Michał Bernardelli[1]

Implementacja metody badania własności reguł przejścia systemów bonus-malus w Apache Spark

1. Wstęp

Systemy bonus-malus od wielu lat są wykorzystywane na rynku ubezpieczeń. Mimo ich wad oraz postępu naukowo-technologicznego, za ich dalszym wykorzystaniem w praktyce ubezpieczeniowej przemawiają m.in. czytelność interpretacji oraz stosunkowa łatwość w odbiorze dla przeciętnego klienta. Dostępnych jest wiele pozycji literaturowych poświęconych systemom bonus-malus. Badanie ich własności sprowadza się jednak zazwyczaj do analizy wysokości składek oraz opracowania metod oceny efektywności ich funkcjonowania[2]. Znacznie mniej opracowań poświęconych jest kwestiom optymalizacji reguł przejścia. Jedną z pierwszych prób omówienia tego problemu podjął M. Morlock, który w artykule z 1985 r. próbował znaleźć optymalne reguły dla niemieckiego systemu bonus-malus dla okresu 25 lat[3]. Jego rozwiązanie okazało się jednak nieoptymalne, co po latach wykazali M. Topolewski i M. Bernardelli[4]. Reguły przejścia między klasami systemu bonus-malus wpływają bezpośrednio na efektywność systemu, jednak metody ich analizy są złożone obliczeniowo. Jest to jednym z głównych powodów istnienia niewielu opracowań poświęconych temu zagadnieniu. Jeden z algorytmów aproksymacyjnych został przedstawiony w dwóch artykułach

[1] Szkoła Główna Handlowa w Warszawie, Kolegium Analiz Ekonomicznych.

[2] Por. np.: A. Szymańska, *Wybrane miary efektywności systemów bonus-malus ubezpieczeń komunikacyjnych OC* [w:] *Ubezpieczenia wobec wyzwań XXI wieku*, seria: Prace Naukowe Akademii Ekonomicznej we Wrocławiu, nr 1127, Wrocław 2008, s. 428–435; A. Jędrzychowska, E. Poprawska, *Pomiar efektywności systemu bonus-malus. Analiza wybranych metod oceny*, „Wiadomości Ubezpieczeniowe" 2014, nr 1, s. 75–90.

[3] M. Morlock, *Aspects of optimization in automobile insurance* [w:] K. Neumann, D. Pallaschke (red.), *Contributions to Operations Research*, Proceedings of the Conference on Operations Research Held in Oberwolfach, West Germany February 26 – March 3, 1984, Springer-Verlag, Berlin Heidelberg 1985, s. 131–141.

[4] M. Topolewski, M. Bernardelli, *Optymalizacja reguł przejścia systemu bonus-malus o składkach Q-optymalnych*, „Roczniki Kolegium Analiz Ekonomicznych" 2015, nr 37, s. 229–252.

M. Topolewskiego i M. Bernardellego[5], w których starano się odpowiedzieć na pytanie o wybór optymalnego (lub bliskiego optymalnemu) systemu bonus-malus ze względu na różne kryteria optymalności. Specyfika wykorzystanego algorytmu jest przyczyną braku dowodu optymalności znalezionego rozwiązania, co może stanowić argument przeciwko stosowaniu tego rodzaju przybliżeń.

Celem niniejszego artykułu jest przedstawienie efektywnej obliczeniowo implementacji metody analizy wszystkich możliwych systemów bonus-malus różniących się liczbą klas i maksymalną liczbą wyróżnianych szkód. Implementacja ta wykorzystuje nowoczesne oprogramowanie – Apache Spark – dedykowane do obliczeń na dużych zbiorach danych, dające możliwość wykonywania obliczeń w sposób równoległy bądź rozproszony[6]. Przedstawiona metoda, dzięki uniwersalności, pozwala na wybór najlepszego rozwiązania (lub zadanej liczby najlepszych rozwiązań) względem dowolnie zdefiniowanego kryterium optymalności. Co więcej, ze względu na pełną skalowalność, zapewnianą przez Apache Spark, nie ma praktycznie ograniczeń wielkości badanych systemów bonus-malus. Metoda wydaje się zatem idealnym, a przede wszystkim wiarygodnym źródłem informacji na temat efektywności reguł przejścia danej klasy badanych systemów bonus-malus.

Niniejszy artykuł składa się z pięciu rozdziałów. W drugim rozdziale podane są podstawowe informacje o systemach bonus-malus, ze szczególnym uwzględnieniem systemów dopuszczalnych. Rozdział trzeci poświęcony jest opisowi oprogramowania Apache Spark oraz paradygmatu programowania MapReduce. Zalety tego nowoczesnego oprogramowania, w postaci skalowalności oraz możliwości automatycznego wykonywanie obliczeń w sposób równoległy, zostały wykorzystane do implementacji metody badania własności reguł przejścia systemów bonus-malus. Kluczowe elementy kodu oraz przykład zastosowania metody przedstawiono w rozdziale czwartym. Uniwersalność metody wyraża się m.in. w możliwości określenia dowolnego kryterium optymalności systemu. Artykuł kończy się podsumowaniem w rozdziale piątym.

[5] Ibidem; M. Topolewski, M. Bernardelli, *Improving global elasticity of bonus-malus system*, „Quantitive Methods in Economics" 2017, vol. XVIII, no. 1, s. 120–133.

[6] W zależności od tego, czy oprogramowanie zainstalowane jest na jednej wieloprocesorowej maszynie, czy też na klastrze komputerowym.

2. Dopuszczalny system bonus-malus

Nazwa systemu bonus-malus pochodzi od łacińskich słów *bonus* (dobry) oraz *malus* (zły), które w praktyce ubezpieczeniowej tłumaczone są jako zwyżka/zniżka. Jest to sposób taryfikacji *a posteriori*, który uzależnia wysokość składki ubezpieczeniowej od liczby zgłoszonych szkód w poprzednim okresie rozliczeniowym (zazwyczaj roku). Za J. Lemaire'em przyjmiemy, że na system bonus-malus składają się:

- skończona liczba klas N; zakładamy, że ubezpieczony może należeć tylko do jednej klasy w danym okresie rozliczeniowym,
- składki b_j, przypisane jednoznacznie do każdej z klas j = 1, 2, …, N,
- klasa początkowa, do której trafiają nowi ubezpieczający się[7].

Podstawowym założeniem klasycznego systemu bonus-malus jest zależność przynależności do konkretnej klasy w kolejnym okresie ubezpieczeniowym tylko od klasy i liczby szkód zgłoszonych w poprzednim okresie. Istnieją jednak uogólnienia idei systemu bonus-malus, które uwzględniają więcej niż jeden okres historii ubezpieczenia[8].

W artykule przyjęto, że najlepszą klasą – to jest taką o najkorzystniejszych regułach przejścia oraz o najniższej składce – jest klasa o numerze 1. Klasa o najwyższej składce i najmniej korzystnych regułach przejścia oznaczona jest numerem N. Przyjęto również standardowo wykorzystywane w praktyce uproszczenie, że system uwzględnia co najwyżej q szkód, co oznacza, że zgłoszenie w jednostkowym okresie ubezpieczeniowym więcej niż q szkód traktowane jest tak samo, jak zgłoszenie q szkód.

W artykule wykorzystane zostanie oznaczenie BMS (N, q) dla systemu bonus-malus o N klasach, wyróżniającego do q szkód. Do opisania takiego systemu można wykorzystać macierz przejść T, której element t_{ij} oznacza numer klasy, do której przechodzi ubezpieczony z klasy o numerze i po zgłoszeniu j szkód. Przykładowa macierz przejść dla systemu o siedmiu klasach ($N = 7$), wyróżniającego do trzech szkód ($q = 3$) może mieć postać:

[7] J. Lemaire, *Bonus-Malus Systems in Automobile Insurance*, Kluwer Academic Publishers, Boston 1995.

[8] Patrz: W. Bijak, P. Dziel, *Systemy bonus-malus z wieloletnią historią szkodową*, „Śląski Przegląd Statystyczny" 2017, nr 15(21), s. 7–33.

$$T = \left\{ t_{ij} \right\}_{\substack{i=1,2,\ldots,7 \\ j=0,1,2,3}} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 4 & 5 \\ 2 & 4 & 5 & 5 \\ 3 & 5 & 5 & 6 \\ 4 & 5 & 6 & 7 \\ 5 & 6 & 7 & 7 \\ 6 & 7 & 7 & 7 \end{bmatrix}.$$

Nie każdy z systemów bonus-malus ma uzasadnienie z punktu widzenia sensowności jego stosowania. Dyskusyjne jest np. istnienie systemu, w którym ubezpieczony z większą liczbą szkód trafia do lepszej klasy niż osoba z bezszkodową historią ubezpieczeniową. Wzorując się na M. Podgórskiej i innych[9], w niniejszym artykule rozpatrywane będą pewne warianty dopuszczalnych i sprawiedliwych[10] systemów bonus-malus. Dokładniej, na BMS (N, q) nałożone zostaną następujące warunki ograniczające:

1. Słaba monotoniczność w wierszach macierzy przejść.
   Założenie to oznacza, że w każdej klasie kara za spowodowanie większej liczby szkód jest nie mniejsza niż kara za spowodowanie mniejszej liczby szkód. Formalnie założenie można opisać w postaci warunku:
   $$t_{i,j1} \le t_{i,j2} \text{ dla } 1 \le j1 \le j2 \le q+1,\ i = 1,2,\ldots,N.$$
2. Słaba monotoniczność w kolumnach macierzy przejść.
   Założenie to ma jasną interpretację: w każdej klasie kara za spowodowanie takiej samej liczby szkód w kasie gorszej nie może być mniejsza niż w klasie lepszej (z wyjątkiem klasy najgorszej). Formalnie:
   $$t_{i1,j} \le t_{i2,j} \text{ dla } 1 \le i1 \le i2 \le N,\ j = 1,2,\ldots,q+1.$$
3. Mocna monotoniczność między pierwszymi dwiema kolumnami macierzy przejść.
   Założenie to można zapisać w postaci:
   $$\exists_{1 \le i1 < i2 \le N}\ t_{i1,1} < t_{i2,2},$$
   a ma ono na celu wyeliminowanie systemów, które wcale nie różnicują składek ze względu na liczbę szkód.

[9] M. Podgórska, B. Cieślik, B. Kryszeń, M. Niemiec, M. Topolewski, *System bonus-malus sprawiedliwy w sensie przejść między klasami*, Instytut Ekonometrii SGH, Warszawa 2006.

[10] Różnice pomiędzy dopuszczalnymi a sprawiedliwymi systemami bonus-malus polegają na użyciu ostrych bądź nieostrych nierówności w warunkach ograniczających.

4. Ergodyczność.
   Oznacza to, że modelem systemu jest łańcuch ergodyczny, a tym samym rozkład stacjonarny nie zależy od klasy startowej. Założenie to jest spełnione w przypadku regularnej macierzy prawdopodobieństw przejścia[11].
5. Nieprzywiedlność.
   Modelem systemu jest łańcuch nieprzywiedlny, czyli wszystkie stany łańcucha są wzajemnie osiągalne. Jest to równoważne stwierdzeniu, że rozkład prawdopodobieństwa na przestrzeni stanów w stanie stacjonarnym nie ma zerowych składowych. Założenie to jest spełnione, gdy zapewniona jest prawdziwość następujących warunków:
   - $t_{1,1} = 1,\ t_{N,q+1} = N,$
   - każda klasa $s \in \{1,2,\ldots,N\}$ występuje chociaż raz w macierzy przejść (poza wierszem związanym z daną klasą):

$$\exists_{i,j}\ t_{i,j} = s \text{ dla } i = 1,2,\ldots,s-1,s+1,\ldots,N,\ j = 1,2,\ldots,q+1,$$

   - wszystkie klasy (poza najlepszą) są osiągalne z klas lepszych:

$$t_{i,q+1} \geq i+1 \text{ dla } i = 1,2,\ldots,N-1,$$

   - wszystkie klasy (poza najgorszą) są osiągalne z klas gorszych:

$$t_{i,1} \leq i-1 \text{ dla } i = 2,3,\ldots,N.$$

Liczba wszystkich systemów bonus-malus jest łatwa do wyznaczenia i równa $N^{N(q+1)}$. Liczba systemów bonus-malus, które spełniają warunki ograniczające podane powyżej jest oczywiście znacznie mniejsza, choć – jak zostanie przedstawione w dalszej części artykułu – nadal duża i szybko rosnąca wraz ze wzrostem liczby klas N oraz maksymalnej liczby wyróżnianych szkód q.

3. MapReduce i Apache Spark

MapReduce to zarówno nazwa platformy do równoległego przetwarzania dużych zbiorów danych na klastrach komputerowych, jak również określenie samego modelu programistycznego. Podejście MapReduce pozwala na wykonywanie obliczeń w sposób równoległy i rozproszony. Opracowany przez firmę Google system aktualnie jest dostępny w wersji Open Source stworzonej przez

[11] M. Podgórska, B. Cieślik, B. Kryszeń, M. Niemiec, M. Topolewski, op.cit.

Apache (projekt Apache Hadoop[12]). Idea stojąca za MapReduce jest prosta: podzielić złożony obliczeniowo problem na tyle mniejszych podproblemów, aby każdy z nich dało się rozwiązać na jednej maszynie (jednym procesorze). Im bardziej podatny problem na wykonywanie obliczeń w sposób równoległy, tym większa korzyść ze stosowania tego podejścia. Potencjał stojący za podejściem MapReduce został szczegółowo zaprezentowany przez J.D. Ullmana[13].

Apache Spark[14] uważany jest za nowszą generację podejścia MapReduce. Początkowo (od 2009 r.) rozwijany na uniwersytecie w Kalifornii (Berkeley's AMPLab), w 2013 r. został przekazany przez Apache Software Foundation do bezpłatnego użytku całej społeczności, przyjmując nazwę Apache Spark. W przeciwieństwie do oryginalnego podejścia MapReduce, Spark nie wymaga zapisów na dysk, co w przypadku wielu obliczeń znacznie przyspiesza ich wykonanie. Szybkość obliczeń osiągnięto jednak kosztem zużycia większej ilości pamięci. Tym niemniej, efektywność działania Apache Spark została potwierdzona w wielu przypadkach, m.in. sortowania 100 TB danych (konkurs Daytona Gray-Sort[15]), analizy Big Data związanej z mediami społecznościowymi, historią poczty elektronicznej, logami komputerowymi, rekomendacjami on-line, analizą ryzyka, a także predykcją opartą na modelach ekonometrycznych[16].

W porównaniu z MapReduce, Apache Spark oferuje kilka nowych funkcji, np. *join* lub *sample*, lecz również znacznie bardziej rozbudowane biblioteki dostępne z poziomu wielu języków programowania: Scala, Java, R i Python. Przykładowo biblioteka uczenia maszynowego MLlib oferuje takie przydatne algorytmy, jak regresja, klasyfikacja, klasteryzacja czy optymalizacja.

Istotną kwestią jest też możliwość uruchomienia Apache Spark nawet na maszynie jednoprocesorowej. Z punktu widzenia szybkości obliczeń i możliwości wykonywania ich równolegle, wydaje się to nie mieć dużego sensu. Jednakże większość współczesnych komputerów to maszyny wieloprocesorowe i obliczenia na nich, dzięki Apache Spark, mogą z powodzeniem być wykonywane w sposób równoległy. Ma to szczególne znaczenie dla zadań, których wielkość przekracza dostępną pamięć, co – jak zostanie przedstawione w kolejnym rozdziale – może mieć miejsce w przypadku dużych systemów bonus-malus.

---

12 http://hadoop.apache.org/ (data odczytu: 20.01.2018).

13 J.D. Ullman, *Designing good MapReduce algorithms*, „XRDS: Crossroads, The ACM Magazine for Students (Association for Computing Machinery)" 2012, vol. 19, iss. 1, s. 30–34.

14 https://spark.apache.org/ (data odczytu: 20.01.2018).

15 http://sortbenchmark.org (data odczytu: 20.01.2018).

16 M. Bernardelli, *Econometric modeling of panel data using parallel computing with Apache Spark*, „Roczniki Kolegium Analiz Ekonomicznych" 2016, nr 41, s. 189–202.

4. Opis implementacji metody badania własności systemów bonus-malus w Apache Spark

Zalety Apache Spark można wykorzystać do badania własności reguł przejścia systemów bonus-malus. Ze względu na dużą liczbę możliwości, dostajemy typowy przykład dużego wolumenu danych, do analizy których Apache Spark został stworzony. Warto przy tym rozważyć stosowalność innych podejść rozwiązania problemu optymalizacji względem wskazanego kryterium, polegających na sprawdzeniu wszystkich możliwych systemów. Po pierwsze można zastosować podejście rekurencyjne, które jednakże ma ograniczenia w postaci szybkości działania oraz maksymalnej liczby dozwolonych zagnieżdżonych wywołań rekurencyjnych. Po drugie zastosowanie pętli – mimo iż szybsze od rekurencji – nie pozwala na łatwe odfiltrowanie dopuszczalnych systemów bonus-malus. Wszystkich systemów bonus-malus jest jednak na tyle dużo, że nawet ich wygenerowanie – dla większych wartości parametrów N i q – jest niemożliwe. Zostało to zobrazowane w tabelach 1 i 2.

Tabela 1. Liczba wszystkich systemów bonus-malus w zależności od parametrów N (liczba klas) oraz q (maksymalna liczba wyróżnianych szkód)

q \ N	$q = 1$	$q = 2$	$q = 3$
$N = 4$	65 536	16 777 216	4 294 967 296
$N = 5$	9 765 625	30 517 578 125	95 367 431 640 625
$N = 6$	2,18E+09	1,02E+14	4,74E+18
$N = 7$	6,78E+11	5,59E+17	4,60E+23
$N = 8$	2,81E+14	4,72E+21	7,92E+28
$N = 9$	1,50E+17	5,81E+25	2,25E+34
$N = 10$	1,00E+20	1,00E+30	1,00E+40

Źródło: opracowanie własne

Utożsamiając upraszczająco generowanie jednego systemu bonus-malus z zaledwie jedną operacją zmiennoprzecinkową i zakładając, że obliczenia wykonywane są na najszybszym istniejącym komputerze na świecie[17], pracującym z szybkością niespełna 100 PFLOPS, czyli wykonującym 100 miliardów operacji zmiennoprzecinkowych w ciągu sekundy, otrzymujemy przewidywany

[17] https://www.top500.org (data odczytu: 20.01.2018).

czas niezbędny na obliczenia dla danej klasy systemów bonus-malus[18]. Czas obliczeń rośnie drastycznie wraz ze wzrostem N oraz q. Dla $q = 3$ i $N = 9$ czas obliczeń jest porównywalny z szacowanym wiekiem Wszechświata[19], zaś dla $q = 3$ i $N = 10$ – znacznie go przekracza!

Tabela 2. Przewidywane czasy generowania wszystkich systemów bonus-malus w zależności od parametrów *N* (liczba klas) oraz *q* (maksymalna liczba wyróżnianych szkód)

q \ N	$q = 1$	$q = 2$	$q = 3$
$N = 4$	< sekunda	< sekunda	< sekunda
$N = 5$	< sekunda	< sekunda	< sekunda
$N = 6$	< sekunda	< sekunda	47 sekund
$N = 7$	< sekunda	6 sekund	53 dni
$N = 8$	< sekunda	13 godzin	25 106 lat
$N = 9$	2 sekundy	18 lat	7,14E+09 lat
$N = 10$	17 minut	316 881 lat	3,17E+15 lat

Źródło: opracowanie własne

Wyraźnie widać zatem, że uwzględnienie jakiegokolwiek założenia związanego z dopuszczalnymi systemami bonus-malus, pozwalającego na ograniczenie liczby rozpatrywanych systemów, jest koniecznością. Zauważmy, że spełnienie pierwszego warunku (słaba monotoniczność w wierszach macierzy przejść) oznacza, że każda z kolumn macierzy przejść jest jedną z kombinacji z powtórzeniami ze zbioru $\{1, 2, \ldots, N\}$ ułożoną w sposób alfabetyczny. W języku programowania Python dostępna jest funkcja, która zwraca zbiór takich kombinacji[20]:

$$combinations\_with\_replacement\ (\ \{1, 2, \ldots, N\}, N).$$

Dla $N = 2$ możliwe kombinacje to:

$$(1, 1), (1, 2), (2, 2),$$

[18] Zakładamy też pełną równoległość obliczeń. Każde z założeń w praktyce nie może zostać spełnione, co oznacza, że mówimy o ograniczeniu dolnym czasu obliczeń.

[19] Wiek Wszechświata szacowany jest na 13,8 miliarda lat (według danych z misji Planck z 2013 r.).

[20] W rzeczywistości zwraca iterator, a nie zbiór.

zaś dla $N = 3$:

$$(1, 1, 1), (1, 1, 2), (1, 1, 3), (1, 2, 2), (1, 2, 3),$$
$$(1, 3, 3), (2, 2, 2), (2, 2, 3), (2, 3, 3), (3, 3, 3).$$

Liczba takich kombinacji z powtórzeniami jest równa $\frac{(2N-1)!}{N!(N-1)!}$. Aby stworzyć macierz przejść systemu bonus-malus należy wybrać (z powtórzeniami) $q+1$ kolumn z wyznaczonego zbioru kombinacji. Możliwości takiego wyboru jest $\frac{(\#kolumn+q)!}{(q+1)!(\#kolumn-1)!}$, gdzie *#kolumn* oznacza liczność zbioru kolumn. W ten sposób otrzymać można systemy bonus-malus spełniające pierwsze dwa warunki ograniczające (słaba monotoniczność w kolumnach i w wierszach macierzy przejść). Liczby takich systemów przedstawione zostały w tabeli 3.

Tabela 3. Liczby systemów bonus-malus z macierzami przejść spełniającymi warunek słabej monotoniczności w kolumnach i w wierszach w zależności od parametrów *N* (liczba klas) oraz *q* (maksymalna liczba wyróżnianych szkód)

N	*#kolumn*	Liczba BMS dla $q = 1$	Liczba BMS dla $q = 2$	Liczba BMS dla $q = 3$
$N = 2$	3	6	10	15
$N = 3$	10	55	220	715
$N = 4$	35	630	7 770	73 815
$N = 5$	126	8 001	341 376	11 009 376
$N = 6$	462	106 953	16 542 064	1 923 014 940
$N = 7$	1 716	1 473 186	843 644 516	3,62556E+11
$N = 8$	6 435	20 707 830	44 432 100 570	7,15135E+13
$N = 9$	24 310	295 500 205	2,39473E+12	1,45558E+16
$N = 10$	92 378	4 266 893 631	1,31392E+14	3,03453E+18

Źródło: opracowanie własne

Liczba systemów bonus-malus z tabeli 3 jest nadal duża, ale znacząco mniejsza niż liczba wszystkich systemów z tabeli 1. Oczywiście liczba systemów dopuszczalnych jest jeszcze mniejsza, gdyż w tabeli 3 nie uwzględniono wszystkich warunków ograniczających, a zaledwie dwa pierwsze. Warto też dodać, że dla liczniejszych zbiorów z tabeli 4 nie jest na ogół możliwe przechowanie wszystkich możliwych systemów w pamięci. Rozwiązaniem jest skorzystanie z funkcji oferowanych przez Apache Spark, tj. *filter* oraz *map*.

Stosując notację języka Python, dla systemów bonus-malus wygenerowanych w sposób opisany powyżej wystarczy odfiltrować systemy dopuszczalne według definicji podanej w rozdziale 2:

filter (lambda x: checkMatrix(x)),

gdzie *checkMatrix* oznacza funkcję z implementacją weryfikacji dopuszczalności systemu bonus-malus.

Następnie można zastosować kryterium optymalności, np. Q-optymalność:

map (lambda x: (x, q2calculate(x))),

gdzie *q2calculate* jest implementacją wyznaczającą wartość miary Q2 dla danego systemu bonus-malus[21].

Na końcu zaś można wybrać k najlepszych systemów bonus-malus:

takeOrdered (k, key = lambda x: -x[1])).

Zaprezentowana metoda pozwala na zastosowanie dowolnej funkcji do każdego z dopuszczalnych systemów bonus-malus. Dla bardziej skomplikowanych funkcji czas obliczeń w oczywisty sposób ulegnie wydłużeniu, ale będzie on wprost proporcjonalny do liczby analizowanych systemów. W celu prezentacji możliwości tego podejścia, w tabeli 4 przedstawiono wyniki działania programu dla różnych wartości parametrów *N* (liczba klas) oraz *q* (maksymalna liczba wyróżnianych szkód), przy czym zamiast jednego wybranego kryterium efektywności zliczono wszystkie dopuszczalne systemy bonus-malus danego rozmiaru. Zwróćmy przykładowo uwagę na liczbę systemów BMS (7, 3). Teoretycznie wszystkich możliwych systemów bonus-malus jest aż 4,60E+23, po ograniczeniu do systemów, których macierze przejść spełniają warunki słabej monotoniczności w wierszach i kolumnach otrzymujemy ponad 360 miliardów systemów[22], zaś po zastosowaniu wszystkich warunków ograniczających okazuje się, że dopuszczalnych systemów jest „raptem" nieco ponad 13 miliardów (13 230 375 067).

[21] M. Topolewski, M. Bernardelli, *Optymalizacja reguł przejścia*…

[22] Dokładnie 362 556 230 751 systemów.

Tabela 4. Liczby dopuszczalnych systemów bonus-malus w zależności od parametrów N (liczba klas) oraz q (maksymalna liczba wyróżnianych szkód)

N	Liczba BMS dla $q = 1$	Liczba BMS dla $q = 2$	Liczba BMS dla $q = 3$
$N = 3$	3	29	139
$N = 4$	14	576	9 853
$N = 5$	81	14 290	925 167
$N = 6$	533	411 617	104 066 888
$N = 7$	3 822	13 152 566	13 230 375 067

Źródło: opracowanie własne

Dla uzupełnienia obrazu działania programu w tabeli 5 przedstawione zostały czasy obliczeń związane ze zliczeniem dopuszczalnych systemów bonus-malus. Koniecznie trzeba podkreślić kilka aspektów związanych z tymi obliczeniami. Po pierwsze, dla małych wartości N oraz q czas obliczeń z wykorzystaniem Apache Spark będzie dłuższy niż dla standardowego programu w języku Python. Związane jest to z nakładem czasu niezbędnym do inicjalizacji programu oraz przesłania danych pomiędzy poszczególnymi maszynami. Po drugie, czas programu zależy w dużym stopniu nie tylko od złożoności analizowanego zagadnienia, ale też od mocy obliczeniowej maszyn, którymi dysponujemy. Stąd czasy podane w tabeli 5 należy uznać za przykładowe, mające charakter wyłącznie porównawczy. Wszystkie obliczenia były bowiem wykonywane na tych samych maszynach. Po trzecie wreszcie, Apache Spark jest skalowalny ze względu na wielkość danych wejściowych, czyli dla bardziej złożonych obliczeniowo zadań oferuje pełną funkcjonalność kosztem potencjalnie dłuższego czasu obliczeń. Tymczasem wydajność wielu programów przy większych danych nie tylko drastycznie maleje, lecz nawet uniemożliwia uzyskanie rozwiązania. W przypadku tego artykułu oznacza to, że dla dużych wartości N oraz q obliczenia numeryczne są możliwe do wykonania, ale w odpowiednio dłuższym czasie, proporcjonalnym do wielkości danych wejściowych. Przykładowo dla $N = 7$, $q = 3$ zliczenie ponad 13 miliardów systemów bonus-malus wymagało prawie 35 godzin, mimo wykorzystania stosunkowo wydajnych obliczeniowo maszyn. W praktyce jednak wyniki takich obliczeń nie są potrzebne w czasie rzeczywistym i poświęcenie kilku dni na obliczenia wydaje się rozsądnym kompromisem, jeżeli na drugim końcu szali dostajemy optymalny – względem zastosowanych kryteriów – system ubezpieczeniowy.

Tabela 5. Czasy obliczeń (w sekundach) związanych z wyznaczeniem dopuszczalnych systemów bonus-malus w zależności od parametrów N (liczba klas) oraz q (maksymalna liczba wyróżnianych szkód)

N	Liczba BMS dla $q = 1$	Liczba BMS dla $q = 2$	Liczba BMS dla $q = 3$
$N = 3$	1,97	14,70	244,27
$N = 4$	1,87	17,44	353,11
$N = 5$	1,89	17,57	361,44
$N = 6$	2,10	20,66	958,02
$N = 7$	2,23	246,15	125 677,76

Źródło: opracowanie własne

5. Podsumowanie

Celem artykułu była prezentacja możliwości zastosowania oprogramowania Apache Spark oraz paradygmatu MapReduce do badania własności reguł przejścia dopuszczalnych systemów bonus-malus. Badanie takie jest z założenia zadaniem o dużej złożoności obliczeniowej i w związku z tym dla dużej liczby klas oraz liczby wyróżnianych szkód wymaga zastosowania wyrafinowanego podejścia, wybiegającego poza klasyczne sekwencyjne czy rekurencyjne algorytmy. Na podstawie przedstawionych rozważań oraz przykładowej implementacji można wyciągnąć następujące wnioski:

- liczba możliwych reguł przejścia dla dopuszczalnych systemów bonus-malus jest zadaniem złożonym obliczeniowo, spełniającym kryteria Big Data;
- wykorzystanie Apache Spark do badania własności systemów bonus-malus jest jak najbardziej uzasadnione ze względu na możliwość rozproszenia obliczeń lub wykonywania ich równolegle, jak również zapewnienie pełnej skalowalności problemu ze względu na rozmiar danych wejściowych;
- zaproponowana metoda jest uniwersalna i pozwala na wybór najlepszego rozwiązania względem dowolnie zdefiniowanego kryterium optymalności;
- wynik działania programu opartego na opisanej w artykule implementacji nie pozostawia wątpliwości, jeżeli chodzi o jego optymalność względem przyjętych kryteriów, w przeciwieństwie do zachłannego algorytmu dającego przybliżone rozwiązanie opisanego przez M. Topolewskiego i M. Bernardellego[23]; rozwiązanie oparte na wykorzystaniu Apache Spark dokonuje

[23] M. Topolewski, M. Bernardelli, *Optymalizacja reguł przejścia…*

bowiem wyboru spośród wszystkich możliwych dopuszczalnych systemów bonus-malus;
- wiarygodna możliwość doboru optymalnych reguł przejścia wydaje się przydatna zarówno pod względem rozwoju teorii systemów bonus-malus, jak i praktyki ubezpieczeniowej.

Przedstawione podejście z pewnością warto zestawić z wynikami aproksymacji optymalnego rozwiązania dla różnych kryteriów efektywności systemów bonus-malus, opisanymi w artykułach M. Topolewskiego i M. Bernardellego[24]. W przypadku trafności wskazań przybliżonego algorytmu możliwa jest jego implementacja w Apache Spark, która pozwoli na znaczne przyspieszenie obliczeń. W przypadku, gdy wyniki działania algorytmu aproksymacyjnego okażą się dalekie od optymalnych, możliwe jest szukanie dalszej poprawy efektywności działania metody. Dodatkowo możliwe jest uogólnienie algorytmu na systemy bonus-malus z wieloletnią historią szkodową oraz zbadanie ich własności[25].

Bibliografia

Bernardelli M., *Econometric modeling of panel data using parallel computing with Apache Spark*, „Roczniki Kolegium Analiz Ekonomicznych" 2016, nr 41, s. 189–202.

Bijak W., Dziel P., *Systemy bonus-malus z wieloletnią historią szkodową*, „Śląski Przegląd Statystyczny" 2017, nr 15(21), s. 7–33.

Jędrzychowska A., Poprawska E., *Pomiar efektywności systemu bonus-malus. Analiza wybranych metod oceny*, „Wiadomości Ubezpieczeniowe" 2014, nr 1, s. 75–90.

Lemaire J., *Bonus-Malus Systems in Automobile Insurance*, Kluwer Academic Publishers, Boston 1995.

Morlock M., *Aspects of optimization in automobile insurance* [w:] K. Neumann, D. Pallaschke (red.), *Contributions to Operations Research*, Proceedings of the Conference on Operations Research Held in Oberwolfach, West Germany February 26 – March 3, 1984, Springer-Verlag, Berlin Heidelberg 1985, s. 131–141.

Podgórska M., Cieślik B., Kryszeń B., Niemiec M., Topolewski M., *System bonus-malus sprawiedliwy w sensie przejść między klasami*, Instytut Ekonometrii SGH, Warszawa 2006.

Szymańska A., *Wybrane miary efektywności systemów bonus-malus ubezpieczeń komunikacyjnych OC* [w:] *Ubezpieczenia wobec wyzwań XXI wieku*, seria: Prace Naukowe Akademii Ekonomicznej we Wrocławiu, nr 1127, Wrocław 2008, s. 428–435.

[24] Ibidem; M. Topolewski, M. Bernardelli, *Improving global elasticity*…

[25] W. Bijak, P. Dziel, op.cit.

Topolewski M., Bernardelli M., *Improving global elasticity of bonus-malus system*, „Quantitive Methods in Economics" 2017, vol. XVIII, no. 1, s. 120–133.

Topolewski M., Bernardelli M., *Optymalizacja reguł przejścia systemu bonus-malus o składkach Q-optymalnych*, „Roczniki Kolegium Analiz Ekonomicznych" 2015, nr 37, s. 229–252.

Ullman J.D., *Designing good MapReduce algorithms*, „XRDS: Crossroads, The ACM Magazine for Students (Association for Computing Machinery)" 2012, vol. 19, iss. 1, pp. 30–34.

Źródła internetowe

http://hadoop.apache.org/ (data odczytu: 20.01.2018).

http://sortbenchmark.org (data odczytu: 20.01.2018).

https://spark.apache.org/ (data odczytu: 20.01.2018).

https://www.top500.org (data odczytu: 20.01.2018).

* * *

The method of examining the properties of transition rules for bonus-malus systems using Apache Spark

Abstract

The aim of this article is to present the possibilities of using Apache Spark and the MapReduce paradigm to study the properties of transition rules between classes of the fair bonus-malus system. Such a study is a task of high computational complexity and therefore, for a large number of classes and the number of claims, requires a sophisticated approach that goes beyond the classic sequential or recursive algorithms. The use of Apache Spark, due to the possibility of distributed calculations, full scalability, as well as the susceptibility of the studied issue to parallelization, proved to be an effective and universal – due to the optimization criteria – approach of finding the optimal solution. Due to the verification of all fair bonus-malus systems, the reliability of the results obtained with this method is beyond dispute.

Keywords: bonus-malus system, transition rules, optimization, automobile insurance, Apache Spark