOpen ().

3. Wymiana danych między Matlabem a MS Excelem

Współpraca z wykorzystaniem **IL Excel Linka** wymaga wykorzystania wszystkich trzech funkcji z paska narzędziowego:**putmatrix** (wstaw do macierzy), **getmatrix** (pobierz z macierzy) i **evalstring** (przelicz). Wykorzystanie tych narzędzi przedstawimy na przykładzie prostego układu równań liniowych z trzema niewiadomymi:

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 14 \\ 4x_1 + 3x_2 - x_3 = 7 \\ x_1 - x_2 + x_3 = 2 \end{cases}$$

co powoduje stworzenie tablicy (tabela 1) z danymi:

Tabela 1. Tablica z danymi

	A	B	C	D	E	F
1	**X_1**	**X_2**	**X_3**	**Wyrazy wolne**	**Wynik**	
2	1	2	3	14	**x_1** =	
3	4	3	-1	7	**x_2** =	
4	1	-1	1	2	**x_3** =	

Problem rozwiązywania układów równań liniowych można wyrazić przy pomocy zapisu macierzowego: mając dane macierze **A** i **B**, należy znaleźć macierz **Y** taką, że **A*Y=B** lub **Y*A=B**. MATLAB dysponuje dwoma operatorami dzielenia macierzowego, używanymi w dwóch wymienionych przypadkach:
- operator $\backslash$ o własności takiej, że **Y=A** spełnia równanie **A*Y=B**,
- operator $/$ o własności takiej, że **Y=A/B** spełnia równanie **Y*A=B**.

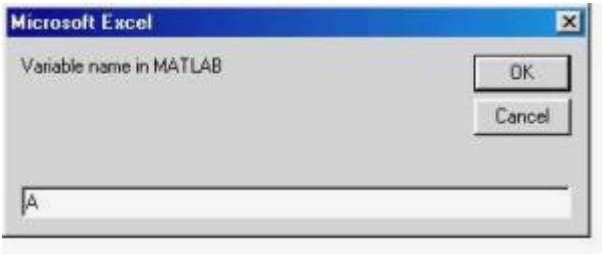
Warunkiem wykonalności dzielenia **Y=A** jest, aby **A** oraz **B** miały równą liczbę wierszy. Liczba kolumn wyniku **Y** jest równa liczbie kolumn **B**, a liczba wierszy liczbie kolumn **A**. W przypadku dzielenia **Y=A/B** wymagana jest zgodność liczby kolumn macierzy **A** i **B**, liczba wierszy macierzy **Y** jest równa liczbie wierszy macierzy **B**, liczba kolumn zaś - liczbie wierszy macierzy **A**. Macierz **A** nie musi być macierzą kwadratową.

Rozwiązanie naszego przykładu rozpatrzymy na podstawie następującego algorytmu.

3.1 Rozwiązywanie zadań liniowych w Matlabie

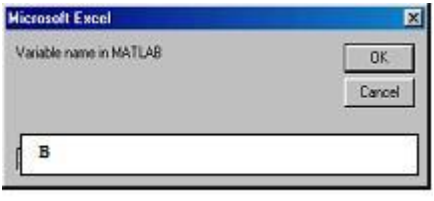
1. Utworzyć macierz **A**.
- utworzymy w Excelu tablicę z danymi, jak to pokazuje tabela 1.
- zaznaczyć na tablicy obszar A3:C5 i wybrać przycisk **putmatrix**. Spowoduje to

otwarcie okna dialogowego, pokazanego na rysunku 5, w którym należy wpisać nazwę dla tworzonej w Matlabie macierzy. W wyniku wykonania funkcji **MLPutMatrix** w obszarze roboczym programu MATLAB pojawiła się macierz klasy *double array* o nazwie A, która zawiera te same dane, co arkusz kalkulacyjny (patrz rys.5).



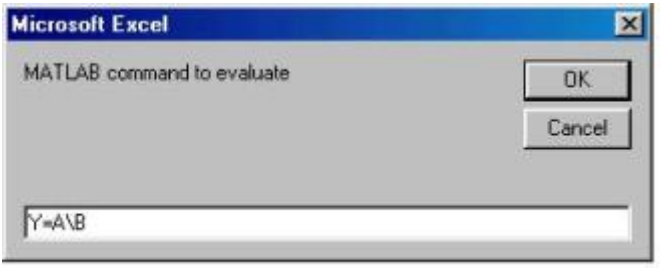
Rys. 5. Okno dialogowe MS Excel dla podania nazwy tworzonej w Matlabie macierzy **A**

2. W sposób analogiczny utworzyć drugą macierz (**B**):
- zaznaczyć obszar wyrazów wolnych równania (D3:D5) i wybrać przycisk **putmatrix**. Spowoduje to otwarcie okna dialogowego, pokazanego na rysunku 6, w którym należy wpisać nazwę dla tworzonej w Matlabie macierzy **B**. W wyniku wykonania funkcji **MLPutMatrix** w obszarze roboczym programu MATLAB pojawiła się macierz klasy *double array* o nazwie B, która zawiera te same dane, co arkusz kalkulacyjny (patrz rys.10).



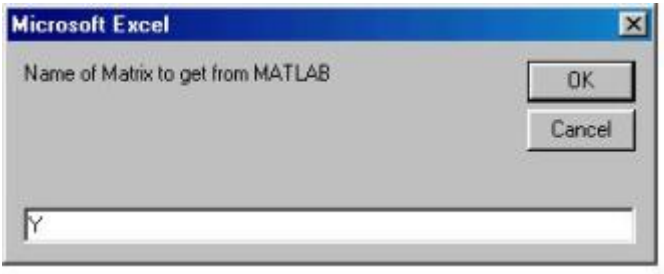
Rys. 6. Okno dialogowe MS Excel dla podania nazwy tworzonej w Matlabie macierzy **B**

3. Wykonać obliczenia, wykorzystując drugi przycisk Excel Linka: **evalstring**. Po jego wybraniu, w otwartym oknie (rys.7) należy wpisać funkcję rachunku macierzowego: **Y= A**, co spowoduje wykonanie obliczeń i utworzenie w Matlabie wektora wyniku.



Rys. 7. Okno dialogowe MS Excel dla wykonania obliczeń w Matlabie

4. Pozostało nam już tylko przesłać wyniki do Excela i rozmieścić je w żądanych komórkach. W tym celu należy zaznaczyć pierwszą komórkę obszaru wyników (F3) i uaktywnić trzeci przycisk Excel Linka, **getmatrix**. Spowoduje to pojawienie się okna jak na rysunku 8, w które należy wpisać nazwę wyniku obliczeń **Y**.



Rys. 8. Okno dialogowe MS Excel dla pobrania wyników obliczeń z Matlabie

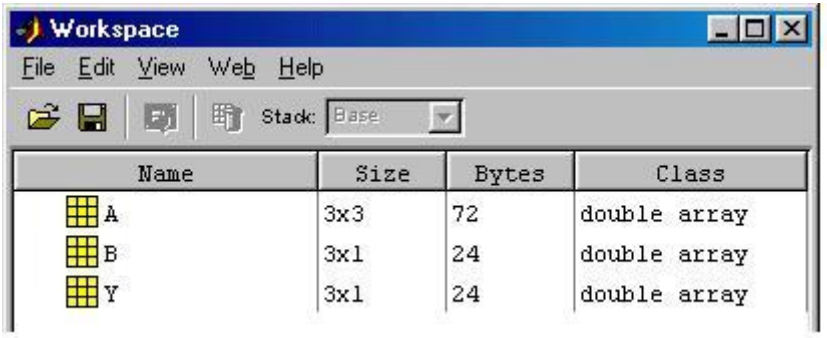
Na skutek wykonania funkcji **MLGetMatrix**, wynik ten zostanie umieszczony w obszarze F3:F5 (rys.9):

Żeby móc obejrzeć utworzone macierze, w programie Matlab Command Windows należy wybrać Workspace Browser, co spowoduje otwarcie okna, pokazanego na rysunku 10. Każdą z utworzonych macierzy można obejrzeć wykonując dwukrotne kliknęiecie na jej nazwie.

Obok tego, jako obiekty podlegające wymianie pomiędzy Matlabem a Excelem, mogą występować dane tekstowe. Rozwiązanie naszego przypadku rozpatrzymy na podstawie następującego przykładu:

	A	B	C	D	E	F
1						
2	**X1**	**X2**	**X3**	**Wyrazy wolne**	**Wynik**	
3	1	2	3	14	**x1 =**	1
4	4	3	-1	7	**x2 =**	2
5	1	-1	1	2	**x3 =**	3

Rys. 9. Tablica z rozwiązaniem pobranym z Matlaba



Rys. 10. Okno dialogowe "Workspace"do wyboru macierzy do obejrzenia w Matlabie

3.2 Wymiana informacji tekstowej

1. Utworzyć macierz **Miesiąc.**
Utworzymy w Excelu tablicę z danymi

Tabela 2. Tablica z danymi tekstowymi

	A	B	C
1	Styczeń	Luty	Marzec
2	Kwiecień	Maj	Czerwiec
3	Lipiec	Sierpień	Wrzesień
4	Październik	Listopad	Grudzień

2. Zaznaczyć na tablicy obszar A1:C4 i wybrać przycisk **putmatrix**. Spowoduje to otwarcie okna dialogowego, pokazanego na rysunku 11, w którym należy wpisać nazwę dla tworzonej w Matlabie macierzy **Miesiąc**. W wyniku wykonania funkcji **MLPutMatrix** w obszarze roboczym programu MATLAB pojawiła się macierz o

nazwie **Miesiąc**, która zawiera dane tekstowe, analogiczne do zawartych w arkuszu kalkulacyjnym.

Rys. 11. Okno dialogowe MS Excel dla podania nazwy tworzonej w Matlabie macierzy **Miesiąc**

3. Wyjaśnimy typ zmiennej Miesiąc, wykorzystując w Matlabie polecenie `whos`:

```
» whos Miesiac
Name            Size    Bytes  Class
Miesiac         4x3     890    cell array

Grand total is 97 elements using 890 bytes
```

Ostatni wiersz oznacza, że całkowita suma jest to 97 elementów, używających 890 bajty.

Okazuje się, ze informacja tekstowa z komórek Excela zapisuje się w plik komórek (`cell array`) Matlaba. Eksport do Matlaba tekstu tylko jednej komórki roboczego arkuszu Excel doprowadzi do rozmieszczenia jej zawartości w obszarze symbolicznym typu `char array`. Import tablicy komórek w Excel doprowadzi do zapełnienia prostokątnej powierzchni na liście roboczym, którego wymiary pokrywają się z rozmiarem obszaru importowanego. Obszar symboliczny importuje się tylko do jednej komórki arkusza Excela.

4. Przykłady rozwązywania zagadnień optymalizacji

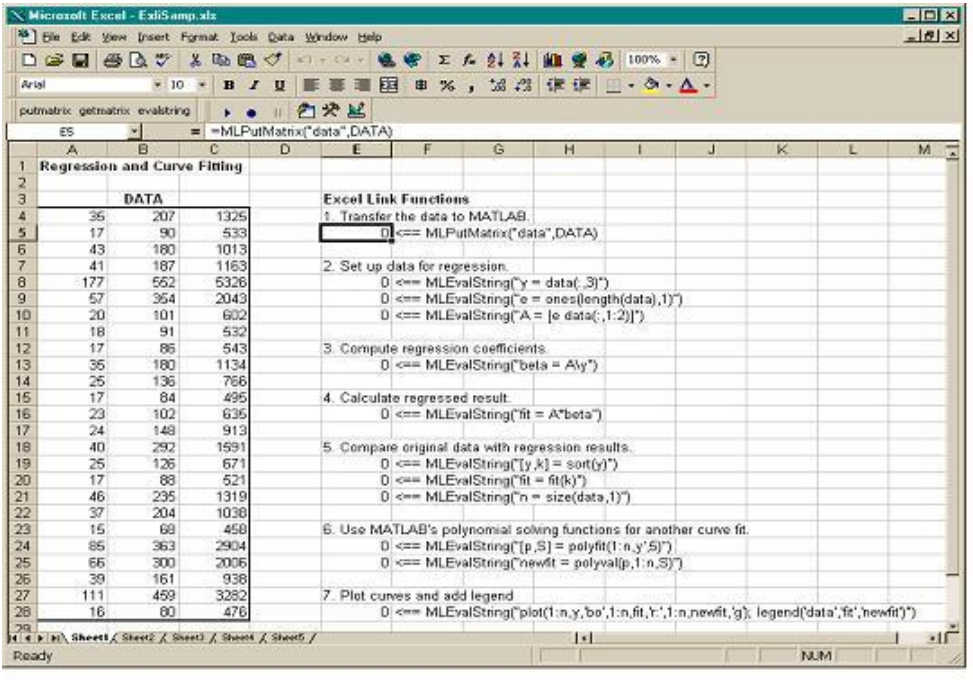
4.1 Regresja i dopasowanie krzywej

Metody analizy regresji i techniki dopasowania krzywych pozwalają znajdować funkcję, która opisuje zależności miedzy zmiennymi. W rzeczywistości, tę funkcję próbują budować matematyczne modele danych z prób. Matlab dostarcza dużo potężnych i łatwych w użyciu macierzowych operatorów i funkcji dla upraszczania realizacji podobnych zadań. Ten przykład realizuje dwie czynności: regresję danych i

dopasowanie ich do zadanej krzywej. Dalej, wykonuje się przykład w wersji arkusza kalkulacyjnego i makrowersji. W przykład używa się arkusza kalkulacyjnego Excela dla graficznego przedstawienia i wywołania danych. Funkcje **Excel Linka** kopiuje dane do Matlaba i wykonuje w jego środowisku operacje obliczeniowe i graficzne. Makrowersja też zwraca dane wyjściowe do arkuszu kalkulacyjnego.

Wersja arkusza kalkulacyjnego

1. Wywołać w **Sheet 1** z przykładu demonstracyjnego **ExliSamp.xls.** Jak widać z rys. 12 arkusz kalkulacyjny zawiera jeden obszar A4:C28 z nazwą DATA (DANE).



Rys. 12. Arkusz kalkulacyjny **Sheet 1** z przykładu demonstracyjnego **ExliSamp.xls**

2. Aktywizować komórką E5, a więc wcisnąć klawisz **F2**, dalej **Enter**. Funkcja Excel Linka kopiuje zestaw danych i przekazuje ich do Matlaba. Zestaw danych zawiera 25 obserwacje trzech zmiennych, które są silnie liniowo zależne; faktycznie, są one skalarnymi wielokrotnościami.

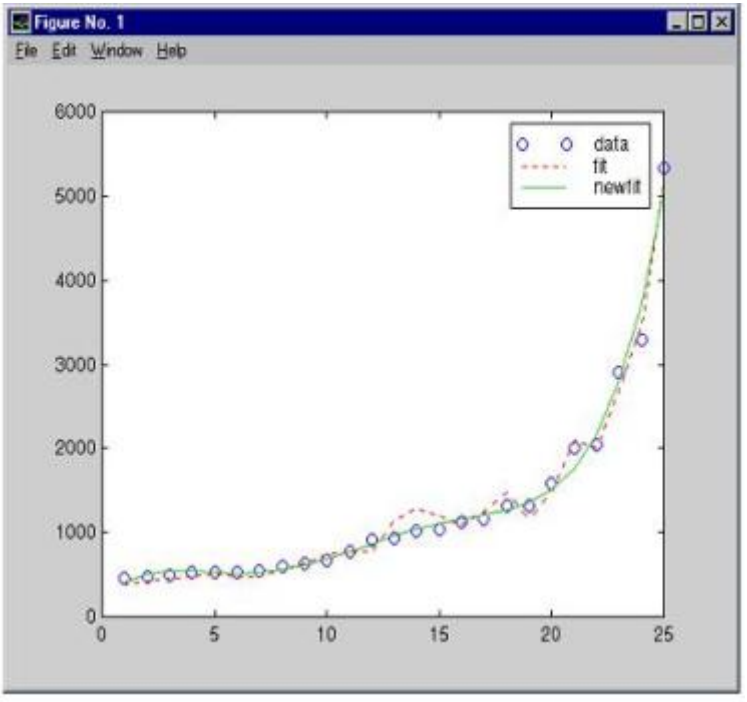
3. Zaznaczyć komórkę E8 i wcisnąć klawisz **F2**, dalej **Enter**. Powtórzyć to samo z komórkami E9 i E10. Funkcja Excel Linka powołuje Matlaba do wykonania analizy regresji trzeciej kolumny danych. Tworzą one pojedynczy wektor y zawierający trzykolumnowe dane i nową trzykolumnową macierz **A**.
4. Wykonuj funkcję w komórce E13. Ta funkcja oblicza współczynniki regresji przez używanie "\ operatora lewostronnego dzielenia macierzy Matlaba, rozwiązując układ równań liniowych, **A* beta = y**.
5. Wykonuj funkcję w komórce E16. Matlab wykonuje macierzowe mnożenie **A** przez wektor współczynników regresji **beta**.
6. Wykonuj funkcje w komórkach E19, E20, i E21. Te funkcje porównują dane oryginalne z dopasowanymi; sortuje dane w porządku rosnącym i stosuje permutację; tworzy skalar dla liczby obserwacji .
7. Wykonuj funkcje w komórkach E24 i E25. Często jest użyteczne dopasowywać współczynniki wielomianu aproksymującego do danych. Zwykle musiałyby ustawiać układ równoczesnych liniowych równań i rozwiązywać go według współczynników. Standardową metodą aproksymacji w Matlabie jest aproksymacja średniokwadratowa wielomianami wybranego stopnia. Polecenie **polifit** automatyzuje to postępowanie i znajduje wektor p współczynników wielomianu, w tym wypadku dla piątego stopnia najlepiej dopasowanego do danych wektorów **x, y**. Funkcja **polyval** ocenia otrzymany wielomian w każdym punkcie danych, żeby kontrolować wynik dopasowania (**newfit**).
8. W końcu wykonuj funkcję w komórce E28. Funkcja Matlaba **plot** wykonuje: wykresy oryginalnych danych (koła), rezultaty regresji **fit** (przerwana linia) i wartości wielomianu (stała linia) oraz dodaje legendę (rys. 13):

Ponieważ dane są dokładnie korelowane, ale niedokładnie liniowo zależne, dopasowana krzywa (linia kreskowa) pokazuje bliskie, ale nieścisłe, przybliżenie. Krzywa wielomianu piątego stopnia **newfit**, przedstawia bardziej dokładny matematyczny model dla danych.

Makrowersja

1. Dla realizacji makrowersji arkusza kalkulacyjnego tego przykładu, należy wywołać **Sheet 2** z przykładu **ExliSamp.xls** (rys.14).
2. Uczynić komórkę A4 aktywną, ale nie naciskać jej jeszcze. Komórka A4 wywołuje makro **CurveFit**, które można badać z otoczenia **Visual Basic**.

3. Podczas gdy ten moduł jest otwarty, wybierz w menu **Narzędzia** i zaznacz **Makro**, dalej **Edytor Visual Basic**. W oknie **Makro** upewnij się czy jest tam pudełko dla



Rys. 13. Wykresy oryginalnych danych (koła), rezultaty regresji **fit** (przerwana linia) i wartości wielomianu aproksymującego (stała linia)

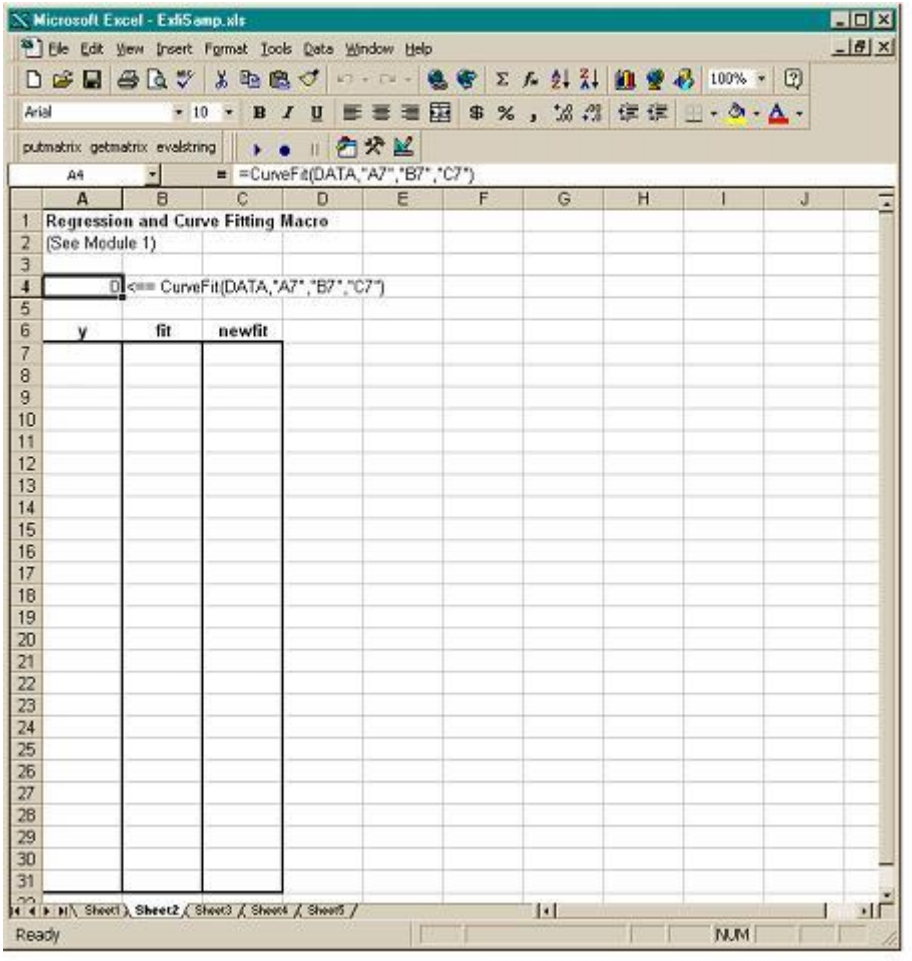
excllink.xla. Jeżeli nie, wprowadź pudełko i nacisk **OK**. Można użyć **Browse** w celu znalezienia pliku **excllink.xla** (rys. 15)

4. Dla wykonania makro **CurveFit** należy wrócić do komórki A4 **Sheet2**, nacisnąć **F2**, dalej **Enter**. Makrowersja wykonuje te same funkcje jak w krokach 1 - 8 wersji arkusza kalkulacyjnego, włączając wykresy (rys. 16). Dodatkowo kopiuje do arkusza kalkulacyjnego oryginalne dane y (sortowane), odpowiednie dane regresji **fit**, i dane wielomianu **newfit** (ostatnie trzy **MLGetMatrix** funkcjonuje w makrokopii danych **CurveFit** arkusza kalkulacyjnego).

4.2 Określenie cen znormalizowanej opcji modeli binominalnych

Sposób interaktywny

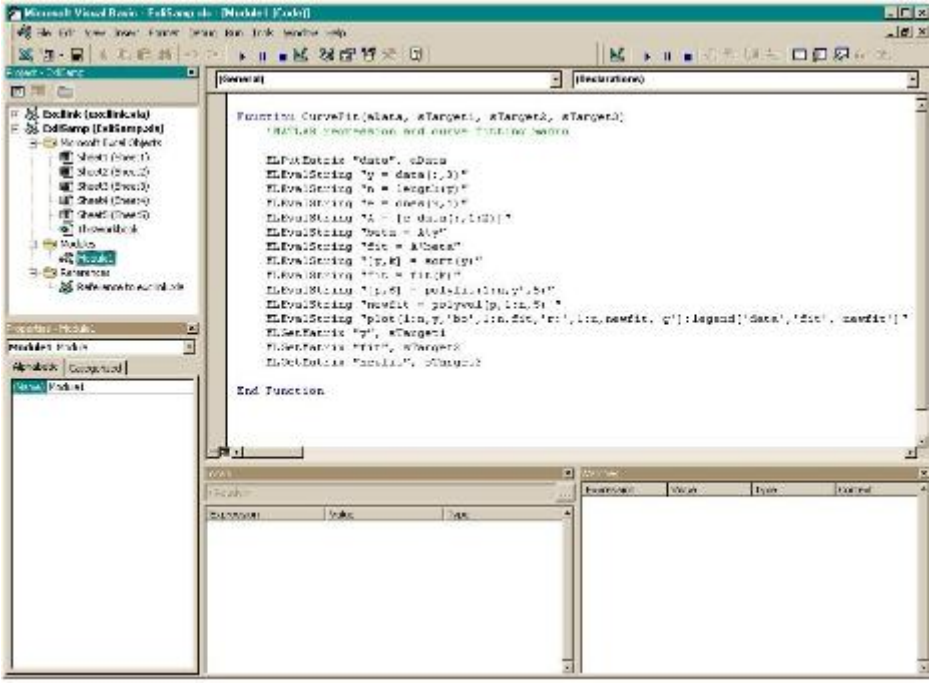
Ten przykład jest związany z zadaniem kształtowania cen opcji i rozwiązuje się na podstawie binominalnego rozkładu wartości opcji. Używanie tego rozkładu



Rys. 14. Arkusz kalkulacyjny **Sheet 2** z przykładu demonstracyjnego **ExliSamp.xls**

uzasadniono tym, że wartość opcji w ciągu określonego krótkiego przedziału czasu może zmieniać się o małe wartości poprzez zwiększenie lub zmniejszenie stron. Procedura sekwencyjnego obliczenia ceny podczas takiej losowej zmiany wiąże się z budową tzw. drzewa binominalnego.

Przykład ilustruje sposób w jaki, znajdując na arkuszu roboczym MS Excel, można wprowadzić, w dialogowym trybie pracy, dane z komórek tablicy elektronicznej do tablicy środowiska Matlab, następnie obliczyć cenę i zwrócić rezultaty w

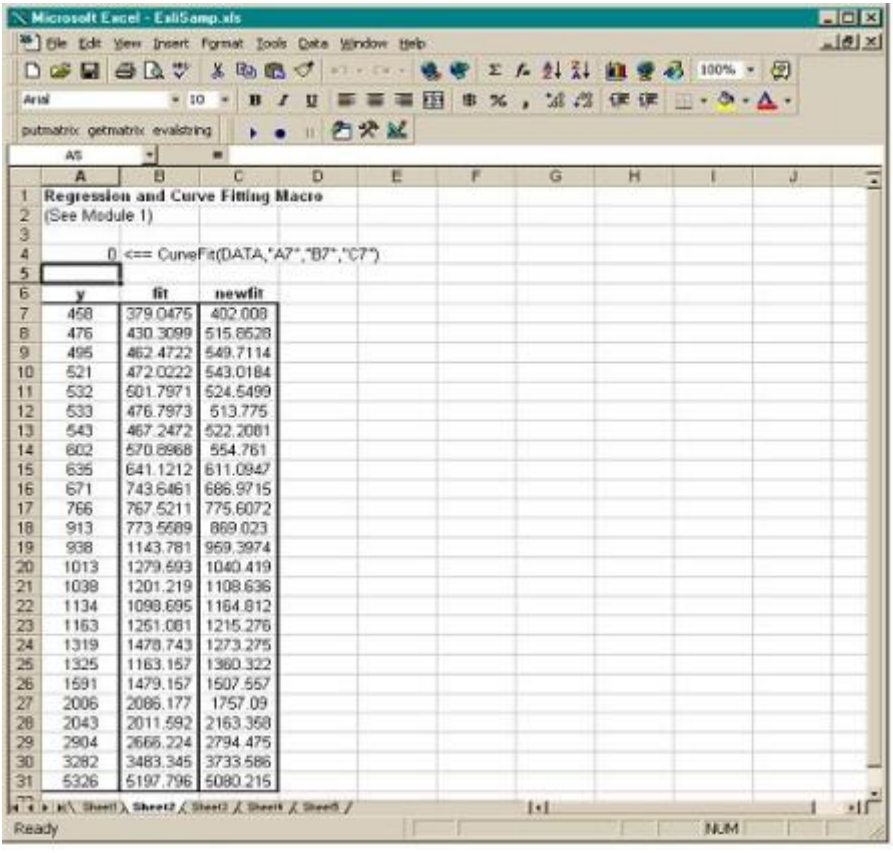


Rys. 15. Makro **CurveFit**, które można badać z otoczenia **Visual Basic**

postaci dwóch tablic w określone komórki MS Excela.

Tryb pracy ma związek z wykorzystaniem tylko trzech funkcji: **putmatrix**, **getmatrix** i **evalstring**, wyświetlanych w postaci przycisków interfejsu **Excel Link** na panelu Ms Excel. Będziemy stosować przykład z arkuszu **Sheet 4** demonstracyjnego pliku **ExliSamp.xls**. Zawartość roboczego arkuszu **Zheet 4** pokazano na rys. poniższym, który zawiera trzy zakresy komórek: B4:B10 z nazwą **bindata**, B15 - **asset_ tree** i B23 - **value_ tree**.

Żeby odczytać dane z komórek B4:B10 i wysłać je do roboczego obszaru Matlaba jako zbiór **b** , należy zaznaczyć te komórki i przycisnąć **putmatrix** na panelu MS Excel. To spowoduje wywołanie okna dialogowego, w którym należy wpisać nazwę zmiennej (**b**) w środowisku Matlab. Po naciskaniu **OK** będzie wykonana funkcja **MLPutMatrix** i w roboczym obszarze Matlaba będzie sformowany zbiór klasy *double array* z nazwą **b**, który zawiera te same dane co komórki arkusza kalkulacyjnego.



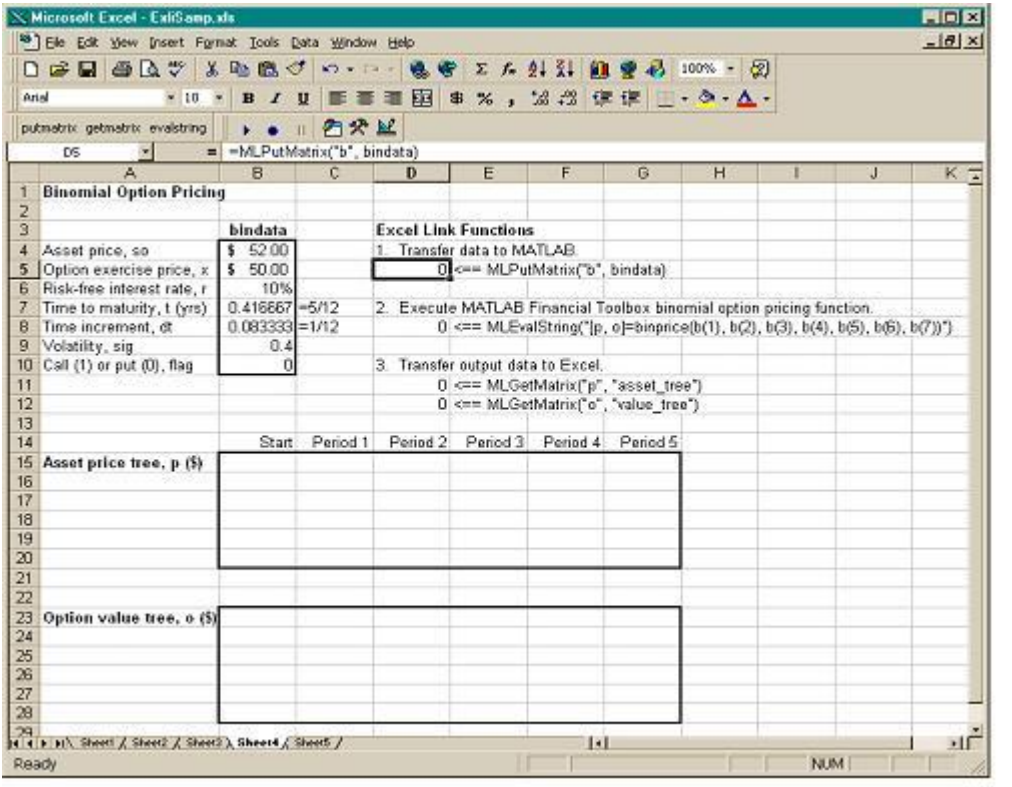
Rys. 16. Makro wersja MS Excel

Dalej obliczamy funkcje **binprice:**

[p,o] = binprice(b(1), b(2), b(3), b(4), b(5), b(6), b(7))

W tym celu należy aktywizować przycisk **evalstring**, co spowoduje pojawienie okna dialogowego, w którym należy wpisać funkcją **binprice** w postaci powyższej oraz nacisnąć **OK**. W rezultacie wykonania funkcji **MLEralString** w obszarze roboczym środowiska Matlab będą sformowane dwie tablice z nazwami **o** i **p**, które zawierają rezultaty obliczeń.

Zostało nam tylko przesłać te dane w MS Excel i rozmieścić je w odpowiednich komórkach. W tym celu należy wyznaczyć komórkę początkową i aktualizować



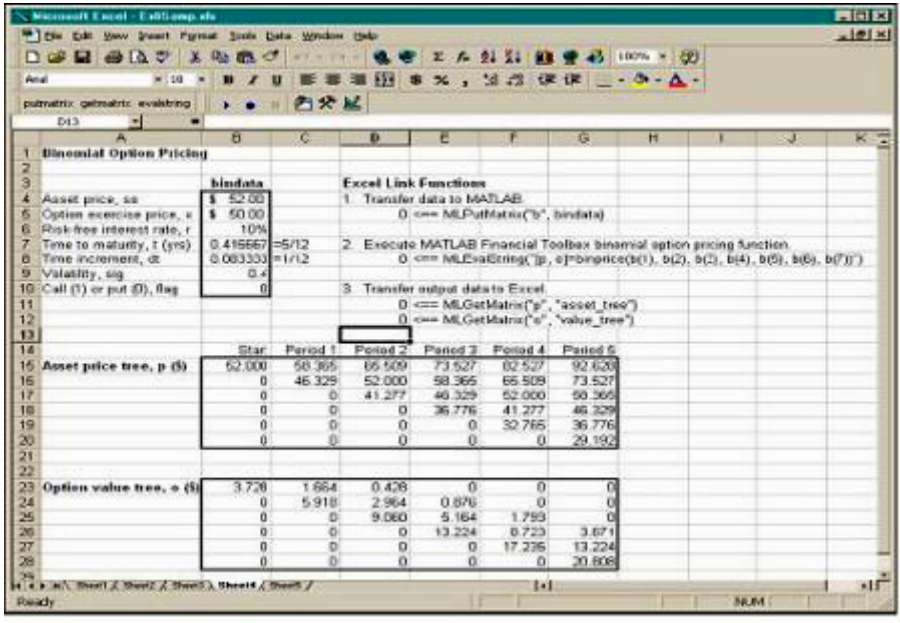
Rys. 17. Zawartość arkuszu kalkulacyjnego **Sheet 4**

przycisk **getmatrix**, co spowoduje pojawienie okna dialogowego, w które należy wpisać nazwę tablicy (**p**). Po wpisaniu nazwy **p**, naciśnięciu **OK** i wykonaniu funkcji **MLGetMatrix** tablica **p** będzie umieszczona w komórkach B15:G20.

Powtarzając analogicznie te same czynności dla tablicy **o**, otrzymamy następującą postać arkusza kalkulacyjnego

Wykorzystanie komórek arkusza kalkulacyjnego

Drugi tryb pracy z funkcjami **IL Excel Link** - to zaniesienie ich do komórek arkusza kalkulacyjnego i zachowanie w celu dalszego wykorzystania. Ten przykład ilustruje w jaki sposób można wykorzystać roboczy arkusz MS Excela dla współ-



	A	B	C	D	E	F	G	H	I	J	K
1	**Binomial Option Pricing**										
2											
3		**bindata**			**Excel Link Functions**						
4	Asset price, ss	$ 52.00			1. Transfer data to MATLAB						
5	Option exercise price, x	$ 50.00			0 <== MLPutMatrix("b", bindata)						
6	Risk-free interest rate, r	10%									
7	Time to maturity, t (yrs)	0.416667	=5/12		2. Execute MATLAB Financial Toolbox binomial option pricing function						
8	Time increment, dt	0.083333	=1/12		0 <== MLEvalString("[p, o]=binprice(b(1), b(2), b(3), b(4), b(5), b(6), b(7))")						
9	Volatility, sig	0.4									
10	Call (1) or put (0), flag	0			3. Transfer output data to Excel						
11					0 <== MLGetMatrix("p", "asset_tree")						
12					0 <== MLGetMatrix("o", "value_tree")						
13											
14		Star	Period 1	Period 2	Period 3	Period 4	Period 5				
15	**Asset price tree, p (b)**	52.000	58.365	65.509	73.527	82.527	92.629				
16		0	46.329	52.000	58.365	65.509	73.527				
17		0	0	41.277	46.329	52.000	58.365				
18		0	0	0	36.776	41.277	46.329				
19		0	0	0	0	32.765	36.776				
20		0	0	0	0	0	29.192				
21											
22											
23	**Option value tree, o (b)**	3.728	1.664	0.428	0	0	0				
24		0	5.918	2.964	0.676	0	0				
25		0	0	9.060	5.164	1.793	0				
26		0	0	0	13.224	8.723	3.671				
27		0	0	0	0	17.236	13.224				
28		0	0	0	0	0	20.808				

Rys. 18. Zawartość sumarycznego arkuszu kalkulacyjnego

pracy z **IL Excel Link**.

Z pliku przykładów demonstracyjnych **elisamp.xls** wybierzemy rozpatrzony już przykład kształtowania cen opcji (arkusz roboczy **Sheet 4**). Na rys. powyżej pokazany jest list, którzy zawiera trzy zakresy: B4:B10 z nazwą **bindata**, B15:G20 - **asset_ tree** i B23:G28 - **value_ tree**. Przy tym dwie komórki w zakresie **bindata** zawierają formuły: komórce B7 odpowiada formuła =5/12, a komórce B8 - formuła **=1/12**. Tutaj, zaczynając od komórki D3, zapisane są funkcji dla pracy z **IL Excel Link**.

Zwróćmy uwagę, że przy odtwarzaniu arkuszu roboczego, zawierającego funkcje **IL Excel Link**, MS Excel próbuje wykonać te funkcje od dołu do góry i z prawa na lewo, co w niektórych wypadkach doprowadzi do pojawienia się informacji o błędach typu **COMMAND!, NONEXIST!** Takie zachowanie MS Excel należy traktować jako normalne oraz ignorować informację podobnego rodzaju.
Następny krok - przejście do komórki D8 w celu realizacji funkcji obliczenia cen według modelu binominalnego. Dalej przechodzimy do komórek D11, D12 w celu

kopiowania danych o cenach w MS Excel.

W rezultacie otrzymamy arkusz kalkulacyjny MS Excel w postaci pokazanej na powyższym rysunku. Można kontynuować eksperymenty z wyżej wymienionym przykładem zmieniając dane wejściowe w B4:B10 i powtarzając funkcję wywołania interfejsu. Należy przy tym zauważyć, że zmiana niektórych parametrów, na przykład terminu dla umorzenia (B7) lub kroku czasowego (B8) może spowodować zwiększenie rozmiaru obszaru wywołania wyników obliczeń.

4.3 Kreślenie wrażliwości opcji portfela w Matlabie

Jest to przykład wykresów w Matlabie wskaźnika gamma jako funkcji ceny i czasu dla portfela opcji 10 ocenianych według modelu Blacka-Scholesa. Wykres pokazuje trójwymiarową powierzchnię. Dla każdego punktu na powierzchni, wzrost (z - wartości) reprezentuje sumę wskaźników gamma dla każdej opcji w portfelu obciążonym pod względem ilości każdej opcji. Oś - x reprezentuje zmienną cenę, a oś - y reprezentuje czas. Czwarty wymiar zobrazowany jest przez pokazanie współczynnika delty jako kolorowej powierzchni.

Na początku zapisujemy portfel z arbitralnymi danymi. Niech aktualne ceny zmienią się z $20 do $90 dla każdej opcji. Zapisujemy odpowiednie ceny wykonywania dla każdej opcji.

```
 Range= 20 : 90;

Plen=  length(Range);

ExPrice= [  75  70  50  55  75  50  40  75  60  35;]
```

Zapisujemy wszystkie wolne - oprocentowania stopy procentowej do 10%, i ustawiamy terminy płatności w dniach. Ustawiamy wartości dla wszystkich zmienności do 0.35 oraz liczbę opcji każdego instrumentu, i wydzielenie miejsca dla macierzy.

```
 Rate= 0.1*ones(10,1);
Time= [  36  36  36  27  18  18  18  9  9  9;]
```

```
Sigma=  0.35*ones(10,1);

NumOpt= 1000*[  4  8  3  5  5.5  2  4.8  3  4.8  2;]

ZVal=  zeros(36, PLen);
Color=  zeros(36, PLen);
```

Dla każdego instrumentu stworzymy macierz (wymiaru time*PLen) cen dla każdego okresu:

```
for i = 1: 10;

Pad =  ones(Time(i),PLen);

V = [Range(ones(Time(i),1),:)]
```

Utworzymy wektor okresów czasowych od 1 do `time`; i macierz czasową, jedna kolumna dla wszystkich cen.

```
T= (1:Time(i))';

NewT =  T(:,ones(PLen,1));
```

Wywołamy toolbox *gammę* i *deltę* wrażliwości funkcji do ich obliczenia.

```
ZVal(36-Time(i)+1:36,:)=  ZVal(36-Time(i)+1:36,:) ...;
+ NumOpt(i) * blsgamma(NewR, ExPrice(i)*Pad, ...
Rate(i)*Pad, NewT/36, Sigma(i)*Pad);
Color(36-Time(i)+1:36,:) = Color(36-Time(i)+1:36,:) ...
+ NumOpt(i) * blsdelta(NewR, ExPrice(i)*Pad, ...|
Rate(i)*Pad, NewT/36, Sigma(i)*Pad);
End
```

Rysujemy powierzchnię jako siatkę, ustawiamy punkt obserwacji, i odwrócimy oś - x względem punktu widzenia. Osie sięgają 20 do 90, 0 do 36, i - dalej.

```
mesh(Range, 1:36, ZVal, Color);
view(60,60);
set(gca, 'xdir','reverse');
axis([20 90 0 36 -inf inf]);
```

Dodajmy tytuł i osie etykiet i narysujmy kwadrat dookoła wykresu ('box', 'on'). Opiszemy za pomocą kolorowych kresek i etykiet (rys. 18)

```
title('Call Option Sensitivity Measures');
xlabel('Stock Price ($)');
ylabel('Time (months)');
zlabel('Gamma');
set(gca, 'box', 'on');
colorbar('horiz');
a = findobj(gcf, 'type', 'axes');
set(get(a(2), 'xlabel'), 'string', 'Delta');
```

5. Podsumowanie

W artykule opisano interfejs łączności **IL Excel Link**, który łączy obszary robocze *MS Excela* a *Matlaba* i pozwala użytkownikowi odwoływać się do bibliotek dodatkowych *Matlab Toolboxes* przy rozwiązywaniu złożonych zagadnień finansowych, do graficznego interfejsu oraz wywoływać programy w innych językach bezpośrednio ze środowiska *MS Excel*. Pokazano na przykładach, w jaki sposób program wspomagający implementację zależności matematycznych o nazwie Excel Link pozwala na współpracę między aplikacją MATLAB a MS Excel.

Literatura

[1] Abdulezer L.: Excel. Praktyczne zastosowania w biznesie, Helion, Gliwice, 2005.

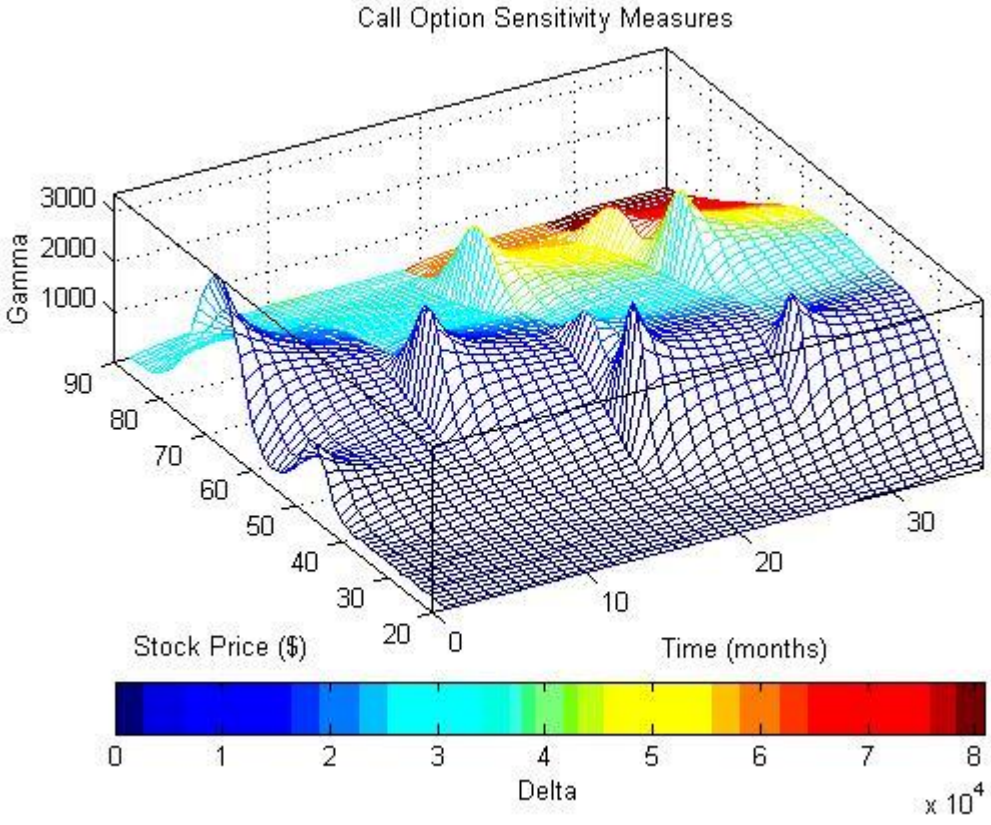
[2] Flanczewski S.: Excel w biurze i nie tylko, Helion, Gliwice, 2003.

[3] Guzik B. (red.): Excel podstawowe zastosowania w ekonometrii., Materiały dydaktyczne, Akademia Ekonomiczna w Poznaniu, Poznań, 1997.

[4] Kolberg M.: Excel w firmie. Przykłady zastosowań, Wydawnictwo Read Me, Łódź, 2001.

[5] Langer M.: Po prostu Excel 2002/XP PL, Helion, Gliwice, 2002.

[6] Liengme B.V.: Microsoft Excel w nauce i technice, Read Me, Warszawa, 2002.



Rys. 19. Wykres wrażliwości opcji portfela

[7] Masłowski K.: Excel 2002/XP PL. Ćwiczenia zaawansowane, Helion, Gliwice, 2002.

[8] Middleton M.R.: Microsoft Excel w analizie danych, Read Me, Warszawa, 2003.

[9] Optimization Toolbox For Use with MATLAB, User's Guide, MathWorks, 2004.

[10] Ostanin A.: Programowanie i modelowanie matematyczne w Excelu, Wydawnictwo Politechniki Białostockiej, Białystok, 2005.

[11] Ostanin A.: Excel podstawowe zastosowania w ekonometrii, Wydawnictwo Wyższej Szkoły Finansów i Zarządzania w Białymstoku, Białystok, 2004.

[12] Penny J., Lindfield G.: Numerical Methods using Matlab, Prentice Hall, N.J., 1999.
[13] Strahl D. i in.: Modelowanie ekonometryczne z Excelem, Wydawnictwo Akademii Ekonomicznej we Wrocławiu, Wrocław, 2004.
[14] USER'S GUIDE Excel LINK for Use with MATLAB, MathWorks, 2004.
[15] Waślicki T.: Excel dla księgowych. Z przykładami praktycznych zastosowań, Wydawnictwo Prawno-Ekonomiczne INFOT, Warszawa, 1997.
[16] Wisniewski M., Dacre T.: Mathematical Programming. Optimization Models for Business and Management Decision Making, McGraw-Hill Book Company, London, 1990.
[17] Zalewski A., Cegieła R.: Matlab - obliczenia numeryczne i ich zastosowania, Nakom, Warszawa, 2002.
[18] http://www.mathworks.com, - adres producenta MATLABa.
[19] http://www.ont.com.pl, - adres dystrybutora MATLABa w Polsce.
[20] http://www.mathworks.com/products/optimization/, - MATLAB Optimization Toolbox.

INTERFACE OF UNITY EXCEL - MATLAB TO SOLVING OF FINANCIAL PROBLEMS

Abstract: The paper deals with the problems of exchange data between *Microsoft Excel* and language of high level *MATLAB*. This paper describes interface of unity *IL Excel Link*, which unites working areas *MS Excel* and *Matlab* and permits for users to turn to additional libraries *Matlab Toolboxes* at solving of complicated financial problems, to graphic interface and to call out programs in languages C, Java, C++ et al. directly from environments *MS Excel*. Connection between application and data-base is given as well.

Keywords: interface of unity, data exchange, cells of worksheet