Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

Rok akademicki 2011/2012

Praca dyplomowa magisterska

Jakub Siudziński

Wnioskowanie z wiedzy niepełnej

Opiekun pracy:
prof. nzw. dr hab. inż. Marzena Kryszkiewicz

Ocena: .....

.....
Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

Specjalność: Informatyka –
Inżynieria oprogramowania
i systemy informacyjne

Data urodzenia: 7 lipca 1979 r.

Data rozpoczęcia studiów: 1 października 1999 r.

Życiorys

Urodziłem się 7 lipca 1979 roku w Bydgoszczy. W 1986 r. rozpocząłem naukę w Szkole Podstawowej nr 3 w Bydgoszczy. W 1994 r. zdałem egzamin wstępny do Technikum Elektronicznego im. XXV-lecia LWP w Bydgoszczy, gdzie w 1999 r. złożyłem egzamin dojrzałości i uzyskałem tytuł technika elektronika o specjalności systemy komputerowe. Następnie rozpocząłem studia na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej.

.....
podpis studenta

Egzamin dyplomowy

Złożył egzamin dyplomowy w dniu .....

z wynikiem .....

Ogólny wynik studiów .....

Dodatkowe wnioski i uwagi Komisji .....

.....

.....

Streszczenie

W niniejszej pracy zawarto opis metod wnioskowania z wiedzy niepełnej, w tym odkrywania wzorców ograniczonych, rozszerzonych i skurczonych. Ponadto przedstawiono zwięzłe reprezentacje wiedzy niepełnej. Szczegółowo opisano algorytmy BIS, EIS, SIS i EBIS umożliwiające przeprowadzenie wnioskowania z wiedzy niepełnej, a także algorytmy R i R* służące do odkrywania zwięzłych reprezentacji wzorców. Omówiono szereg algorytmów obliczających wielkości warstw wzorców bez ich wyznaczania. Zaimplementowano i zbadano praktyczne właściwości przedstawionych algorytmów.*

Słowa kluczowe: *wiedza niepełna, zwięzłe reprezentacje wiedzy, liczność warstwy, eksploracja danych*

Mining from partial knowledge

In the thesis, several methods of mining from partial knowledge are described, including discovering of bounded, extended and shrunk itemsets. Moreover, concise representations of partial knowledge are presented. The BIS, EIS, SIS and EBIS algorithms of reasoning about partial knowledge are explained in detail, as well as R and R* algorithms of finding concise representations of patterns. Several algorithms for calculating cardinalities of strata are also presented. The algorithms have been implemented, and their practical properties have been verified.*

Keywords: *partial knowledge, concise representations of knowledge, cardinality of stratum, data mining*

Spis treści

Rozdział 1

Wprowadzenie

1.1 Odkrywanie wiedzy w bazach danych

Odkrywanie wiedzy w bazach danych (ang. *knowledge discovery in databases*) jest obecnie jedną z bardziej dynamicznie rozwijających się dziedzin informatyki. Pomimo swojego młodego wieku dziedzina ta stworzyła wiele technik *eksploracji danych* (ang. *data mining*), które dzięki swojej dużej wydajności i skuteczności znalazły szerokie praktyczne zastosowanie w rozwiązywaniu problemów związanych z szeroko rozumianą analizą danych.

Główną przyczyną szybkiego rozwoju tej dziedziny jest upowszechnienie efektywnych metod pozyskiwania i gromadzenia informacji. Stało się to możliwe za sprawą dużego postępu technologicznego w zakresie pamięci masowych i systemów bazodanowych, a także dzięki upowszechnieniu urządzeń cyfrowych, które umożliwiają automatyczną, bardzo dokładną rejestrację sposobu ich wykorzystania. W świecie konsumenckim można tu wymienić kody paskowe, elektroniczne karty płatnicze czy tzw. urządzenia mobilne, w szczególności telefony komórkowe, smartfony i tablety. Ważnym źródłem danych jest także sieć internet, w której możliwe jest zarejestrowanie praktycznie każdej czynności korzystających z niej użytkowników, którzy ponadto samodzielnie umieszczają liczne informacje o sobie na przykład w portalach społecznościowych. Z tego powodu współczesne zbiory danych osiągają ogromne rozmiary.

Gromadzenie dużych ilości danych nie zawsze jest jednak wystarczające. Często użyteczna wiedza ukryta jest w zależnościach pomiędzy poszczególnymi składowymi tych danych, a zatem do jej odkrycia niezbędne są odpowiednie algorytmy. Właśnie tym zagadnieniom poświęcone są badania z dziedziny eksploracji danych, którą definiuje się jako dziedzinę zajmującą się wydobywaniem nowej, interesującej, użytecznej i nietrywialnej wiedzy

ukrytej w dużych zbiorach danych.

Proces odkrywania wiedzy

Typowy proces odkrywania wiedzy składa się z wielu etapów, wśród których należy wymienić:

- gromadzenie danych - zbieranie danych z różnych źródeł w bazach danych,
- wstępną analizę danych - poznanie natury danych oraz zdefiniowanie celu eksploracji,
- selekcję danych - wybór danych, które zostaną poddane analizie, czyszczenie, weryfikacja spójności i poprawności,
- transformację danych - konwersję typów, wybór strategii wobec danych brakujących, przekształcenie danych do wymaganej postaci,
- eksplorację danych - odkrywanie wiedzy ukrytej w danych za pomocą metod eksploracji danych,
- interpretację wyników - wizualizację, analizę i ocenę otrzymanych wyników, wnioskowanie.

W praktyce etap eksploracji danych może stanowić niewielką część całego procesu odkrywania wiedzy, ponieważ selekcja danych poprzedzona wstępną analizą, odpowiednie ich przygotowanie, a także wybór najbardziej odpowiedniej metody eksploracji danych to etapy, które mają kluczowe znaczenie dla uzyskania satysfakcjonujących rezultatów.

Podstawowe rodzaje odkrywanej wiedzy

Techniki eksploracji danych umożliwiają uzyskanie różnego rodzaju wiedzy reprezentowanej w postaci tzw. *modeli*. Do podstawowych technik eksploracji danych należą:

- klasyfikacja – umożliwia przypisanie obiektom jednej z predefiniowanych klas,
- grupowanie – służy do podziału obiektów na kilka grup w sposób maksymalizujący podobieństwo obiektów w ramach tej samej grupy oraz zróżnicowanie obiektów pomiędzy grupami,

- regresja – analizuje zależności pomiędzy wartościami atrybutów obiektów oraz tworzy model służący do predykcji wartości atrybutów zależnych,

- odkrywanie asocjacji – umożliwia odnalezienie związków między obiektami polegających na ich współwystępowaniu w zbiorach danych.

Wiedza odkrywana w danych przy wykorzystaniu konkretnych technik może przyjmować bardzo różnorodną postać. W szczególności można wyróżnić takie jej reprezentacje jak drzewa decyzyjne, reguły asocjacyjne, sieci neuronowe, formuły logiki predykatów czy automaty skończone.

1.2 Wnioskowanie z wiedzy niepełnej

Zdecydowana większość prac poświęconych zagadnieniom odkrywania wiedzy dotyczy poszukiwania interesujących faktów w bazach danych. Znacznie rzadziej rozpatrywana jest natomiast problematyka wnioskowania na podstawie *wiedzy niepełnej* przy braku dostępu do oryginalnej bazy danych. Przez wiedzę niepełną należy rozumieć pewien zestaw informacji opisujących nieznane dane, a jej źródłem mogą być publicznie dostępne raporty, zestawienia czy różnego rodzaju wyciągi. Klasyczne metody eksploracji danych nie mają w takim przypadku zastosowania. Okazuje się jednak, że wykorzystując jedynie znajomość zależności występujących między poszczególnymi elementami wiedzy, istnieje możliwość przeprowadzenia skutecznego wnioskowania i odkrycia nowych, wartościowych faktów. W dobie ciągłej wymiany informacji, np. między współpracującymi przedsiębiorstwami, brak świadomości zagrożenia, jakim jest wydostanie się tajnych informacji poprzez analizę informacji jawnych, może doprowadzić do realnych strat. I odwrotnie - znajomość technik wnioskowania na podstawie niepełnej wiedzy może zaważyć o zdobyciu przewagi konkurencyjnej.

W niniejszej pracy szczegółowo przedstawiono metody wnioskowania z wiedzy niepełnej reprezentowanej w formie wzorców. Wzorce, oprócz tego, że same w sobie niosą pewną informację o charakterze danych, które opisują, mogą być także wykorzystywane do odkrywania innych rodzajów wiedzy, takich jak reguły asocjacyjne, reguły epizodyczne, wzorce sekwencyjne itd. Skuteczne wnioskowanie na podstawie pewnego danego zbioru wzorców w praktyce może więc okazać się bardzo silnym narzędziem poznawania danych bez konieczności posiadania bezpośredniego do nich dostępu.

1.3 Cel pracy

Celem pracy było opracowanie i implementacja wybranych algorytmów wnioskowania z wiedzy niepełnej, w tym algorytmów odkrywania wzorców ograniczonych, wzorców rozszerzonych i skurczonych, a także algorytmów odkrywających zwięzłą, bezstratną reprezentację tych wzorców. W ramach pracy przedstawiono i zaimplementowano również algorytmy umożliwiające obliczanie liczności warstw wzorców bez konieczności wyznaczania wszystkich wzorców. Do zakresu pracy należało także zbadanie praktycznych własności algorytmów, w szczególności skuteczności odkrywania nowej wiedzy, zwięzłości przedstawionych reprezentacji oraz wydajności algorytmów wyznaczania liczności warstw.

1.4 Układ pracy

Praca ma następujący układ. **Rozdział drugi** jest wprowadzeniem w tematykę odkrywania wzorców częstych w bazach danych. Przedstawia podstawowe pojęcia z zakresu odkrywania wzorców oraz opisuje reguły asocjacyjne jako najbardziej elementarny przykład praktycznego ich wykorzystania. W **rozdziale trzecim** przedstawiono szczegółowo problematykę wnioskowania z wiedzy niepełnej, a także omówiono algorytmy `BIS`, `EIS`, `SIS` i `EBIS*` umożliwiające odkrywanie nowej wiedzy bez dostępu do bazy danych. W **rozdziale czwartym** zaprezentowano zwięzłe reprezentacje wiedzy niepełnej wraz z algorytmami `R` i `R*` umożliwiającymi ich wyznaczanie. **Rozdział piąty** został poświęcony tematyce warstw wzorców, a w szczególności metodom obliczania liczności warstw bez ich wyznaczania. Przedstawiono także algorytmy wyznaczające liczność warstw. W **rozdziale szóstym** zaprezentowano uniwersalne oprogramowanie służące do analizy danych stworzone w ramach pracy. **Rozdział siódmy** zawiera opis przeprowadzonych eksperymentów mających na celu zbadanie skuteczności i wydajności omówionych algorytmów, a także uzyskane wyniki eksperymentalne. W **rozdziale ósmym** zamieszczono podsumowanie pracy. **Dodatek A** zawiera opis wszystkich procedur dostępnych w zrealizowanym oprogramowaniu oraz format wykorzystywanych plików.

Rozdział 2

Odkrywanie asocjacji

2.1 Wprowadzenie

Odkrywanie asocjacji jest jedną z najważniejszych i szeroko stosowaną techniką eksploracji danych, w której wiedza wyprowadzana jest w oparciu o wzorce częste. Wiedza ta reprezentowana jest w postaci modelu *reguł asocjacyjnych* opisujących związki zachodzące pomiędzy poszczególnymi elementami zbioru danych. Swoją dużą popularność technika ta zawdzięcza przede wszystkim efektywnym algorytmom umożliwiającym ekstrakcję wzorców częstych i reguł asocjacyjnych, a także mnogości możliwych praktycznych zastosowań.

Klasycznym przykładem wykorzystania modeli asocjacji jest tzw. *problem koszyka sklepowego*. Polega on na analizie przeprowadzonych transakcji pod kątem wykrywania grup produktów lub usług, które są często kupowane jednocześnie. Wiedza uzyskana w ten sposób może być użyta do umiejętnego rozmieszczania produktów w sklepie lub przygotowywania odpowiednio sprofilowanych kampanii marketingowych.

Przykładowo model asocjacji może być wykorzystany przez operatora sieci GSM do zdefiniowania profili klientów na podstawie używanych przez nich usług dodatkowych. Zaproponowanie klientowi włączenia nowej usługi, która jest chętnie wykorzystywana przez innych klientów o profilu zbliżonym do profilu tego klienta, przyniesie operatorowi potencjalnie większe korzyści niż oferta skorzystania z usługi wybranej przez operatora w sposób czysto losowy. Zastosowanie modelu asocjacji zwiększa prawdopodobieństwo akceptacji oferty przy jednoczesnym zmniejszeniu ryzyka znużenia klienta niedopasowanymi ofertami.

Technika ta stosowana jest także powszechnie w handlu elektronicznym. Obecnie prawie każdy sklep internetowy wykorzystuje asocjacje do prezento-

Robert Cialdini – *Wywieranie wpływu na ludzi. Teoria i praktyka.*
(...)

Klienci, którzy kupili tę książkę, kupili również:
– Artur Schopenhauer – *Erystyka czyli sztuka prowadzenia sporów.*
– Elliot Aronson – *Człowiek. Istota społeczna.*
– Kevin Hogan – *Sztuka porozumienia.*
(...)

Rysunek 2.1: Przykład wykorzystania asocjacji w sklepie internetowym.

wania klientom produktów, które być może ich zainteresują, zarówno online w trakcie wizyty klienta w sklepie, jak i offline poprzez wysyłanie ofert pocztą elektroniczną. Na rysunku 2.1 przedstawiono fragment przykładowej strony WWW z opisem wybranej książki[1]. Zawarta na niej dodatkowa informacja jest cenna zarówno dla klienta, który być może dzięki niej dotrze do nowej, interesującej pozycji, jak i dla sprzedawcy, który ma szansę na zwiększenie sprzedaży[2].

W dalszej części rozdziału zdefiniowane zostaną podstawowe pojęcia związane z odkrywaniem wzorców częstych i reguł asocjacyjnych, wybrane własności oraz statystyczne miary jakości umożliwiające wybór wyłącznie tych reguł, które odzwierciedlają istotne tendencje ukryte w danych.

2.2 Wzorce częste i reguły asocjacyjne

Definicje podstawowe

Zagadnienie odkrywania wzorców częstych i reguł asocjacyjnych po raz pierwszy zdefiniowane zostało w [2] dla baz danych opisujących transakcje sprzedaży (tzw. *transakcyjnych baz danych*), natomiast uogólnienie definicji reguł asocjacyjnych do postaci obecnie wykorzystywanej podano w [3]. Konwencja oznaczeń oraz definicje przedstawione w niniejszym punkcie zostały zaczerpnięte z [18]. Oznaczmy przez $I = \{i_1, i_2, ..., i_m\}$ zbiór wszystkich możliwych *pozycji* (ang. *item*). Przez pozycję rozumiemy pewien podstawowy element bazy danych. W przypadku sklepu odpowiednikiem jednej pozycji będzie oferowany w sprzedaży produkt, zaś I będzie zbiorem wszystkich dostępnych produktów. Przez $T \subseteq I$ oznaczymy zbiór pozycji składający się

[1] Wynik zapytania uzyskany w księgarni internetowej `www.matras.pl`.
[2] Technika ta nosi nazwę *sprzedaży uzupełniającej* (ang. *cross-selling*).

na pojedynczą transakcję. Jest on odpowiednikiem zestawu produktów zakupionych jednocześnie. Transakcyjna baza danych D definiowana jest jako zbiór transakcji T.

Definicja 2.1 Wzorcem *nazywamy dowolny zbiór pozycji X taki, że $X \subseteq I$.*

Zbiory X i T różnią się tym, że T odzwierciedla pewną faktyczną transakcję zapisaną w bazie danych („koszyk zakupów"), zaś X jest dowolnym zbiorem pozycji („hipotetycznym koszykiem zakupów"). Mówimy, że transakcja T *wspiera* zbiór X, jeżeli $X \subseteq T$. Przez *długość* wzorca rozumiemy liczbę należących do niego pozycji. W dalszej części pracy określenia *wzorzec* i *zbiór pozycji* będą używane zamiennie.

Definicja 2.2 Regułą asocjacyjną *nazywamy wyrażenie postaci $X \rightarrow Y$, gdzie $Y \subset I$, $X \subseteq I \setminus Y$ i $Y \neq \emptyset$.*

Zbiór X nazywa się *poprzednikiem*, Y *następnikiem*, zaś $X \cup Y$ *bazą* reguły. Reguła asocjacyjna definiuje zatem zależność polegającą na współwystępowaniu dwóch rozłącznych zbiorów pozycji w bazie danych. Miary umożliwiające ilościową ocenę tej zależności przedstawione są w kolejnym punkcie.

Zbiór wszystkich reguł asocjacyjnych będzie oznaczany przez AR.

Miary jakości wzorców i reguł asocjacyjnych

Istotność i siła zależności opisywanych przez wzorce i reguły asocjacyjne określana jest dzięki prostym miarom statystycznym. Odpowiednio dobrane wartości progowe dla tych miar zapewniają, że uzyskiwana wiedza odzwierciedla ważne zjawiska zachodzące w danych. Miarą statystycznej istotności wzorca X jest jego *wsparcie* (ang. *support*).

Definicja 2.3 Wsparciem *zbioru X nazywamy liczbę transakcji T w bazie danych D, które wspierają zbiór X i oznaczamy przez $sup(X)$:*

$$sup(X) = |\{T \in D | X \subseteq T\}|.$$

W praktyce często wykorzystywana jest także analogiczna definicja, w której powyższa wartość dzielona jest przez liczbę wszystkich transakcji $|D|$. W takim przypadku wsparcie przyjmuje wartości od 0 do 1, dzięki czemu może być interpretowane jako estymacja prawdopodobieństwa tego, że losowo wybrana transakcja T z bazy danych D wspiera zbiór X. Dla sklepowej bazy danych wsparcie zbioru X odpowiada procentowi dokonanych zakupów, które zawierały wszystkie produkty ze zbioru X (oraz być może inne produkty).

Następująca istotna własność wsparcia zbiorów pozycji jest podstawą konstrukcji większości algorytmów operujących na wzorcach:

Własność 2.1 (Podstawowa własność wsparcia)

$$\forall X, Y \in I \colon X \subseteq Y \Rightarrow sup(X) \geqslant sup(Y).$$

Jeżeli zbiór X jest podzbiorem pewnego zbioru Y, to wsparcie $sup(X)$ nie może być mniejsze niż wsparcie $sup(Y)$. Wynika to z tego, że jeśli transakcja T zawiera pozycje ze zbioru Y, to na pewno zawiera także wszystkie pozycje ze zbioru X będącego podzbiorem Y. Przedstawiona własność będzie często wykorzystywana w dalszej części pracy.

Podstawowymi miarami wykorzystywanymi do oceny reguł asocjacyjnych są *wsparcie* i *zaufanie* (ang. *confidence*).

Definicja 2.4 Wsparciem reguły $X \rightarrow Y$ *nazywamy wsparcie zbioru* $X \cup Y$ *stanowiącego bazę tej reguły i oznaczamy przez* $sup(X \rightarrow Y)$*:*

$$sup(X \rightarrow Y) = sup(X \cup Y).$$

Jeżeli wsparcie reguły jest wyrażane jako wartość względna, to odpowiada ono prawdopodobieństwu tego, że reguła ta jest prawdziwa dla losowo wybranej transakcji T (tzn. wszystkie pozycje ze zbioru $X \cup Y$ są zawarte w T).

Definicja 2.5 Zaufaniem reguły $X \rightarrow Y$ *nazywamy stosunek wsparcia tej reguły do wsparcia jej poprzednika i oznaczamy przez* $conf(X \rightarrow Y)$*:*

$$conf(X \rightarrow Y) = \frac{sup(X \rightarrow Y)}{sup(X)}.$$

Zaufanie może być zatem interpretowane jako estymacja prawdopodobieństwa warunkowego tego, że jeżeli poprzednik reguły X należy do wybranej losowo transakcji T, to następnik Y także do niej należy.

Wzorce częste i silne reguły asocjacyjne

Przedstawione podstawowe miary jakości umożliwiają wybranie wyłącznie tych wzorców i reguł, które odzwierciedlają statystycznie istotne zależności zachodzące w danych. Pozwalają także na porządkowanie wzorców i reguł względem wybranego kryterium, dzięki czemu możliwa jest np. odpowiednia prezentacja propozycji książkowych jak w przykładzie z rysunku 2.1. W praktyce ogranicza się liczbę odkrywanych wzorców i reguł poprzez określenie minimalnych wartości wsparcia i zaufania.

Definicja 2.6 Wzorzec $X \subseteq I$ *jest* **częsty**, *jeżeli jego wparcie jest większe niż pewna ustalona wartość* $minSup$. *W przeciwnym przypadku* **wzorzec** X *jest* **rzadki**.

Definicja 2.7 *Regułę* $X \rightarrow Y$ *nazywamy* **silną regułą asocjacyjną**, *jeżeli jej wsparcie jest większe niż ustalona wartość* $minSup$ *oraz zaufanie przekracza ustaloną wartość* $minConf$. *W przeciwnym przypadku* **regułę** *określamy jako* **słabą**.

Z definicji 2.6 i własności 2.1 wynika, że wszystkie podzbiory zbioru częstego są częste, zaś wszystkie nadzbiory zbioru rzadkiego są rzadkie.

Wybór do dalszej analizy wyłącznie wzorców częstych i reguł silnych zapewnia, że opisywane przez nie zależności nie są przypadkowe, a niosą ważną informację o charakterze danych. W przykładzie ze sklepową bazą danych, im wartość wsparcia wzorca $X \cup Y$ jest większa, tym większe jest prawdopodobieństwo tego, że wszystkie produkty ze zbioru $X \cup Y$ są kupowane jednocześnie. Ponadto wysoka wartość zaufania reguły $X \rightarrow Y$ zapewnia, że opisywana przez nią prawidłowość zachodzi z dużym prawdopodobieństwem, tzn. klienci, którzy kupili produkty ze zbioru X, prawdopodobnie zainteresują się także produktami ze zbioru Y.

Zbiór wszystkich wzorców częstych będzie oznaczany przez $\mathcal{F}$:

$$\mathcal{F} = \{X \subseteq I | sup(X) > minSup\}.$$

Zbiór wszystkich silnych reguł asocjacyjnych będzie oznaczany przez $\mathcal{AR}$:

$$\mathcal{AR} = \{r \in AR | sup(r) > minSup \wedge conf(r) > minConf\},$$

gdzie $minSup \in [0, |D|)$, $minConf \in [0, 1)$.

2.3 Odkrywanie wzorców częstych i reguł asocjacyjnych

Klasyczny problem odkrywania reguł asocjacyjnych sprowadza się do wykrycia wszystkich reguł spełniających warunek na minimalną wartość wsparcia i zaufania. Zauważmy, że zgodnie z definicją wsparcia reguły (definicja 2.4), zbiór będący bazą reguły silnej musi być zbiorem częstym. To spostrzeżenie oraz własność wsparć 2.1 umożliwiają sformułowanie następującego ogólnego schematu odkrywania reguł z bazy danych D:

1. Wyznaczenie wszystkich wzorców częstych $\mathcal{F}$ i ich wsparć z uwzględnieniem zadanej wartości $minSup$.

2. Konstrukcja reguł asocjacyjnych na podstawie każdego wzorca $Z \in \mathcal{F}$ zgodnie z zależnością: $X \rightarrow Z \setminus X$, gdzie $X \subset Z$ oraz odrzucenie tych reguł, których wartość zaufania nie przekracza zadanej wartości $minConf$.

Własność 2.1 zapewnia, że każdy zbiór X będący podzbiorem zbioru częstego Z jest także zbiorem częstym, a więc należy do $\mathcal{F}$. Dzięki temu w drugim kroku powyższego schematu wartość jego wsparcia jest znana i może być wykorzystana do obliczenia wartości zaufania badanej reguły.

Głównym problemem obliczeniowym, związanym z koniecznością bezpośredniego dostępu do dużych baz danych i wyznaczaniem wsparć zbiorów pozycji, jest pierwszy krok powyższego schematu. Konstrukcja reguł na podstawie wykrytych wzorców częstych jest zadaniem stosunkowo prostym. Badania nad optymalizacją odkrywania reguł asocjacyjnych skupiają się więc przede wszystkim na usprawnieniu procesu wykrywania zbiorów częstych. Podstawowym algorytmem służącym wyznaczaniu zbiorów częstych jest algorytm przeszukiwania wszerz Apriori opisany w [3]. Analogiczne rozwiązanie przedstawiono równolegle w [24]. Duża część późniejszych prac badawczych poświęconych odkrywaniu wzorców i reguł asocjacyjnych bazuje na tym algorytmie, np. [25], [32], [33], [9], [6]. Alternatywną strategię przeszukiwania w głąb zaoferowano m.in. w [10] i [1].

Rozdział 3

Wnioskowanie z wiedzy niepełnej

3.1 Wprowadzenie

Jak wspomniano we wstępie do niniejszej pracy, eksploracja danych jest najczęściej pojmowana jako dziedzina zajmująca się wydobywaniem interesującej wiedzy ukrytej w dużych zbiorach danych. Istnieją jednak sytuacje, w których bezpośredni dostęp do danych nie jest możliwy, a znany jest tylko częściowy ich opis w postaci wyciągów, statystyk czy różnego rodzaju zestawień. W takim przypadku typowe metody wydobywania wiedzy nie mają zastosowania. Nadal istnieje jednak możliwość przeanalizowania dostępnych informacji i przeprowadzenia wnioskowania. Okazuje się, że na podstawie jedynie fragmentarycznego opisu danych, nazywanego dalej *wiedzą niepełną*, możliwe jest przeprowadzenie efektywnego wnioskowania o charakterze tych danych. Proces ten, w analogii do eksploracji danych, można by określić *eksploracją wiedzy*.

W czasach, w których dostęp do informacji oraz umiejętne jej wykorzystanie stanowią o sile i przewadze konkurencyjnej, znajomość metod wnioskowania na podstawie wiedzy niepełnej może być kluczowa dla bezpieczeństwa i stabilności przedsiębiorstw. Przykładowo współpracujące ze sobą firmy zazwyczaj muszą dzielić się pewnymi informacjami niezbędnymi do realizacji wspólnych celów biznesowych. Część przedsiębiorstw jest także statutowo zobowiązana do publikowania danych na temat swojej działalności, inne zaś robią to dobrowolnie. W każdym przypadku konieczne jest zapewnienie, że przekazywana w ten sposób wiedza nie umożliwia wyprowadzenia innych, być może poufnych i strategicznych faktów na temat firmy. Efektywne wykorzystanie technik wnioskowania z wiedzy niepełnej w celu

poznania *faktycznie* przekazywanej wiedzy może okazać się w takiej sytuacji bardzo pomocne.

Istnieje cały szereg innych zastosowań omawianych dalej metod wnioskowania. Brak dostępu do danych źródłowych może wynikać z ich utraty w wyniku awarii systemów informatycznych, z wewnętrznej polityki bezpieczeństwa dającej możliwość przeglądania pełnych danych wyłącznie wybranym osobom, czy z charakteru informacji, która ma być analizowana (np. przetwarzanie ogólnodostępnych zestawień i raportów w celu wykrycia nieprawidłowości lub nadużyć). Jeżeli tylko posiadana fragmentaryczna wiedza na temat określonego zjawiska jest w jakimś stopniu skorelowana, to istnieje szansa na odkrycie informacji, także nietrywialnych i nowatorskich, ukrytych w jej wewnętrznych zależnościach.

Badania nad wnioskowaniem z wiedzy niepełnej zapoczątkowane zostały w [16]. Przedstawiony tam mechanizm umożliwiał wyprowadzenie nowych reguł asocjacyjnych na podstawie pewnego wejściowego zestawu *znanych* reguł. Nowo powstałe reguły były co najmniej tak samo silne jak reguły oryginalne, tzn. ich wartości wsparcia i zaufania były nie mniejsze od wartości wsparcia i zaufania reguł, z których zostały wyprowadzone. Zaprezentowana metoda wnioskowania wykorzystywała dwa operatory. *Operator pokrycia* (ang. *cover operator*, pierwotnie zdefiniowany w [14]), umożliwiał generację nowych reguł na podstawie jednej znanej reguły, przy czym znajomość wartości wsparcia i zaufania tej reguły nie była wymagana. Jeśli tylko oryginalna reguła była silna, to nowo powstałe reguły także były silne. Drugim przedstawionym operatorem był *operator rozszerzania* (ang. *extension operator*). W przeciwieństwie do operatora pokrycia, reguły powstałe w wyniku jego użycia były dłuższe od reguły bazowej. Operator ten wymagał ponadto informacji o wartości wsparcia znanych reguł. W [16] przedstawiono również metodę szacowania minimalnej wartości wsparcia i zaufania powstałych reguł na podstawie wartości tych parametrów dla danego zbioru reguł.

Wadą powyższych rozwiązań był fakt, że nie wykorzystywały one wszystkich możliwych zależności zachodzących pomiędzy znanymi regułami asocjacyjnymi. Przykładowo operator pokrycia tworzył nowe reguły wyłącznie na podstawie jednej znanej reguły. W przypadku, gdy znane są wartości parametrów statystycznych reguł, wskazane jest zbadanie wiedzy "współdzielonej" i wykorzystanie wszystkich istniejących związków podczas dalszego wnioskowania. W [15] zaproponowano zatem nowe podejście do eksploracji wiedzy niepełnej, które znacząco rozszerzono w [20]. Polega ono na tym, że wejściowy zestaw znanych reguł asocjacyjnych wstępnie transformowany jest do postaci zbiorów pozycji. Uzyskany w ten sposób zbiór wzorców poddawany jest przekształceniom przy pomocy odpowiednich operatorów, których

zastosowanie powoduje wzbogacenie go o nowe wzorce o wyznaczonych (być może tylko w przybliżeniu) wartościach wsparć. Końcowy zestaw wzorców stanowi podstawę do konstrukcji nowych reguł asocjacyjnych. W [15] wykazano, że uzyskana w ten sposób wiedza stanowi nadzbiór reguł, jakie można uzyskać korzystając z bezpośredniego wnioskowania na podstawie znanych reguł.

Podejście takie odpowiada w pewnym sensie realizacji obu etapów ogólnego algorytmu wyznaczania reguł asocjacyjnych z danych (punkt 2.3). Pierwszym krokiem jest wygenerowanie zbiorów pozycji, drugim zaś konstrukcja reguł asocjacyjnych na podstawie tych zbiorów. Zasadniczą różnicą jest jednak to, że wyprowadzanie zbiorów częstych wykonywane jest wyłącznie na podstawie innych, znanych zbiorów, bez wykorzystania oryginalnej bazy danych.

W niniejszym rozdziale w sposób szczegółowy zaprezentowane zostaną techniki wnioskowania z wiedzy niepełnej reprezentowanej w postaci zbioru wzorców. Zdefiniowane zostaną odpowiednie operatory umożliwiające generację nowych zbiorów pozycji, a także podstawowe twierdzenia i własności.

3.2 Wiedza niepełna

Algorytmy badane w ramach niniejszej pracy zakładają całkowity brak dostępu do danych źródłowych. Przetwarzaną informacją wejściową jest jedynie reprezentacja pewnej fragmentarycznej wiedzy, która ma być poddana dalszej analizie. Przez wiedzę tę będziemy rozumieli pewien wejściowy zestaw znanych zbiorów pozycji.

Definicja 3.1 Wzorce znane *(ang.* known itemsets*) to pewien zbiór wzorców, których wsparcia są dokładnie określone. Zbiór ten oznaczamy przez* $\mathcal{K}$*.*

$\mathcal{K}$ stanowi zatem całkowitą posiadaną wiedzę, która może być wykorzystana do dalszego wnioskowania.

W przypadku, gdy informacją wejściową jest pewien zestaw reguł asocjacyjnych o znanych wsparciach i zaufaniach, należy przekształcić te reguły do równoważnej postaci wzorców. Wystarczy w tym celu przypomnieć, że zbiór pozycji stanowiący bazę reguły ma takie samo wsparcie jak sama reguła (definicja 2.4). Ponadto z definicji 2.5 widać, że wsparcie zbioru będącego poprzednikiem reguły jest równe wsparciu tej reguły podzielonemu przez jej zaufanie. Tym samym dla każdej znanej reguły asocjacyjnej otrzymujemy dwa wzorce, których wartość wsparcia można w prosty sposób wyznaczyć. Przekształcenie wejściowego zbioru reguł R do postaci wzorców $K(R)$ przyj-

muje zatem postać:

$$K(R) = \{X \cup Y | X \rightarrow Y \in R\} \cup \{X | X \rightarrow Y \in R\}.$$

Wiedza niepełna może być reprezentowana także na inne sposoby lub w różnym stopniu szczegółowości. Przykładowo jedną z częściej wykorzystywanych miar jakości modeli danych badających skuteczność kampanii marketingowych jest *współczynnik podniesienia* (ang. *lift*, patrz np. [8]) zdefiniowany jako:

$$lift(X \rightarrow Y) = \frac{sup(X \cup Y)}{sup(X) \cdot sup(Y)}.$$

Jeśli współczynnik ten jest również określony dla reguł wejściowych, to możliwe jest obliczenie wartości wsparcia dla następnika każdej reguły, a tym samym rozszerzenie powyższej definicji $K(R)$ do postaci:

$$K(R) = \{X \cup Y | X \rightarrow Y \in R\} \cup \{X | X \rightarrow Y \in R\} \cup \{Y | X \rightarrow Y \in R\}.$$

W dalszej części zakładamy jednak, że źródłem danych dla opisywanych algorytmów jest wiedza niepełna reprezentowana w formie pośredniej przez zdefiniowany powyżej zbiór $\mathcal{K}$. Dzięki temu prezentowane metody są niezależne od konkretnej postaci informacji wejściowej. W szczególności, jeśli wnioskowanie ma być przeprowadzone dla pewnego zbioru znanych reguł R, to przyjmujemy $\mathcal{K} = K(R)$.

3.3 Wyprowadzanie wzorców ograniczonych

3.3.1 Wzorce ograniczone

Pierwszą zaproponowaną metodą rozszerzania zbioru znanych wzorców $\mathcal{K}$ było wyznaczanie tzw. *wzorców ograniczonych* (ang. *bounded itemsets*). Pojęcie to zostało wprowadzone w [15]. Dla wyznaczonych w ten sposób zbiorów pozycji możliwe jest oszacowanie ich rzeczywistej wartości wsparcia, przy czym oszacowanie to nie zawsze musi być dokładne. Każdorazowo jednak znany jest przedział wartości, w którym to wsparcie się znajduje.

Niech Y i Z będą pewnymi znanymi zbiorami pozycji z $\mathcal{K}$ takimi, że $Y \subseteq Z$.

Definicja 3.2 Wzorcem ograniczonym *w $\mathcal{K}$ nazywamy taki zbiór X, że*

$$Y \subseteq X \subseteq Z.$$

Zbiór X jest zatem ograniczony z góry przez zbiór Z, zaś z dołu przez zbiór Y. Zauważmy jednak, że X nie musi należeć do $\mathcal{K}$.

Definicja 3.3 *Zbiór wszystkich wzorców ograniczonych w $\mathcal{K}$ oznaczamy przez $BIS(\mathcal{K})$ i definiujemy jako:*

$$BIS(\mathcal{K}) = \{X | \exists Y, Z \in \mathcal{K} \colon Y \subseteq X \subseteq Z\}.$$

Z powyższej definicji wynika, że $BIS(\mathcal{K})$ jest nadzbiorem $\mathcal{K}$. Liczba nowych wzorców powstałych w wyniku wykorzystania powyższego mechanizmu silnie zależy od stopnia korelacji zbiorów w $\mathcal{K}$. Dla każdych dwóch znanych zbiorów $X, Z \in \mathcal{K}$ takich, że $X \subseteq Z$ otrzymujemy $2^{|Z-X|}$ wzorców ograniczonych.

Korzystając z podstawowej zależności wsparć wzorców, które są w relacji zawierania się zbiorów (własność 2.1), a także z powyższej definicji, możliwe jest oszacowanie wartości rzeczywistego wsparcia dla każdego wzorca ograniczonego $X \in BIS(\mathcal{K})$. Wystarczy jedynie zauważyć, że wsparcie X nie może być mniejsze niż wsparcie każdego jego nadzbioru w $\mathcal{K}$. Nie może być ono także większe niż wsparcie każdego jego podzbioru w $\mathcal{K}$. Rzeczywista wartość wsparcia każdego $X \in BIS(\mathcal{K})$ zawiera się zatem w przedziale:

$$sup(X) \geqslant max(\{sup(Z) | Z \in \mathcal{K} \wedge X \subseteq Z\}) \wedge$$
$$sup(X) \leqslant min(\{sup(Y) | Y \in \mathcal{K} \wedge Y \subseteq X\}).$$

Ze względu na to, że wsparcie dla zbiorów ograniczonych określone jest w postaci przedziału, konieczne jest odpowiednie rozszerzenie notacji wprowadzonej w punkcie 2.2.

Definicja 3.4 Wsparciem pesymistycznym *zbioru $X \in BIS(\mathcal{K})$ względem $\mathcal{K}$ nazywamy maksymalną wartość wśród wsparć wszystkich jego nadzbiorów w $\mathcal{K}$:*

$$pSup(X, \mathcal{K}) = max(\{sup(Z) | Z \in \mathcal{K} \wedge X \subseteq Z\}).$$

Wsparciem optymistycznym *zbioru $X \in BIS(\mathcal{K})$ względem $\mathcal{K}$ nazywamy minimalną wartość wśród wsparć wszystkich jego podzbiorów w $\mathcal{K}$:*

$$oSup(X, \mathcal{K}) = min(\{sup(Y) | Y \in \mathcal{K} \wedge Y \subseteq X\}).$$

Dla każdego znanego zbioru $X \in \mathcal{K}$, wartości wsparcia pesymistycznego i optymistycznego są takie same i równe $sup(X)$. W przypadku zbiorów $X \in BIS(\mathcal{K}) \setminus \mathcal{K}$ rzeczywista, nieznana wartość $sup(X)$ jest nie mniejsza niż $pSup(X, \mathcal{K})$ i nie większa niż $oSup(X, \mathcal{K})$. Może się jednak zdarzyć, że $pSup(X, \mathcal{K}) = oSup(X, \mathcal{K}) = sup(X)$. Zbiory X, dla których taka równość zachodzi, nazywane są *zbiorami dokładnie ograniczonymi* w $\mathcal{K}$ (ang. *exact bounded itemsets*).

Definicja 3.5 *Zbiór wszystkich wzorców dokładnie ograniczonych w* $\mathcal{K}$ *oznaczamy przez* $EBIS(\mathcal{K})$ *i definiujemy jako:*

$$EBIS(\mathcal{K}) = \{X | \exists Y, Z \in \mathcal{K} \colon Y \subseteq X \subseteq Z \wedge pSup(X, \mathcal{K}) = oSup(X, \mathcal{K})\}.$$

$EBIS(\mathcal{K})$ zawiera zatem wszystkie te zbiory z $BIS(\mathcal{K})$, które są dokładnie ograniczone, a więc: $\mathcal{K} \subseteq EBIS(\mathcal{K}) \subseteq BIS(\mathcal{K})$.

Podstawowa własność wsparcia zbiorów pozycji (własność 2.1) zachodzi także dla zdefiniowanych wcześniej wartości wsparć optymistycznych i pesymistycznych.

Własność 3.1

$$\forall X, Y \in BIS(\mathcal{K}) \colon X \subseteq Y \Rightarrow$$
$$pSup(X, \mathcal{K}) \geqslant pSup(Y, \mathcal{K}) \wedge oSup(X, \mathcal{K}) \geqslant oSup(Y, \mathcal{K}).$$

Przedziały wartości wsparć dla zbiorów ograniczonych wyznaczane są na podstawie wartości wsparć zbiorów znanych $\mathcal{K}$. Zgodnie z definicją 3.4, dla każdego $X \in BIS(\mathcal{K})$ istnieje taki zbiór $Z \supseteq X, Z \in \mathcal{K}$, że $sup(Z) = pSup(X, \mathcal{K})$. Ponadto istnieje także pewien zbiór $Y \subseteq X, Y \in \mathcal{K}$ taki, że $sup(Y) = oSup(X, \mathcal{K})$. Spostrzeżenie to prowadzi do następującej własności [18]:

Własność 3.2 *Niech* $X \in BIS(\mathcal{K})$*. Wtedy:*

$$pSup(X, EBIS(\mathcal{K})) = pSup(X, \mathcal{K}),$$
$$oSup(X, EBIS(\mathcal{K})) = oSup(X, \mathcal{K}).$$

Powyższy wniosek oznacza, że rozszerzenie zbioru $\mathcal{K}$ o zbiory dokładnie ograniczone nie wpływa na oszacowania wartości wsparć dla pozostałych zbiorów ograniczonych. Inaczej mówiąc wzorce $EBIS(\mathcal{K}) \setminus \mathcal{K}$ nie wnoszą nowej wiedzy o wsparciach.

3.3.2 Algorytm BIS

Algorytm `BIS` służący do wyznaczania wzorców ograniczonych i ich wsparć składa się z dwóch etapów. W pierwszej kolejności na podstawie zadanego zbioru wzorców znanych $\mathcal{K}$ wyznaczane są wszystkie wzorce ograniczone. W drugim kroku dla wyznaczonych wzorców obliczane są wsparcia optymistyczne i pesymistyczne.

Część pierwsza algorytmu rozpoczyna się od wyszukania zbiorów maksymalnych $MaxBIS$ oraz minimalnych $MinBIS$ w $\mathcal{K}$ oraz dodania ich do zbioru wynikowego. Następnie, dla każdego zbioru maksymalnego

$Z \in MaxBIS$ i wszystkich jego podzbiorów $X \in MinBIS$, wyznaczane są wszystkie zbiory będące podzbiorem właściwym Z i nadzbiorem właściwym X. Utworzone w ten sposób wzorce dodawane są do zbioru wynikowego. Algorytm tworzy wzorce ograniczone w sposób nieunikalny.

Algorytm 3.1: BIS

input : $\mathcal{K}$ - wzorce znane
output: BIS - wzorce ograniczone z wyznaczonymi oszacowaniami wsparć

begin
  $MaxBIS \leftarrow$ `MaximalSets(`$\mathcal{K}$`)`
  $MinBIS \leftarrow$ `MinimalSets(`$\mathcal{K}$`)`
  $BIS \leftarrow MaxBIS \cup MinBIS$
  foreach $Z \in MaxBIS$ **do**
    $\mathcal{X} \leftarrow \{X \in MinBIS | X \subset Z\}$
    foreach $X \in \mathcal{X}$ **do**
      $\mathcal{L} \leftarrow$ `NonEmptyProperSubsets(`$Z \setminus X$`)`
      $BIS \leftarrow BIS \cup \{X \cup Y | Y \in \mathcal{L}\}$
    end
  end
  `PSup(`BIS`)`
  `OSup(`BIS`)`
end

W celu wyznaczenia wszystkich zbiorów Y takich, że $X \subset Y \subset Z$, $Z \in MaxBIS$, $X \in MinBIS$, tworzony jest tymczasowy zbiór $Z \setminus X$, a następnie generowane są wszystkie niepuste podzbiory właściwe tego zbioru przy użyciu funkcji `NonEmptyProperSubsets`. Funkcja ta jest iteracyjna, a zasada jej działania jest zbliżona do algorytmu generowania wzorców częstych Apriori. W pierwszym kroku tworzone są zbiory jednoelementowe składające się z kolejnych pozycji zbioru $Z \setminus X$. W kroku $k > 1$, na podstawie zbiorów wyznaczonych w kroku $k-1$, tworzone są zbiory o długości k. W tym celu każde dwa zbiory o długości $k-1$ o takich samych elementach na pierwszych $k-2$ pozycjach i ostatnich pozycjach spełniających warunek mniejszości między ich wartościami, łączone są w nowy zbiór o długości k. Taki sposób łączenia zapewnia, że każdy nowy zbiór będzie wygenerowany dokładnie raz. Proces łączenia jest kontynuowany, dopóki wartość k jest mniejsza od długości zbioru $Z \setminus X$.

Po wyznaczeniu wszystkich wzorców ograniczonych $BIS(\mathcal{K})$ obliczane

Funkcja 3.2: NonEmptyProperSubsets(Y)

```
begin
    if |Y| > 1 then
        L_1 ← {{a}|a ∈ Y}
        for k ← 1 to |Y| − 2 do
            foreach F, G ∈ L_k do
                if F[1] = G[1] ∧ ... ∧ F[k−1] = G[k−1] ∧ F[k] < G[k]
                then
                    Z ← F[1] · ... · F[k] · G[k]
                    L_{k+1} ← L_{k+1} ∪ {Z}
                end
            end
        end
        return ⋃ L_i
    end
    else
        return {}
    end
end
```

są oszacowania wartości wsparć pesymistycznych i optymistycznych nowych wzorców przy pomocy iteracyjnych algorytmów `PSup` i `OSup`. Oba algorytmy zakładają, że dla wzorców $\mathcal{K}$ optymistyczne i pesymistyczne wartości wsparcia są wstępnie zainicjalizowane ich znaną wartością wsparcia, a dla pozostałych wzorców z $BIS(\mathcal{K})$ wsparcie optymistyczne jest zainicjalizowane wartością inf, zaś pesymistyczne wartością 0.

Algorytm `PSup` w każdej iteracji wyznacza wartości wsparć pesymistycznych nowych wzorców korzystając z wartości wsparć pesymistycznych wszystkich ich nadzbiorów o jeden dłuższych. W tym celu w pierwszym kroku wybierane są z $BIS(\mathcal{K})$ wzorce Z o długości k równej długości najdłuższego wzorca w $\mathcal{K}$. Z definicji wzorców ograniczonych wiadomo, że wybrane w ten sposób wzorce są wzorcami znanymi, a tym samym ich wsparcie jest określone dokładnie. Następnie dla każdego wzorca Z wyszukiwane są jego podzbiory $X \in BIS(\mathcal{K})$ o długości $k-1$, których wsparcie nie jest określone dokładnie, tzn. nie są wzorcami znanymi. W przypadku, gdy wartość pesymistycznego wsparcia wzorca X jest mniejsza od wartości wsparcia pesymistycznego jego nadzbioru Z, pierwsza wartość zastępowana jest drugą. Po wykonaniu iteracji wszystkie wzorce o długości $k-1$ mają poprawnie

wyznaczoną wartość pesymistycznego wsparcia, wartość k jest zmniejszana o jeden i algorytm przechodzi do kolejnej iteracji. Proces kończy się w chwili, gdy k osiągnie wartość równą długości najkrótszego zbioru w $\mathcal{K}$.

Algorytm 3.3: PSup

input : BIS - wzorce ograniczone
input : $MaxK$ - długość najdłuższego wzorca w $\mathcal{K}$
input : $MinK$ - długość najkrótszego wzorca w $\mathcal{K}$
output: wartości pesymistycznego wsparcia dla $X \in BIS(\mathcal{K})$

```
begin
    assert: ∀X ∈ K: X.oSup = X.pSup = X.sup
    assert: ∀X ∈ BIS(K) \ K: X.pSup = 0 ∧ X.oSup = ∞
    for k ← MaxK to MinK + 1 do
        foreach k-itemset Z ∈ BIS do
            foreach (k − 1)-itemset X ⊂ Z do
                if X ∈ BIS then
                    if not Exact(X) then
                        X.pSup ← max(X.pSup, Z.pSup)
                    end
                end
            end
        end
    end
end
```

Zasada działania algorytmu `OSup` jest analogiczna do algorytmu `PSup`, przy czym wartości wsparć optymistycznych nowych wzorców wyznaczane są w oparciu o wsparcia optymistyczne ich podzbiorów o jeden krótszych. Tym samym główna pętla algorytmu przechodzi od wzorców najkrótszych do najdłuższych.

Algorytm 3.4: OSup

input : BIS - wzorce ograniczone
input : $MaxK$ - długość najdłuższego wzorca w $\mathcal{K}$
input : $MinK$ - długość najkrótszego wzorca w $\mathcal{K}$
output: wartości optymistycznego wsparcia dla $X \in BIS(\mathcal{K})$

```
begin
    assert: ∀X ∈ K: X.oSup = X.pSup = X.sup
    assert: ∀X ∈ BIS(K) \ K: X.pSup = 0 ∧ X.oSup = ∞
    for k ← MinK + 1 to MaxK do
        foreach k-itemset Z ∈ BIS do
            if not Exact(Z) then
                foreach (k − 1)-itemset X ⊂ Z do
                    if X ∈ BIS then
                        Z.oSup ← min(Z.oSup, X.oSup)
                    end
                end
            end
        end
    end
end
```

Funkcja 3.5: MaximalSets($\mathcal{K}$)

return $\{X \in \mathcal{K} | \neg\exists Y \in \mathcal{K} \colon X \subset Y\}$

Funkcja 3.6: MinimalSets($\mathcal{K}$)

return $\{X \in \mathcal{K} | \neg\exists Y \in \mathcal{K} \colon X \supset Y\}$

Funkcja 3.7: Exact(X)

return $X.pSup = X.oSup$

3.4 Wyprowadzanie wzorców rozszerzonych i skurczonych

W [20] w sposób znaczący rozszerzono przedstawiony w poprzednim punkcie mechanizm wyprowadzania nowych wzorców na podstawie wzorców znanych. W przeciwieństwie do wzorców ograniczonych w $\mathcal{K}$, dla których wartość rzeczywistego wsparcia może nie być znana, a jedynie oszacowana przez wartość pesymistyczną i optymistyczną, dla nowych zbiorów pozycji wyprowadzonych przy pomocy opisanych dalej metod wartość ta jest zawsze dokładnie określona. Ponadto część z tych zbiorów nie jest ograniczona przez żadne inne zbiory z $\mathcal{K}$, tzn. nie ma podzbiorów lub nadzbiorów w $\mathcal{K}$. Połączenie tych metod z wcześniej zaprezentowaną metodą ograniczania stanowi silne narzędzie umożliwiające znaczne zwiększenie liczby wyprowadzonych wzorców.

3.4.1 Wzorce rozszerzone

Przed przytoczeniem ścisłej definicji zaprezentujemy następujące rozumowanie. Niech $X, Y, Z \in I$ będą pewnymi zbiorami pozycji takimi, że $X \subseteq Y$ oraz $X \subseteq Z$. Niech ponadto $sup(X) = sup(Z)$. Wynika z tego, że zbiory transakcji z bazy danych D wspierające X i Z są takie same: $\{T \in D | X \subseteq T\} = \{T \in D | Z \subseteq T\}$. Każda transakcja wspierająca X zawiera zatem pozycje ze zbioru $Z \setminus X$. Zbiór Y jest nadzbiorem zbioru X, a więc wszystkie transakcje, które go wspierają, wspierają również X: $\{T \in D | Y \subseteq T\} \subseteq \{T \in D | X \subseteq T\}$, czyli także zawierają zbiór $Z \setminus X$. Jeśli więc zbiór Y rozszerzymy o pozycje ze zbioru $Z \setminus X$, to operacja ta nie spowoduje zmiany wartości wsparcia. Utworzony w ten sposób zbiór Y' będzie miał zatem wsparcie takie samo jak zbiór Y. Powyższe rozumowanie prowadzi do następującego wniosku:

Twierdzenie 3.1 [20] *Niech $X, Y, Z \in I$, $X \subseteq Y, Z$ oraz $sup(X) = sup(Z)$. Jeżeli $Y' = Y \cup Z$, to:*

$$Y' \supseteq Y \text{ oraz } sup(Y') = sup(Y).$$

Definicja 3.6 Wzorcem rozszerzonym w $\mathcal{K}$ *nazywamy taki wzorzec $Y \cup Z$, gdzie $Y, Z \in \mathcal{K}$, że:*

$$\exists X \in \mathcal{K} \colon X \subseteq Y, Z \wedge sup(X) = sup(Z).$$

Zastosowanie powyższego twierdzenia dla wzorców znanych daje możliwość wygenerowania nowych wzorców o dokładnie określonych wsparciach.

Zauważmy ponadto, że wyprowadzone wzorce mogą mieć tę właściwość, że nie istnieje dla nich żaden nadzbiór wśród znanych wzorców $\mathcal{K}$. Wynika to z tego, że zbiory rozszerzone powstają poprzez zsumowanie dwóch innych zbiorów znanych. Widać zatem, że zbiory rozszerzone nie muszą być ograniczone w $\mathcal{K}$.

Definicja 3.7 *Zbiór wszystkich wzorców rozszerzonych względem $\mathcal{K}$ oznaczamy przez $EIS(\mathcal{K})$ i definiujemy jako:*

$$EIS(\mathcal{K}) = \{Y \cup Z | Y, Z \in \mathcal{K} \wedge (\exists X \in \mathcal{K} \colon X \subseteq Y, Z \wedge sup(X) = sup(Z))\}.$$

Wsparcie każdego zbioru w $EIS(\mathcal{K})$ można wyznaczyć korzystając z twierdzenia 3.1.

3.4.2 Wzorce skurczone

Podobne rozumowanie do przedstawionego wcześniej można przeprowadzić również w drugą stronę. Niech analogicznie $X, Y, Z \in I$ będą pewnymi zbiorami pozycji takimi, że $X \subseteq Y$ oraz $X \subseteq Z$, a także $sup(X) = sup(Z)$. Jeśli zbiór X występuje w dokładnie tych samych transakcjach co zbiór Z w bazie danych D, to w ramach tych transakcji zbiory te są „równoważne", tzn. pozycje ze zbioru $Z \setminus X$ nie wpływają na wartość wsparcia zbiorów X i Z. Zbiór Y jest nadzbiorem X, a więc wspierany jest przez część transakcji wspierających X: $\{T \in D | Y \subseteq T\} \subseteq \{T \in D | X \subseteq T\}$. Odrzucenie z Y pozycji $Z \setminus X$ nie spowoduje zatem zmiany wartości wsparcia. Powstały w ten sposób zbiór Y'' będzie miał więc takie samo wsparcie jak zbiór Y.

Ponadto można zauważyć, że zbiór z którego usuwane są pozycje $Z \setminus X$, niekoniecznie musi być nadzbiorem X. Wystarczy jedynie, aby występował w tych samych transakcjach w bazie danych D co zbiór $Y \supseteq X$, a więc musi być pewnym podzbiorem $V \subseteq Y$ o takim samym wsparciu. Jeśli jednak $X \not\subseteq V$, to jest możliwe, że: $\{T \in D | V \setminus (Z \setminus X) \subseteq T\} \not\subseteq \{T \in D | X \subseteq T\}$. Jak wcześniej wspomniano, zbiory X i Z są „równoważne" jedynie w ramach transakcji wspierających X. Dla zachowania wartości wsparcia konieczne jest więc zapewnienie, aby nowy zbiór Y'' był pokrywany tylko przez te transakcje. Y'' musi więc być nadzbiorem X.

Twierdzenie 3.2 [20] *Niech $X, Y, Z \in I$, $X \subseteq Y, Z$ oraz $sup(X) = sup(Z)$. Wtedy dla każdego $V \subseteq Y$, $sup(V) = sup(Y)$, jeżeli $Y'' = X \cup (V \setminus Z)$ to:*

$$Y'' \subseteq Y \text{ oraz } sup(Y'') = sup(Y).$$

Definicja 3.8 Wzorcem skurczonym w $\mathcal{K}$ *nazywamy taki wzorzec* $X \cup (V \setminus Z)$*, gdzie* $X, V, Z \in \mathcal{K}$*, że:*

$$X \subseteq Z \wedge sup(X) = sup(Z) \wedge (\exists Y \in \mathcal{K}\colon X \subseteq Y \wedge V \subseteq Y \wedge sup(V) = sup(Y)).$$

Podobnie jak w przypadku wzorców rozszerzonych, przedstawione twierdzenie umożliwia dokładne wyznaczenie wsparć dla nowych wzorców niewystępujących w $\mathcal{K}$. Ponadto powstałe wzorce skurczone nie muszą być ograniczone w $\mathcal{K}$. Sytuacja taka zachodzi wtedy, gdy żaden wzorzec znany nie jest podzbiorem wzorca skurczonego.

Definicja 3.9 *Zbiór wszystkich zbiorów skurczonych względem* $\mathcal{K}$ *oznaczamy przez* $SIS(\mathcal{K})$ *i definiujemy jako:*

$$SIS(\mathcal{K}) = \{X \cup (V \setminus Z) | X, V, Z \in \mathcal{K} \wedge X \subseteq Z \wedge sup(X) = sup(Z) \wedge (\exists Y \in \mathcal{K}\colon X \subseteq Y \wedge V \subseteq Y \wedge sup(V) = sup(Y))\}.$$

Twierdzenie 3.2 umożliwia wyznaczenie wartości wsparcia dla każdego zbioru skurczonego w $SIS(\mathcal{K})$.

Z powyższych definicji otrzymujemy: $\mathcal{K} \subseteq EIS(\mathcal{K})$ i $\mathcal{K} \subseteq SIS(\mathcal{K})$. Powiększenie znanych wzorców $\mathcal{K}$ o wszystkie wzorce rozszerzone $EIS(\mathcal{K})$ i skurczone $SIS(\mathcal{K})$ umożliwia wyprowadzenie większej liczby zbiorów ograniczonych niż bezpośrednio z $\mathcal{K}$, tzn. $BIS(\mathcal{K}) \subseteq BIS(\mathcal{K} \cup EIS(\mathcal{K}) \cup SIS(\mathcal{K}))$ oraz $EBIS(\mathcal{K}) \subseteq EBIS(\mathcal{K} \cup EIS(\mathcal{K}) \cup SIS(\mathcal{K}))$. Ponadto w przypadku, gdy zbiór ograniczony (niekoniecznie dokładnie) jest jednocześnie zbiorem skurczonym lub rozszerzonym, to jego wartość wsparcia jest wyznaczona dokładnie. Umożliwia to lepsze oszacowanie wsparć dla pozostałych zbiorów ograniczonych.

Jednokrotne zastosowanie operatorów EIS i SIS do zbiorów znanych $\mathcal{K}$ zazwyczaj wyprowadza tylko część zbiorów, które można otrzymać w wyniku ich wykorzystania. Ponowne obliczenie EIS i SIS może zakończyć się wyznaczeniem nowych zbiorów. Pełne wykorzystanie opisanych operatorów wymaga więc iteracyjnego obliczania zbiorów skurczonych i rozszerzonych oraz dokładnie ograniczonych aż do momentu, w którym żaden nowy zbiór nie zostanie wygenerowany.

Definicja 3.10 *Zbiór wszystkich wzorców dokładnie ograniczonych, które można wyprowadzić z* $\mathcal{K}$ *poprzez wielokrotne wykorzystanie operatorów* EIS *i* SIS *oznaczamy przez* $EBIS^*(\mathcal{K})$ *i definiujemy jako:*

$$EBIS^*(\mathcal{K}) = E_k(\mathcal{K}),\ gdzie:$$

- $E_1(\mathcal{K}) = EBIS(\mathcal{K})$,
- $E_n(\mathcal{K}) = EBIS(E_{n-1}(\mathcal{K}) \cup EIS(E_{n-1}(\mathcal{K})) \cup SIS(E_{n-1}(\mathcal{K})))$ *dla* $n \geqslant 2$,
- k *jest najmniejszą wartością* n *taką, że* $E_n(\mathcal{K}) = E_{n+1}(\mathcal{K})$.

Definicja 3.11 *Zbiór wszystkich wzorców ograniczonych, które można wyprowadzić z* $\mathcal{K}$ *poprzez wielokrotne wykorzystanie operatorów EIS i SIS oznaczamy przez* $BIS^*(\mathcal{K})$ *i definiujemy jako:*

$$BIS^*(\mathcal{K}) = BIS(EBIS^*(\mathcal{K})).$$

3.4.3 Algorytm EIS

Algorytm `EIS` wyznaczania wzorców rozszerzonych z $\mathcal{K}$ korzysta bezpośrednio z definicji 3.7 oraz twierdzenia 3.1. Z każdym wzorcem znanym związane jest pole *Supers-SameSup* przeznaczone do przechowywania listy wzorców będących jego nadzbiorami w $\mathcal{K}$ o tym samym wsparciu. Na etapie przygotowawczym algorytmu listy *Supers-SameSup* są inicjalizowane dla każdego wzorca znanego. Następnie algorytm wybiera kolejno wzorce znane Y i wyszukuje wszystkie ich podzbiory X w $\mathcal{K}$. Wyznaczone w ten sposób dane, tzn. Y, $\{X|X \subseteq Y\}$ oraz listy *Supers-SameSup*, są wystarczające do wyznaczenia w sposób nieunikalny wszystkich wzorców ograniczonych.

Algorytm 3.8: EIS

input : $\mathcal{K}$ - wzorce znane
output: EIS - wzorce rozszerzone z wyznaczonymi wsparciami

begin
  assert: dla każdego $X \in \mathcal{K}$ lista *Supers-SameSup* nadzbiorów zbioru X w $\mathcal{K}$ o wsparciu równym $sup(X)$ jest znana
  foreach $Y \in \mathcal{K}$ **do**
    $\mathcal{X} \leftarrow$ `Subsets`$(Y, \mathcal{K})$
    foreach $X \in \mathcal{X}$ **do**
      foreach $Z \in X.Supers\text{-}SameSup$ **do**
        $Y' \leftarrow Y \cup Z$
        $Y'.sup \leftarrow Y.sup$
        $EIS \leftarrow EIS \cup \{Y'\}$
      end
    end
  end
end

3.4.4 Algorytm SIS

Algorytm `SIS` wyznaczania wzorców skurczonych z $\mathcal{K}$ został skonstruowany w oparciu o definicję 3.9 i twierdzenie 3.2. Z każdym wzorcem znanym związane są pola *Subs-SameSup* oraz *Supers-SameSup* przeznaczone do przechowywania listy wzorców będących kolejno jego podzbiorami lub nadzbiorami w $\mathcal{K}$ o tym samym wsparciu. Na etapie przygotowawczym algorytmu obie listy są inicjalizowane dla każdego wzorca znanego. Następnie algorytm wybiera kolejno wzorce znane Y i wyszukuje wszystkie ich podzbiory X w $\mathcal{K}$. Korzystając z wyznaczonych danych, tj. Y, $\{X|X \subseteq Y\}$ oraz list *Subs-SameSup* i *Supers-SameSup*, algorytm w kolejnych pętlach generuje wzorce skurczone. Wzorce te są tworzone w sposób nieunikalny.

Algorytm 3.9: SIS

input : $\mathcal{K}$ - wzorce znane
output: SIS - wzorce skurczone z wyznaczonymi wsparciami

```
begin
    assert: dla każdego X ∈ K listy Subs-SameSup
            i Supers-SameSup podzbiorów i nadzbiorów zbioru
            X w K o wsparciu równym sup(X) są znane
    foreach Y ∈ K do
        X ← Subsets(Y, K)
        foreach X ∈ X do
            foreach Z ∈ X.Supers-SameSup do
                foreach V ∈ Y.Subs-SameSup do
                    Y'' ← X ∪ (V \ Z)
                    Y''.sup ← Y.sup
                    SIS ← SIS ∪ {Y''}
                end
            end
        end
    end
end
```

Wyznaczanie list *Subs-SameSup* i *Supers-SameSup*

Algorytm `SIS` wymaga wstępnego wyznaczenia dla każdego wzorca znanego list *Subs-SameSup* i *Supers-SameSup* wszystkich podzbiorów i nadzbiorów w $\mathcal{K}$ o tym samym wsparciu. W tym celu wykorzystywana jest

procedura pomocnicza `AllSupersAndSubsSameSup` przedstawiona poniżej. W przypadku algorytmu `EIS` konieczne jest wyznaczenie jedynie list *Supers-SameSup*. Wykonywane jest to zmodyfikowaną wersją procedury `AllSupersAndSubsSameSup`, która nie zapisuje listy podzbiorów o tym samym wsparciu w polu *Subs-SameSup*.

Procedura 3.10: AllSupersAndSubsSameSup($\mathcal{K}$)

input : $\mathcal{K}$ - wzorce znane
output: listy *Subs-SameSup* i *Supers-SameSup* podzbiorów i nadzbiorów zbioru X w $\mathcal{K}$ o wsparciu równym $sup(X)$ dla każdego $X \in \mathcal{K}$

```
begin
    foreach Z ∈ K do
        X ← Subsets-SameSup(Z, K)
        Z.Subs-SameSup ← X
        foreach X ∈ X do
            X.Supers-SameSup ← X.Supers-SameSup ∪ {Z}
        end
    end
end
```

Funkcja 3.11: Subsets($Y, \mathcal{K}$)

return $\{X \in \mathcal{K} | X \subseteq Y\}$

Funkcja 3.12: Subsets-SameSup($Y, \mathcal{K}$)

return $\{X \in \mathcal{K} | X \subseteq Y \wedge sup(X) = sup(Y)\}$

3.4.5 Algorytm EBIS*

Konstrukcja algorytm `EBIS*` jest bezpośrednio zaczerpnięta z definicji 3.10. W celu wyznaczenia wszystkich wzorców dokładnie ograniczonych w $\mathcal{K}$, algorytmy `EIS`, `SIS` i `EBIS` są wykonywane wielokrotnie do czasu, aż kolejna iteracja nie wyprowadzi żadnego nowego wzorca.

Warto przy tym zauważyć, że pętla przetwarzania algorytmu `EIS` jest częścią pętli przetwarzania algorytmu `SIS`. Tym samym możliwe jest łatwe połączenie obu tych algorytmów w nowy algorytm `EISAndSIS` i wyznaczanie wzorców ograniczonych jednocześnie ze wzorcami skurczonymi. Ponadto, $E_{k-1} \subseteq \texttt{EIS}(E_{k-1})$ oraz $E_{k-1} \subseteq \texttt{SIS}(E_{k-1})$. Dzięki tym dwóm obserwacjom algorytm `EBIS*` upraszcza się względem definicji 3.10 do postaci przedstawionej poniżej.

Algorytm 3.13: EBIS*

input : $\mathcal{K}$ - wzorce znane
output: $EBIS^*$ - wszystkie wzorce dokładnie ograniczone w $\mathcal{K}$

begin
  $E_1 \leftarrow \texttt{EBIS}(\mathcal{K})$
  $k \leftarrow 1$
  repeat
    $k \leftarrow k + 1$
    $E_k \leftarrow \texttt{EBIS}(\texttt{EISAndSIS}(E_{k-1}))$
  until $E_k \neq E_{k-1}$
  $EBIS^* \leftarrow E_k$
end

Rozdział 4

Zwięzłe reprezentacje wzorców

4.1 Wprowadzenie

Proces odkrywania wiedzy, czyli zależności statystycznie istotnych i interesujących z punktu widzenia użytkownika, jest w ogólności zadaniem przeszukiwania przestrzeni wszystkich możliwych rozwiązań w celu znalezienia tych, które spełniają pewne ustalone kryteria. Przykładowo klasyczny problem odkrywania wzorców częstych sprowadza się do odnalezienia tych wzorców występujących w bazie danych, których wsparcie przekracza ustaloną minimalną wartość. Naiwnym rozwiązaniem tego problemu mogłoby więc być wyznaczenie wszystkich wzorców, obliczenie ich wsparć, a następnie odrzucenie wzorców słabych. Rozwiązanie to jest jednak całkowicie niepraktyczne, ponieważ liczba wszystkich wzorców rośnie wykładniczo względem liczby różnych pozycji w bazie danych.

Problem ogromnej wielkości przestrzeni rozwiązań jest wspólny dla większości zagadnień eksploracji danych. Badania nad algorytmami skupiają się zatem na wybraniu optymalnej ścieżki poszukiwań, dzięki której w każdym kolejnym kroku odrzucany jest możliwie największy fragment przestrzeni, o którym z góry wiadomo, że nie zawiera rozwiązań spełniających zadane kryteria, a zatem może zostać pominięty podczas dalszych poszukiwań. W przypadku wykrywania wszystkich wzorców częstych zazwyczaj wykorzystuje się podstawową własność wsparć 2.1, która umożliwia odrzucenie z analizy wszystkich nadzbiorów zbiorów, o których już wiadomo, że są rzadkie. Wspomniany w 2.3 algorytm Apriori zawdzięcza tej technice swoją nazwę.

Dla wielu zastosowań istniejące algorytmy wykorzystujące takie podejście umożliwiają uzyskanie zadowalających efektów. Technika ta nie jest jed-

nak wystarczająca wtedy, gdy liczba rozwiązań spełniających zadane kryteria sama w sobie jest bardzo duża. Sytuacja taka może spowodować dwa podstawowe problemy: nieakceptowalny czas obliczeń potrzebny na ich odkrycie oraz małą praktyczną przydatność uzyskanych wyników. Przykładowo przeanalizowanie i efektywne wykorzystanie kilku czy kilkunastu tysięcy statystycznie istotnych reguł asocjacyjnych może okazać się niewykonalne. W takim przypadku możliwe jest wprowadzenie dodatkowych ograniczeń lub przybliżeń (np. [29], [31]) czy też wykorzystanie różnorodnych technik automatycznego określania stopnia „użyteczności" odkrywanej wiedzy (np. [4], [12], [32], [33]).

Inną istotną techniką mającą zredukować przedstawiony problem jest wykorzystanie tzw. *zwięzłych reprezentacji wiedzy*. Umożliwiają one przechowywanie wiedzy w możliwie najbardziej zredukowanej, a jednocześnie bezstratnej postaci. Korzystając z własności opisywanej wiedzy i zależności między rozwiązaniami definiuje się pewną klasę rozwiązań, której liczność jest nie większa od liczności wszystkich poprawnych rozwiązań (pożądanym jest, aby była znacznie mniejsza). W rezultacie ta część odnalezionych rozwiązań, która nie należy do wspomnianej klasy, nie jest przechowywana *explicite*, niemniej istnieje efektywna metoda wyznaczenia ich na podstawie rozwiązań należących do wyróżnionej klasy. Na zwięzłą reprezentację wiedzy składa się zatem definicja pewnej klasy rozwiązań oraz mechanizm wnioskowania o pozostałych rozwiązaniach w oparciu o rozwiązania należące do tej klasy.

W niniejszym rozdziale przedstawione zostaną dwie podstawowe reprezentacje wszystkich wzorców częstych, tj. reprezentacja oparta na zbiorach zamkniętych [26] oraz reprezentacja generatorowa [17], a następnie oparte na nich reprezentacje wiedzy niepełnej [20].

4.2 Reprezentacje wszystkich wzorców częstych

Wszystkie omawiane w pracy reprezentacje wzorców są *bezstratne*. Reprezentację pewnego zbioru wzorców $\mathcal{X}$ uważamy za bezstratną wtedy, gdy umożliwia wyznaczenie wszystkich wzorców $X \in \mathcal{X}$ oraz wartości ich wsparć wyłącznie w oparciu o wzorce należące do tej reprezentacji. Inaczej mówiąc, wartość informacyjna bezstratnej reprezentacji musi być nie mniejsza od wartości informacyjnej zbioru reprezentowanych przez nią wzorców. Omawiane w dalszej części rozdziału reprezentacje są bezstratne.

Jak wspomniano we wstępie do niniejszego rozdziału, reprezentacje wzorców częstych definiują pewne klasy wzorców. Odkrywanie reprezentacji polega zatem na wyprowadzaniu wyłącznie wzorców określonego typu. W lite-

raturze można znaleźć m.in. reprezentacje wzorców częstych, które bazują na następujących klasach wzorców: zbiorach zamkniętych [26], generatorach [17], zbiorach wolnych od dysjunkcji [7], generatorach wolnych od dysjunkcji [17], uogólnionych generatorach wolnych od dysjunkcji [23]. Wyczerpujące omówienie i porównanie powyższych reprezentacji można znaleźć w [18].

Reprezentacje wiedzy niepełnej korzystają z własności reprezentacji opartej na zbiorach zamkniętych i reprezentacji generatorowej, dlatego zostaną one w kolejnych punktach pokrótce omówione.

4.2.1 Reprezentacja oparta na zbiorach zamkniętych

Reprezentacja oparta na zbiorach zamkniętych jest szeroko stosowaną reprezentacją wzorców częstych. Przed podaniem ścisłej definicji konieczne jest wprowadzenie pojęcia *domknięcia* zbioru pozycji i kilku własności.

Definicja 4.1 Domknięcie zbioru $X \subseteq I$ *oznaczamy przez* $\gamma(X)$ *i definiujemy jako:*

$$\gamma(X) = \bigcap \{T \in D | T \supseteq X\}.$$

Domknięcie zbioru X to zatem taki jego nadzbiór Y, który stanowi część wspólną wszystkich transakcji bazy danych D pokrywających zbiór X. Z powyższej definicji wynikają następujące własności:

Własność 4.1 *Niech* $X, Y \subseteq I$*. Wtedy:*

$$sup(\gamma(X)) = sup(X),$$
$$sup(X) \neq sup(Y) \Rightarrow \gamma(X) \neq \gamma(Y),$$
$$X \subseteq Y \subseteq \gamma(X) \Rightarrow sup(X) = sup(Y).$$

Wsparcia dowolnego zbioru i jego domknięcia są równe. Ponadto wsparcie każdego zbioru Y będącego nadzbiorem X oraz podzbiorem $\gamma(X)$ jest także równe wsparciu X. Warto jednocześnie zauważyć, że dla każdego zbioru X istnieje dokładnie jeden zbiór będący jego domknięciem. Domknięcie zbioru X można zatem równoważnie zdefiniować jako maksymalny nadzbiór $Y \supseteq X$ taki, że $sup(X) = sup(Y)$.

Definicja 4.2 *Zbiór* $X \subseteq I$ *nazywamy* **zbiorem zamkniętym**, *jeżeli* $X = \gamma(X)$*. Zbiór wszystkich zbiorów zamkniętych oznaczamy przez* C*:*

$$C = \{X \subseteq I | \gamma(X) = X\}.$$

Zbiór X jest więc zbiorem zamkniętym, jeżeli nie istnieje żaden $Y \supset X$ o takim samym wsparciu. Można łatwo wykazać, że zbiór C jest wystarczający do tego, aby obliczyć wartość wsparcia dowolnego zbioru X. Na

podstawie własności wsparć 2.1 wiadomo, że dla każdego $Y \supseteq X$ zachodzi $sup(X) >= sup(Y)$. Ponadto, jeśli $Y = \gamma(X)$, to $sup(X) = sup(Y)$. Otrzymujemy zatem następującą zależność umożliwiającą obliczenie wartości wsparcia dla każdego $X \subseteq I$:

$$\forall X \subseteq I \quad sup(X) = max(\{sup(Y) | Y \in C \wedge X \subseteq Y\}).$$

Zbiór C można zatem uznać za bezstratną reprezentację wszystkich wzorców $X \subseteq I$.

Definicja 4.3 Reprezentacją wzorców częstych $\mathcal{F}$ opartą na zbiorach zamkniętych *nazywamy zbiór wszystkich częstych zbiorów zamkniętych $\mathcal{FC} = \mathcal{F} \cap C$ wraz z wartościami wsparć dla każdego $X \in \mathcal{FC}$.*

Obliczanie wartości wsparcia dla dowolnego wzorca $X \in \mathcal{F}$ w oparciu o reprezentację $\mathcal{FC}$ przebiega następująco:

- Jeżeli istnieje wzorzec $Y \in \mathcal{FC}$ taki, że $X \subseteq Y$, to wzorzec X jest częsty, a jego wsparcie jest równe $sup(X) = max(\{sup(Y) | Y \in \mathcal{FC} \wedge X \subseteq Y\})$.
- W przeciwnym przypadku wzorzec X jest rzadki.

Efektywne algorytmy umożliwiające odkrywanie reprezentacji $\mathcal{FC}$ z danych przedstawiono m.in. w [28], [27], [30], [35].

4.2.2 Reprezentacja generatorowa

Reprezentacja generatorowa zaproponowana w [17] jest oparta na klasie wzorców zwanych *generatorami*. Pojęcie generatorów zbiorów pozycji jest analogiczne do pojęcia domknięcia, a tym samym wspólne są także niektóre własności.

Definicja 4.4 Generatorem zbioru $X \subseteq I$ *nazywamy minimalny zbiór $Y \subseteq X$, którego domknięcie jest równe domknięciu X:*

$$Y \subseteq X \wedge \gamma(Y) = \gamma(X) \wedge \forall Z \subset Y \gamma(Z) \neq \gamma(X).$$

Wsparcia dowolnego zbioru i jego generatora są równe. Ponadto na podstawie powyższej definicji łatwo zauważyć, że generatorem wzorca X jest każdy minimalny wzorzec występujący dokładnie w tych samych transakcjach co X, a więc w przeciwieństwie do domknięcia wzorzec X może mieć więcej niż jeden generator. Zbiór wszystkich generatorów X oznaczamy przez $\mathcal{G}(X)$:

$$\mathcal{G}(X) = MIN\{Y \subseteq X | \gamma(Y) = \gamma(X)\}.$$

Definicja 4.5 *Zbiór $X \subseteq I$ nazywamy* **generatorem**, *jeżeli $\{X\} = \mathcal{G}(X)$. Zbiór wszystkich generatorów oznaczamy przez $\mathcal{G}$:*

$$\mathcal{G} = \bigcup_{X \subseteq I} \mathcal{G}(X).$$

Zbiór X jest więc generatorem, gdy nie istnieje żaden $Y \subset X$ o tym samym wsparciu. W [5] udowodniono ciekawe twierdzenie, że każdy podzbiór generatora jest również generatorem.

Analogicznie jak w przypadku zbioru wszystkich zbiorów zamkniętych C, zbiór wszystkich generatorów $\mathcal{G}$ jest wystarczający do wyprowadzenia wartości wsparcia dowolnego zbioru $X \subseteq I$ według następującej formuły:

$$\forall X \subseteq I \quad sup(X) = min(\{sup(Y) | Y \in \mathcal{G} \wedge Y \subseteq X\}).$$

Zbiór $\mathcal{G}$ można zatem uznać za bezstratną reprezentację wszystkich wzorców $X \subseteq I$.

Definicja 4.6 Reprezentacją generatorową wzorców częstych *$\mathcal{F}$ nazywamy zbiór wszystkich częstych generatorów $\mathcal{FG} = \mathcal{F} \cap \mathcal{G}$ wraz z wartościami wsparć dla każdego $X \in \mathcal{FG}$ oraz rzadką granicę GBd^- zdefiniowaną jako:*

$$GBd^- = \{X \subseteq I | X \notin \mathcal{F} \wedge (\forall Y \subset X : Y \in \mathcal{F}\}.$$

Obliczanie wartości wsparcia dla dowolnego wzorca $X \in \mathcal{F}$ przebiega według następującego schematu:

- Jeżeli istnieje wzorzec $Z \in GBd^-$ taki, że $Z \subseteq X$, to wzorzec X jest rzadki.
- W przeciwnym przypadku wzorzec X jest częsty, a jego wsparcie jest równe $sup(X) = min(\{sup(Y) | Y \in \mathcal{FG} \wedge Y \subseteq X\})$.

4.3 Reprezentacje wiedzy niepełnej

Reprezentacje wiedzy niepełnej przedstawione w niniejszym punkcie muszą uwzględniać wyjątkową specyfikę wiedzy, którą mają opisywać. W szczególności z uwagi na to, że analizowany zbiór wzorców znanych $\mathcal{K}$ jest niekompletny, a także ze względu na fakt, że jak przedstawiono w punkcie 3.3.1, wzorce ograniczone mogą mieć wsparcie określone w sposób przybliżony, przyjęte wcześniej założenia i wynikające z nich własności tracą ważność. W dalszej części przedstawiono zatem nowe reprezentacje korzystające ze zmodyfikowanych definicji zbiorów zamkniętych i generatorów uwzględniające odmienne własności niepełnej wiedzy.

4.3.1 Reprezentacja wzorców ograniczonych

Zwięzła reprezentacja wzorców ograniczonych $BIS(\mathcal{K})$ została zaproponowana w [20]. Ze względu na fakt, że w przypadku wzorców ograniczonych wprowadzono pojęcie wsparcia optymistycznego i pesymistycznego, należy odpowiednio zmodyfikować definicję domknięcia zbioru 4.1.

Definicja 4.7 Domknięciem zbioru $X \in BIS(\mathcal{K})$ *w* $\mathcal{K}$ *nazywamy maksymalny znany zbiór* $Y \supseteq X$*, którego wsparcie jest równe wsparciu pesymistycznemu* X *w* $\mathcal{K}$*. Zbiór wszystkich domknięć* X *w* $\mathcal{K}$ *oznaczamy przez* $\gamma(X, \mathcal{K})$*:*

$$\gamma(X, \mathcal{K}) = MAX\{Y \in \mathcal{K} | Y \supseteq X \wedge sup(Y) = pSup(X, \mathcal{K})\}.$$

Zbiór wszystkich domknięć zbiorów w $\mathcal{K}$ oznaczamy przez $C(\mathcal{K})$:

$$C(\mathcal{K}) = \bigcup_{X \in \mathcal{K}} \gamma(X, \mathcal{K}\}.$$

Definicja 4.8 *Zbiór* $X \in \mathcal{K}$ *nazywamy* **zbiorem zamkniętym w** $\mathcal{K}$*, jeżeli* $\{X\} = \gamma(X, \mathcal{K})$.

Należy zauważyć, że w przeciwieństwie do klasycznego domknięcia zbioru rozumianego jako przecięcie wszystkich transakcji zawierających ten zbiór, domknięcie zbioru ograniczonego określone zgodnie z powyższą definicją może składać się z więcej niż jednego elementu.

W analogiczny sposób należy zaadoptować definicję generatorów zbiorów pozycji 4.4.

Definicja 4.9 Generatorem zbioru $X \in BIS(\mathcal{K})$ *w* $\mathcal{K}$ *nazywamy minimalny znany zbiór* $Y \subseteq X$*, którego wsparcie jest równe wsparciu optymistycznemu* X *w* $\mathcal{K}$*. Zbiór wszystkich generatorów* X *w* $\mathcal{K}$ *oznaczamy przez* $\mathcal{G}(X, \mathcal{K})$*:*

$$\mathcal{G}(X, \mathcal{K}) = MIN\{Y \in \mathcal{K} | Y \subseteq X \wedge sup(Y) = oSup(X, \mathcal{K})\}.$$

Zbiór wszystkich generatorów zbiorów w $\mathcal{K}$ oznaczamy przez $\mathcal{G}(\mathcal{K})$:

$$\mathcal{G}(\mathcal{K}) = \bigcup_{X \in \mathcal{K}} \mathcal{G}(X, \mathcal{K}\}.$$

Definicja 4.10 *Zbiór* $X \in \mathcal{K}$ *nazywamy* **generatorem w** $\mathcal{K}$*, jeżeli* $\{X\} = \mathcal{G}(X, \mathcal{K})$.

Podobnie jak w przypadku klasycznych generatorów i zbiorów zamkniętych, generatory i zbiory zamknięte w $\mathcal{K}$ muszą mieć inną wartość wsparcia odpowiednio od swoich znanych nadzbiorów i podzbiorów właściwych. Własność ta umożliwia skonstruowanie algorytmu wyznaczania generatorów i zbiorów zamkniętych w $\mathcal{K}$.

Można wykazać [20], że powyższe definicje zbiorów $C(\mathcal{K})$ i $\mathcal{G}(\mathcal{K})$ umożliwiają wyznaczenie pesymistycznych wsparć wzorców ograniczonych w $\mathcal{K}$ wyłącznie w oparciu o wsparcia domknięć znanych wzorców $C(\mathcal{K})$, zaś ich wsparcia optymistyczne mogą być obliczone w oparciu o wsparcia generatorów znanych wzorców $\mathcal{G}(\mathcal{K})$:

Własność 4.2 *Niech $X \in BIS(\mathcal{K})$. Wtedy:*

$$pSup(X,\mathcal{K}) = max(\{sup(Z)|Z \in C(\mathcal{K}) \wedge X \subseteq Y\}) = pSup(X, C(\mathcal{K})),$$
$$oSup(X,\mathcal{K}) = min(\{sup(Y)|Y \in \mathcal{G}(\mathcal{K}) \wedge Y \subseteq X\}) = oSup(X, \mathcal{G}(\mathcal{K})).$$

Wsparcia wzorców znanych można wyznaczyć analogicznie korzystając ze wsparć zbiorów w $C(\mathcal{K})$ lub $\mathcal{G}(\mathcal{K})$.

W wymienionej pracy udowodniono także, że każdy zbiór w $BIS(\mathcal{K})$ jest ograniczony przez podzbiór będący generatorem w $\mathcal{K}$ oraz nadzbiór będący zbiorem zamkniętym w $\mathcal{K}$. Spostrzeżenie to prowadzi do następującego twierdzenia:

Twierdzenie 4.1

$$BIS(\mathcal{K}) = \{X \subseteq I|\exists Y \in \mathcal{G}(\mathcal{K})\exists Z \in C((K)\colon Y \subseteq X \subseteq Z\},$$
$$EBIS(\mathcal{K}) = \{X \subseteq I|\exists Y \in \mathcal{G}(\mathcal{K})\exists Z \in C((K)\colon Y \subseteq X \subseteq Z$$
$$\wedge\, pSup(X,\mathcal{K}) = oSup(X,\mathcal{K})\}.$$

Przedstawione definicje i wynikające z nich własności dowodzą, że para zbiorów ($\mathcal{G}(\mathcal{K})$, $C(\mathcal{K})$) jest wystarczająca do tego, aby wyznaczyć wszystkie zbiory ograniczone i ich wsparcia optymistyczne i pesymistyczne. Tym samym ($\mathcal{G}(\mathcal{K})$, $C(\mathcal{K})$) stanowi bezstratną reprezentację wzorców ograniczonych w $\mathcal{K}$.

Definicja 4.11 Reprezentacją wzorców ograniczonych *nazywamy sumę teoriomnogościową zbiorów $\mathcal{G}(\mathcal{K})$ i $C(\mathcal{K})$ i oznaczamy przez $\mathcal{R}$:*

$$\mathcal{R}(\mathcal{K}) = \mathcal{G}(\mathcal{K}) \cup C(\mathcal{K}).$$

Wszystkie wzorce ograniczone można wyznaczyć z reprezentacji $\mathcal{R}$ korzystając z twierdzenia 4.1. Własność 4.2 opisuje natomiast sposób obliczania ich wsparć optymistycznych i pesymistycznych.

4.3.2 Reprezentacja wzorców rozszerzonych i skurczonych

Analogicznie jak w przypadku reprezentacji wzorców ograniczonych, reprezentacja wzorców rozszerzonych i skurczonych przedstawiona w [20] wykorzystuje zmodyfikowane definicje zbiorów zamkniętych i generatorów omówione w poprzednim punkcie. Przed jej formalnym zdefiniowaniem przedstawimy kilka interesujących własności.

Przede wszystkim można wykazać, że żaden wzorzec skurczony nie jest zbiorem zamkniętym, a jednocześnie żaden wzorzec rozszerzony nie jest generatorem. Własność ta prowadzi do następującego wniosku:

Twierdzenie 4.2

$$C(\mathcal{K} \cup EIS(\mathcal{K}) \cup SIS(\mathcal{K})) = C(\mathcal{K} \cup EIS(\mathcal{K})),$$
$$\mathcal{G}(\mathcal{K} \cup EIS(\mathcal{K}) \cup SIS(\mathcal{K})) = \mathcal{G}(\mathcal{K} \cup SIS(\mathcal{K})).$$

Tym samym formuła definiująca sumę teoriomnogościową $\mathcal{R}$ domknięć i generatorów wszystkich zbiorów rozszerzonych i skurczonych jest następująca:

$$\mathcal{R}(\mathcal{K} \cup EIS(\mathcal{K}) \cup SIS(\mathcal{K})) = C(\mathcal{K} \cup EIS(\mathcal{K})) \cup \mathcal{G}(\mathcal{K} \cup SIS(\mathcal{K})).$$

Co więcej, zbiór zbiorów zamkniętych i generatorów wzorców rozszerzonych i skurczonych zawiera całą informację niezbędną do wyznaczenia obu powyższych zbiorów:

Twierdzenie 4.3

$$C(\mathcal{K} \cup EIS(\mathcal{K})) = C(C(\mathcal{K}) \cup C(EIS(\mathcal{K}))),$$
$$\mathcal{G}(\mathcal{K} \cup SIS(\mathcal{K})) = \mathcal{G}(\mathcal{G}(\mathcal{K}) \cup \mathcal{G}(SIS(\mathcal{K}))).$$

Zdefiniowana wyżej suma teoriomnogościowa $\mathcal{R}$ może być zatem wyznaczona wyłącznie w oparciu o $\mathcal{R}(K)$ oraz $C(EIS(\mathcal{K}))$ i $\mathcal{G}(SIS(\mathcal{K}))$. Uwzględniając następujące spostrzeżenie:

Twierdzenie 4.4

$$C(EIS(\mathcal{K})) = C(EIS(\mathcal{R}(\mathcal{K}))),$$
$$\mathcal{G}(SIS(\mathcal{K})) = \mathcal{G}(SIS(\mathcal{R}(\mathcal{K}))),$$

możemy stwierdzić, że zbiór $\mathcal{R}(\mathcal{K} \cup EIS(\mathcal{K}) \cup SIS(\mathcal{K}))$ może być wyznaczony wyłącznie w oparciu o reprezentację $\mathcal{R}(\mathcal{K})$.

Zaprezentowanie wyżej rozumowanie umożliwia zdefiniowanie zwięzłej, bezstratnej reprezentacji wszystkich wzorców rozszerzonych i skurczonych, które można wyprowadzić z $\mathcal{K}$ przez wielokrotne wykorzystanie operatorów EIS i SIS.

Definicja 4.12 Reprezentację wszystkich wzorców rozszerzonych i skurczonych *w $\mathcal{K}$ oznaczamy przez $\mathcal{R}^*$ i definiujemy jako:*

$$\mathcal{R}^*(\mathcal{K}) = R_k(\mathcal{K}),\ gdzie:$$

- $R_1(\mathcal{K}) = \mathcal{R}(\mathcal{K})$,
- $R_n(\mathcal{K}) = \mathcal{R}(R_{n-1}(\mathcal{K}) \cup EIS(R_{n-1}(\mathcal{K})) \cup SIS(R_{n-1}(\mathcal{K})))$ *dla* $n \geqslant 2$,
- k *jest najmniejszą wartością* n *taką, że* $R_n(\mathcal{K}) = R_{n+1}(\mathcal{K})$.

Korzystając z tak zdefiniowanej reprezentacji wszystkie wzorce rozszerzone i skurczone $EBIS^*(\mathcal{K})$, a także wszystkie wzorce ograniczone $BIS^*(\mathcal{K})$ można wyznaczyć przy użyciu odpowiednio operatorów $EBIS$ i BIS zgodnie z następującą formułą:

$$EBIS^*(\mathcal{K}) = EBIS(\mathcal{R}^*(\mathcal{K})),$$
$$BIS^*(\mathcal{K}) = BIS(\mathcal{R}^*(\mathcal{K})).$$

4.3.3 Algorytm R

Algorytm `R` wyznaczania reprezentacji wzorców ograniczonych w $\mathcal{K}$ wykorzystuje omówioną własność generatorów i zbiorów zamkniętych, która mówi, że generatory i zbiory zamknięte mają inną wartość wsparcia odpowiednio od wszystkich swoich znanych nadzbiorów i podzbiorów właściwych. Wyznaczanie reprezentacji $\mathcal{R}$ jest wykonywane następująco. Z każdym wzorcem znanym związane są flagi c i g mówiące o tym, czy dany wzorzec jest odpowiednio zbiorem zamkniętym, czy generatorem. W pierwszym kroku wszystkie wzorce znane oznaczane są jako zbiory zamknięte. Następnie, dla każdego wzorca znanego Y wyszukiwane są wszystkie jego podzbiory właściwe w $\mathcal{K}$ o tym samym wsparciu. Jeżeli w $\mathcal{K}$ nie istnieje żaden wzorzec będący podzbiorem Y o tym samym wsparciu, to zgodnie z przytoczoną własnością Y jest generatorem, więc ustawiana jest jego flaga g. W przeciwnym przypadku Y nie jest generatorem, a wszystkie jego podzbiory o tym samym wsparciu nie są zbiorami zamkniętymi, więc flaga c jest im usuwana.

Po wykonaniu tej procedury dla każdego wzorca znanego Y, wszystkie wzorce znane mają prawidłowo wyznaczone wartości flag c i g. Wyznaczaną reprezentację R stanowią te wzorce, dla których flaga c lub g jest ustawiona.

Algorytm 4.1: R

input : $\mathcal{K}$ - wzorce znane
output: $\mathcal{R}$ - zwięzła reprezentacja wzorców $BIS(\mathcal{K})$

```
begin
    foreach Z ∈ K do
        Z.c ← 1;
        Z.g ← 0;
    end
    foreach Y ∈ K do
        X ← ProperSubsets-SameSup(Y, K)
        if X = {} then
            Y.g ← 1;
        else
            foreach X ∈ X do
                X.c ← 0;
            end
        end
    end
    R ← {Z ∈ K | Z.c = 1 ∨ Z.g = 1}
end
```

4.3.4 Algorytm R*

Iteracyjny schemat algorytmu `R*` wyznaczania reprezentacji wszystkich wzorców dokładnie ograniczonych w $\mathcal{K}$ jest taki sam jak schemat algorytmu `EBIS*` omówionego w punkcie 3.4.5. Jedyną różnicą jest wykorzystanie operatora R zamiast operatora $EBIS$. Korzystając z tych samych obserwacji jak w przypadku algorytmu `EBIS*`, algorytmy `EIS` i `SIS` zostały połączone w jeden algorytm `EISAndSIS`. Ponadto $R_{k-1} \subseteq$ `EIS(`R_{k-1}`)` oraz $R_{k-1} \subseteq$ `SIS(`R_{k-1}`)`. Te dwa spostrzeżenia umożliwiają skonstruowanie algorytmu `R*` w uproszczonej względem definicji 4.12 postaci.

Algorytm 4.2: R*

input : $\mathcal{K}$ - wzorce znane
output: $\mathcal{R}^*$ - zwięzła reprezentacja wzorców $EBIS^*(\mathcal{K})$

begin
  $R_1 \leftarrow \texttt{R}(\mathcal{K})$
  $k \leftarrow 1$
  repeat
    $k \leftarrow k + 1$
    $R_k \leftarrow \texttt{R(EISAndSIS}(R_{k-1}))$
  until $R_k \neq R_{k-1}$
  $\mathcal{R}^* \leftarrow R_k$
end

Rozdział 5

Warstwy jako reprezentacje wzorców

5.1 Pojęcia podstawowe i wybrane warstwy wzorców

Definicje przedstawione w niniejszym punkcie zostały zaczerpnięte z [21].

Zgodnie z konwencją przyjętą w rozdziale 2, przez $I = \{i_1, i_2, ..., i_m\}$ oznaczmy niepusty skończony zbiór wszystkich pozycji (*atomów*). Zbiór ten będziemy nazywali *uniwersum*.

Definicja 5.1 Kratę *rozpiętą między zbiorami X i Y, $X \subseteq Y \subseteq I$, oznaczamy przez $[X, Y]$ i definiujemy jako rodzinę $\mathcal{L}$ wszystkich zbiorów Z takich, że:*

$$\forall Z \in \mathcal{L} \colon X \subseteq Z \subseteq Y.$$

Kratę $[X, Y]$ stanowią zatem wszystkie podzbiory Y, które są jednocześnie nadzbiorami X. Zauważmy, że krata $[\emptyset, I]$ zawiera wszystkie wzorce możliwe do wyprowadzenia dla bazy danych o uniwersum I.

Definicja 5.2 *Rodzinę zbiorów $\mathcal{X} \subseteq 2^I$ nazywamy* **antyłańcuchem** *wtedy i tylko wtedy, gdy:*

$$\forall X_1, X_2 \in \mathcal{X} \colon X_1 \neq X_2 \Rightarrow \neg X_1 \subseteq X_2 \wedge \neg X_2 \subseteq X_1.$$

Zgodnie z podaną definicją przez antyłańcuch rozumiemy każdą rodzinę zbiorów taką, że dla dowolnej pary różnych zbiorów w tej rodzinie nie zachodzi relacja zawierania. Wniosek ten umożliwia skonstruowanie prostego algorytmu przekształcającego w sposób stratny dowolną rodzinę zbiorów

w antyłańuch, polegającego na wybraniu wyłącznie zbiorów maksymalnych lub minimalnych i odrzuceniu wszystkich pozostałych zbiorów.

Definicja 5.3 Warstwą *rozpiętą między rodzinami zbiorów* $\mathcal{X}$ *i* $\mathcal{Y}$, $\mathcal{X}, \mathcal{Y} \subseteq 2^I$, *oznaczamy przez* $[\mathcal{X}, \mathcal{Y}]$ *i definiujemy jako rodzinę wszystkich zbiorów* Z *takich, że:*

$$\exists X \in \mathcal{X}\ \exists Y \in \mathcal{Y}\colon X \subseteq Z \subseteq Y.$$

Rodzinę zbiorów $\mathcal{X}$ nazywamy w takim przypadku *dolną granicą warstwy*, natomiast rodzinę zbiorów $\mathcal{Y}$ *górną granicą warstwy*. Granice $\mathcal{X}$ i $\mathcal{Y}$ nazywamy*granicami właściwymi* warstwy $[\mathcal{X}, \mathcal{Y}]$ wtedy, gdy:

- $\mathcal{X}$ i $\mathcal{Y}$ są antyłańcuchami,
- $\forall X \in \mathcal{X}\ \exists Y \in \mathcal{Y}\colon X \subseteq Y$,
- $\forall Y \in \mathcal{Y}\ \exists X \in \mathcal{X}\colon Y \supseteq X$,

Na warstwę rozpiętą między granicami właściwymi składają się zatem wszystkie wzorce X, dla których istnieją przynajmniej jeden podzbiór w granicy dolnej $\mathcal{X}$ oraz przynajmniej jeden nadzbiór w granicy górnej $\mathcal{Y}$. W niniejszej pracy rozważane są wyłącznie warstwy o granicach właściwych.

Zauważmy, że przyjmując powyższe definicje krata może być rozpatrywana jako szczególny przypadek warstwy, a mianowicie taki, w którym granicami górną i dolną są jednoelementowe rodziny zbiorów: $\mathcal{X} = \{X\}$ oraz $\mathcal{Y} = \{Y\}$. Istnieją jeszcze dwa interesujące szczególne przypadki warstw: warstwy przywiązane do podstawy oraz warstwy przywiązane do szczytu.

Definicja 5.4 Warstwą przywiązaną do podstawy *nazywamy warstwę* $[\{X\}, \mathcal{Y}]$, *której dolnym ograniczeniem jest antyłańcuch będący jednoelementową rodziną zbiorów* $\{X\}$. *Analogicznie przez* **warstwę przywiązaną do szczytu** *rozumiemy warstwę* $[\mathcal{X}, \{Y\}]$, *której górnym ograniczeniem jest antyłańcuch będący jednoelementową rodziną zbiorów* $\{Y\}$.

5.2 Wybrane warstwy wzorców

Wprowadzone w poprzednim punkcie definicje umożliwiają opisanie wybranych klas wzorców z wykorzystaniem pojęcia warstwy. Spośród omówionych we wcześniejszych rozdziałach klas wzorców warto wymienić:

- wszystkie możliwe wzorce: $[\{\emptyset\}, \{I\}]$,
- wzorce częste: $[\{\emptyset\}, MAX(\mathcal{FC})]$,

- częste generatory: $[\{\emptyset\}, MAX(\mathcal{FG})]$,
- wzorce rzadkie: $[GBd^{-}, \{I\}]$,
- wzorce ograniczone w $\mathcal{K}$: $[MIN(\mathcal{R}(\mathcal{K})), MAX(\mathcal{R}(\mathcal{K}))]$,
- wszystkie wzorce ograniczone w $\mathcal{K}$: $[MIN(\mathcal{R}^{*}(\mathcal{K})), MAX(\mathcal{R}^{*}(\mathcal{K}))]$.

Powyższy zapis może być więc traktowany jako szczególny rodzaj reprezentacji wzorców, umożliwiający wyprowadzenie wzorców określonej klasy bez informacji o wartościach ich wsparć.

Zdefiniowanie klasy wzorców z wykorzystaniem pojęcia warstwy umożliwia skorzystanie z algorytmów wyznaczających liczności warstw do obliczania liczby wzorców interesującej klasy. W wielu sytuacjach informacja o liczbie wzorców spełniających określone kryteria jest wystarczająca. Jednym z przykładów może być ocena zwięzłości reprezentacji wzorców częstych opartej na zbiorach zamkniętych. Zakładając, że dysponujemy zbiorem częstych wzorców zamkniętych $\mathcal{FC}$ oraz maksymalnych częstych zbiorów zamkniętych $MAX(\mathcal{FC})$, do oceny zwięzłości reprezentacji $\mathcal{FC}$ nie jest konieczne wyznaczanie wszystkich wzorców częstych. Wystarczy obliczenie liczności warstwy $[\{\emptyset\}, MAX(\mathcal{FC})]$ oraz określenie stosunku $\frac{|\mathcal{FC}|}{|[\{\emptyset\}, MAX(\mathcal{FC})]|}$. W przypadkach, w których czas wyznaczania wszystkich wzorców częstych jest nieakceptowalny, rozwiązanie to może być jedynym możliwym. W analogiczny sposób można zweryfikować zwięzłość reprezentacji generatorowej.

Problemem komplementarnym do powyższego jest obliczanie liczby wzorców rzadkich, na bazie których konstruuje się rzadkie reguły asocjacyjne. Wyszukiwanie reguł tego typu jest zadaniem trudnym ze względu na ogromną liczbę wzorców rzadkich. Jednocześnie reguły rzadkie o dużym zaufaniu mogą być szczególnie interesujące, ponieważ opisują nietypowe zależności o dużym stopniu korelacji. Przykładem takiej zależności mogą być charakterystyczne symptomy wyjątkowo rzadkiej choroby. Wyznaczenie wszystkich wzorców rzadkich zazwyczaj nie jest praktycznie możliwe, niemniej ich liczba może być interesująca. Przy założeniu, że dysponujemy zbiorem minimalnych wzorców rzadkich GBd^{-} i wzorcem I będącym uniwersum, liczba ta jest równa liczności warstwy $[GBd^{-}, \{I\}]$.

Innym zastosowaniem algorytmów obliczających liczność warstw jest wyznaczenie liczby wszystkich wzorców, które można wyprowadzić korzystając z metod wnioskowania z niepełnej wiedzy z pewnego zbioru wzorców znanych. Wystarczy w tym celu wyznaczyć reprezentację $R^{*}(\mathcal{K})$, a następnie obliczyć liczność warstwy $[MIN(\mathcal{R}^{*}(\mathcal{K})), MAX(\mathcal{R}^{*}(\mathcal{K}))]$, która jest równa liczbie wszystkich wzorców ograniczonych w $\mathcal{K}$.

Omówione dalej algorytmy, zaproponowane w [22], umożliwiają wyznaczenie liczby wzorców dowolnej klasy, którą można zdefiniować przy pomocy przedstawionych we wstępie pojęć.

5.3 Wyznaczanie warstw i ich liczności

Podczas prezentacji algorytmów wyznaczania warstw i ich liczności przyjęto założenie, że granice danych warstw są granicami właściwymi.

5.3.1 Algorytmy wyznaczania krat i ich liczności

Funkcja `LatticeCardinality` wyznacza liczność kraty rozpiętej między dwoma wzorcami X i Y, $X \subseteq Y$. Liczność ta jest równa k-tej potędze liczby 2, gdzie k jest mocą zbioru będącego różnicą wzorców $Y \setminus X$.

Funkcja 5.1: LatticeCardinality(X, Y)

assert: $X \subseteq Y$

begin
  $k \leftarrow |Y \setminus X|$
  return 2^k
end

Funkcja `Lattice` generuje wszystkie zbiory należące do kraty $[X, Y]$. Warto zauważyć, że podobną funkcjonalność realizuje funkcja `NonEmptyProperSubsets` będącej częścią algorytmu `BIS` przedstawionego w punkcie 3.3.2. Rzeczą, która odróżnia `NonEmptyProperSubsets` od `Lattice` jest fakt, że ta pierwsza przyjmuje jako argument wywołania tylko jeden wzorzec będący górnym ograniczeniem kraty zakładając jednocześnie, że dolnym ograniczeniem jest zbiór pusty. Ponadto `NonEmptyProperSubsets` zgodnie ze swoim przeznaczeniem w wyniku nie zwraca wzorców będących dolnym (zbiór pusty) i górnym (argument wywołania) ograniczeniem kraty. Tym samym między `Lattice` a `NonEmptyProperSubsets` zachodzi następująca zależność:

$$\texttt{NonEmptyProperSubsets}(Y) = \texttt{Lattice}(\emptyset, Y) \setminus \{\emptyset, Y\}.$$

Sposób generowania wzorców przez funkcję `Lattice` jest analogiczny do metody realizowanej przez funkcję `NonEmptyProperSubsets`, dlatego nie będzie tutaj ponownie omawiany.

Funkcja 5.2: Lattice(X, Y)

```
assert: X ⊆ Y
begin
    L_{|X|} ← {X}
    L_{|X|+1} ← {X ∪ {a} | a ∈ Y \ X}
    for k ← |X| + 1 to |Y| − 1 do
        foreach F, G ∈ L_k do
            if F[1] = G[1] ∧ ... ∧ F[k − 1] = G[k − 1] ∧ F[k] < G[k]
            then
                Z ← F[1] · ... · F[k] · G[k]
                L_{k+1} ← L_{k+1} ∪ {Z}
            end
        end
    end
    return ⋃ L_i
end
```

5.3.2 Algorytmy wyznaczania warstw przywiązanych do podstawy i ich liczności

Wyznaczanie liczności warstw przywiązanych do podstawy

Pierwszy przedstawiony algorytm wyznaczania liczności warstwy przywiązanej do podstawy realizuje funkcja `BotStratumCardinality`. Algorytm ten jest iteracyjno-rekurencyjny. Na ustalonym poziomie rekurencji, w i-tym kroku wyznaczana jest liczność warstwy $[\{X\}, \{Y_1, \ldots, Y_i\}]$. Zakładając, że liczność warstwy $[\{X\}, \{Y_1, \ldots, Y_{i-1}\}]$ została określona w kroku poprzednim, algorytm wyznacza liczność kraty $[X, Y_i]$ oraz liczność warstwy będącej częścią wspólną kraty $[X, Y_i]$ i warstwy $[\{X\}, \{Y_1, \ldots, Y_{i-1}\}]$. Po wyznaczeniu tych wartości liczność warstwy $[\{X\}, \{Y_1, \ldots, Y_{i-1}\}]$ określana jest zgodnie ze wzorem:

$$|[\{X\}, \{Y_1, \ldots, Y_i\}]| = |[\{X\}, \{Y_1, \ldots, Y_{i-1}\}]| + |[X, Y_i]| - |[\{X\}, \{Y_1, \ldots, Y_{i-1}\}] \cap [X, Y_i]|.$$

Liczność kraty $[X, Y_i]$ obliczana jest przy użyciu funkcji `LatticeCardinality`. W celu wyznaczenia liczności warstwy $|[\{X\}, \{Y_1, \ldots, Y_{i-1}\}] \cap [X, Y_i]|$ wystarczy zauważyć, że:

$$[\{X\}, \{Y_1, \ldots, Y_{i-1}\}] \cap [X, Y_i] = [X, MAX\{Y_i \cap Y_j | j = 1, \ldots, i-1\}].$$

Zdefiniowane w ten sposób granice części wspólnej warstwy i kraty są właściwie, a więc możliwe jest wyznaczenie liczności nowej warstwy przez rekurencyjne wywołanie funkcji `BotStratumCardinality`.

Algorytm kończy się po wykonaniu n kroków obliczeń na pierwszym poziomie rekurencji, gdzie n to liczba wzorców należących do górnej granicy warstwy będącej argumentem wywołania. Warunkiem stopu dla rekurencji jest natomiast stan, w którym górna granica warstwy jest zbiorem jednoelementowym. Warunek ten będzie zawsze osiągnięty, ponieważ w każdym wywołaniu rekurencyjnym górna granica warstwy ma nie więcej elementów niż liczba elementów górnej granicy warstwy na wcześniejszym poziomie rekurencji pomniejszona o jeden.

Funkcja 5.3: BotStratumCardinality($\{X\}, \{Y_1, \ldots, Y_n\}$)

assert: $\{Y_1, \ldots, Y_n\}$ jest niepustym antyłańcuchem składającym się wyłącznie z nadzbiorów X

begin
  $card \leftarrow$ `LatticeCardinality`(X, Y_1)
  for $i \leftarrow 2$ **to** n **do**
    $\mathcal{Y} \leftarrow MAX(\{Y_i \cap Y_j | j = 1, \ldots, i-1\})f$
    $card \leftarrow card +$ `LatticeCardinality`(X, Y_i) $-$ `BotStratumCardinality`$(\{X\}, \mathcal{Y})$
  end
  return $card$
end

Alternatywny algorytm wyznaczania liczności warstwy przywiązanej do podstawy implementuje funkcja `EnumBotStratumCardinality`, która jako dolną granicę warstwy przyjmuje zawsze zbiór pusty[1]. Idea przyjętej metody polega na podziale warstwy na dwie rozłączne warstwy, a następnie rekurencyjnym wyznaczeniu ich liczności i zsumowaniu uzyskanych wyników. W omawianym algorytmie przyjęto, że pierwszą warstwę stanowią wzorce zawierające pewną arbitralnie wybraną pozycję a, natomiast drugą wszystkie wzorce niezawierające tej pozycji. Problem sprowadza się zatem do wyznaczenia liczby wzorców należących do obu grup.

W celu podziału warstwy $[\{\emptyset\}, \mathcal{Y}]$ na dwie rozłączne warstwy i obliczenia ich liczności, dla obu warstw algorytm wyznacza górne i dolne ograniczenie.

[1] W celu obliczenia liczności dowolnej warstwy przywiązanej do podstawy wystarczy zauważyć, że $|[\{X\}, \mathcal{Y}]| = |[\{\emptyset\}, \{Y \setminus X | Y \in \mathcal{Y}\}]|$. Z własności tej korzysta omawiany algorytm `EnumBotStratumCardinality`.

Warstwa pierwsza składa się ze wszystkich wzorców zawierających pozycję a, a tym samym zdefiniowana jest następująco: $[\{a\}, \{Y|Y \in \mathcal{Y} \wedge a \in Y\}]$. Jak łatwo zauważyć, określone w ten sposób granice warstwy są właściwe. Druga warstwa składa się ze wszystkich wzorców niezawierających pozycji a. Jej granice są zatem następujące: $[\{\emptyset\}, MAX(\{Y|Y \in \mathcal{Y} \wedge a \notin Y\} \cup \{Y \setminus \{a\}|Y \in \mathcal{Y} \wedge a \in Y\})]$. Ze względu na to, że górna granica warstwy powstaje w wyniku zsumowania zbioru wzorców niezawierających a ze wzorcami zawierającymi a, z których a usunięto, te drugie mogą nie być maksymalne. Ograniczenie warstwy musi być jednak właściwe, dlatego konieczne jest zastosowanie operatora MAX.

Po takim zdefiniowaniu dwóch warstw rozłącznych, algorytm wyznacza rekurencyjnie ich liczność. Warunkiem stopu dla rekurencji jest stan, w którym górna granica warstwy jest zbiorem jednoelementowym. Warunek ten jest zawsze osiągany przy przedstawionym algorytmie podziału warstwy.

Funkcja 5.4: EnumBotStratumCardinality($\{\emptyset\}, \mathcal{Y}$)

assert: $\mathcal{Y}$ jest niepustym antyłańcuchem
assert: zbiory w $\mathcal{Y}$ są uporządkowane leksykograficznie

begin
  if $|\mathcal{Y}| = 1$ **then**
    $n = |Y|$, gdzie Y to jedyny zbiór w $\mathcal{Y}$
    return 2^n
  end
  else
    $a \leftarrow$ pierwszy element pierwszego zbioru w $\mathcal{Y}$
    $\mathcal{Y}_1 \leftarrow$ {wszystkie zbiory w $\mathcal{Y}$ zaczynające się od elementu a}
    $\mathcal{Y}_2 = \mathcal{Y} \setminus \mathcal{Y}_1$
    usuń element a ze wszystkich zbiorów w $\mathcal{Y}_1$
    $\mathcal{Y}_2 = MAX(\mathcal{Y}_2 \cup \mathcal{Y}_1)$
    return `EnumBotStratumCardinality(`$\{\emptyset\}, \mathcal{Y}_1$`)` +
    `EnumBotStratumCardinality(`$\{\emptyset\}, \mathcal{Y}_2$`)`
  end
end

W algorytmie 5.4 przyjęto ponadto następujące założenie implementacyjne wpływające pozytywnie na jego efektywność: wzorce należące do górnej granicy warstwy muszą być uporządkowane leksykograficznie. Dzięki temu, jeśli pozycja a będzie zawsze wybierana jako pierwsza pozycja pierwszego wzorca górnej granicy, to podział górnej granicy $\mathcal{Y}$ na dwie części $\mathcal{Y}_1$ i $\mathcal{Y}_2$ będzie operacją o złożoności liniowej względem wielkości $\mathcal{Y}$.

Wyznaczanie warstw przywiązanych do podstawy

Funkcja 5.5: BotStratum($\{\emptyset\}, \mathcal{Y}$)

assert: $\mathcal{Y}$ jest antyłańcuchem

begin
  if $\mathcal{Y} \neq \{\}$ **then**
    $result \leftarrow \mathcal{Y}$
    loop begin
      $Y \leftarrow$ dowolny najdłuższy nierozpatrywany zbiór z $result$
      if $Y = \emptyset$ **then return** $result$
      else $result \leftarrow result \cup \{Y \setminus \{a\} | a \in Y\}$
    end
  end
end

Funkcja 5.6: LatticeBotStratum($\{\emptyset\}, \mathcal{Y}$)

assert: $\mathcal{Y}$ jest antyłańcuchem

begin
  $result \leftarrow \{\}$
  foreach $Y \in \mathcal{Y}$ **do**
    $result \leftarrow result \cup$ `Lattice(`$\emptyset, Y$`)`
  end
  return $result$
end

5.3.3 Algorytmy wyznaczania warstw przywiązanych do szczytu i ich liczności

Wyznaczanie liczności warstw przywiązanych do szczytu

Algorytm wyznaczania liczności warstwy przywiązanej do szczytu realizowany przez funkcję `TopStratumCardinality` jest analogiczny do algorytmu implementowanego przez funkcję `BotStratumCardinality` przedstawionego w poprzednim punkcie. Podobnie jak w przypadku `TopStratumCardinality`, w i-tym kroku wyznaczana jest liczność warstwy

$[\{X_1, \ldots, X_i\}, \{Y\}]$ według następującej formuły:

$$|[\{X_1, \ldots, X_i\}, \{Y\}]| = |[\{X_1, \ldots, X_{i-1}\}, \{Y\}]| + |[X_i, Y]| - |[\{X_1, \ldots, X_{i-1}\}, \{Y\}] \cap [X_i, Y]|.$$

Warstwa $[\{X_1, \ldots, X_{i-1}\}, \{Y\}] \cap [X_i, Y]$ jest natomiast zdefiniowana następująco:

$$[\{X_1, \ldots, X_{i-1}\}, \{Y\}] \cap [X_i, Y] = [MIN\{X_i \cup X_j | j = 1, \ldots, i-1\}, Y].$$

Wszystkie pozostałe cechy algorytmu `TopStratumCardinality` są analogiczne do algorytmu `BotStratumCardinality`.

Funkcja 5.7: TopStratumCardinality($\{X_1, \ldots, X_n\}, \{Y\}$)

assert: $\{X_1, \ldots, X_n\}$ jest niepustym antyłańcuchem składającym się wyłącznie z podzbiorów Y

begin
  $card \leftarrow$ `LatticeCardinality(`X_1, Y`)`
  for $i \leftarrow 2$ **to** n **do**
    $\mathcal{X} \leftarrow MIN(\{X_i \cup X_j | j = 1, \ldots, i-1\})$
    $card \leftarrow card +$ `LatticeCardinality(`X_i, Y`)` $-$
    `TopStratumCardinality(`$\mathcal{X}, \{Y\}$`)`
  end
  return *card*
end

Alternatywna funkcja `EnumTopStratumCardinality` wyznacza liczność warstwy wykorzystując przedstawioną w poprzednim punkcie funkcję `EnumBotStratumCardinality`. W tym celu dana warstwa przywiązana do szczytu poddawana jest transformacji, w wyniku której powstaje nowa warstwa przywiązana do podstawy. Ponieważ przyjęta metoda przekształcenia zachowuje liczność warstwy, wartością zwracaną przez funkcję `EnumTopStratumCardinality` jest liczność nowej warstwy wyznaczona funkcją `EnumBotStratumCardinality`.

Funkcja 5.8: EnumTopStratumCardinality($\mathcal{X}, \{Y\}$)

assert: $\mathcal{X}$ jest niepustym antyłańcuchem składającym się wyłącznie z podzbiorów Y

begin
  $\mathcal{Y} \leftarrow \{Y \setminus X | X \in \mathcal{X}\}$
  uporządkuj zbiory w $\mathcal{Y}$ leksykograficznie
  return `EnumBotStratumCardinality(`$\{\emptyset\}, \mathcal{Y}$`)`
end

Wyznaczanie warstw przywiązanych do szczytu

Funkcja 5.9: TopStratum($\mathcal{X}, \{Y\}$)

assert: $\mathcal{X}$ jest antyłańcuchem składającym się wyłącznie z podzbiorów Y

begin
  if $\mathcal{X} \neq \{\}$ **then**
    $result \leftarrow \mathcal{X}$
    loop begin
      $X \leftarrow$ dowolny najkrótszy nierozpatrywany zbiór z $result$
      if $Y \setminus X = \emptyset$ **then return** $result$
      else $result \leftarrow result \cup \{X \cup \{a\} | a \in Y \setminus X\}$
    end
  end
end

Funkcja 5.10: LatticeTopStratum($\mathcal{X}, \{Y\}$)

assert: $\mathcal{X}$ jest antyłańcuchem składającym się wyłącznie z podzbiorów Y

begin
  $result \leftarrow \{\}$
  foreach $X \in \mathcal{X}$ **do**
    $result \leftarrow result \cup$ `Lattice(`X, Y`)`
  end
  return $result$
end

5.3.4 Algorytmy wyznaczania dowolnych warstw i ich liczności

Wyznaczanie liczności dowolnych warstw

Idea algorytmu wyznaczania liczności dowolnej warstwy $[\mathcal{X}, \mathcal{Y}]$ implementowanego przez funkcję `StratumCardinality-BotOriented` jest następująca. W pierwszym kroku obliczana jest liczność warstwy $[\{\emptyset\}, \mathcal{Y}]$ przy użyciu funkcji `BotStratumCardinality`. Warstwa ta uwzględnia wzorce, które są podzbiorami $\mathcal{Y}$, ale nie są nadzbiorami $\mathcal{X}$, a więc nienależące do $[\mathcal{X}, \mathcal{Y}]$. Z tego względu algorytm wyznacza granicę górną $\mathcal{Z}$ warstwy nadmiarowej, którą są wszystkie maksymalne podzbiory $\mathcal{Y}$ nieposiadające podzbiorów w $\mathcal{X}$. Następnie liczność warstwy nadmiarowej $[\{\emptyset\}, \mathcal{Z}]$ jest wyznaczana funkcją `BotStratumCardinality`, zaś poszukiwana wartość warstwy $[\mathcal{X}, \mathcal{Y}]$ wynosi:

$$|[\mathcal{X}, \mathcal{Y}]| = |[\{\emptyset\}, \mathcal{Y}]| - |[\{\emptyset\}, \mathcal{Z}]|.$$

Funkcja 5.11: StratumCardinality-BotOriented($\mathcal{X}, \mathcal{Y}$)

assert: $\mathcal{X}$ jest antyłańcuchem, którego wszystkie elementy mają nadzbiory w $\mathcal{Y}$
assert: $\mathcal{Y}$ jest antyłańcuchem, którego wszystkie elementy mają podzbiory w $\mathcal{X}$

```
begin
    if X = {∅} then
        return BotStratumCardinality({∅}, Y)
    end
    else
        Z ← MaxΔ(X, Y)
        return BotStratumCardinality({∅}, Y) −
        BotStratumCardinality({∅}, Z)
    end
end
```

Obliczanie górnej granicy warstwy nadmiarowej wykonywane jest przy pomocy funkcji `MaxΔ`$(\mathcal{X}, \mathcal{Y})$ przedstawionej poniżej. Funkcja ta dla każdego wzorca $Y \in \mathcal{Y}$ wyznacza wszystkie jego maksymalne podzbiory niebędące nadzbiorami wzorców z $\mathcal{X}$. Następnie z tak wygenerowanych wzorców wybierane są wzorce maksymalne.

Naiwny algorytm wyszukiwania maksymalnych podzbiorów wzorca Y niebędących nadzbiorem wzorców z $\mathcal{X}$ realizuje funkcja `NaiveMaxΔ`$(\mathcal{X}, Y)$.

Funkcja 5.12: $\mathrm{Max}\Delta(\mathcal{X}, \mathcal{Y})$

assert: $\mathcal{X}$ jest antyłańcuchem, którego wszystkie elementy mają nadzbiory w $\mathcal{Y}$

assert: $\mathcal{Y}$ jest antyłańcuchem, którego wszystkie elementy mają podzbiory w $\mathcal{X}$

begin

  foreach $Y \in \mathcal{Y}$ **do**

    $\mathcal{X}_Y \leftarrow \{X \in \mathcal{X} | X \subseteq Y\}$

    $\mathcal{Z}_Y \leftarrow \texttt{Max}\Delta(\mathcal{X}_Y, Y)$

    $\mathcal{Z} \leftarrow \mathcal{Z} \cup \mathcal{Z}_Y$

  end

  $\mathcal{Z} \leftarrow MAX(\mathcal{Z})$

  return $\mathcal{Z}$

end

Tworzy ona kolejne podzbiory Y poprzez usunięcie z Y jednej pozycji z każdego wzorca $X \in \mathcal{X}$. Nowo powstały wzorzec nie będzie więc nadzbiorem żadnego wzorca X. Ostatecznie funkcja zwraca wzorce maksymalne spośród wszystkich wygenerowanych wzorców.

Funkcja 5.13: $\mathrm{NaiveMax}\Delta(\mathcal{X}, Y)$

assert: $\mathcal{X}$ jest antyłańcuchem, którego wszystkie elementy są podzbiorami Y

begin

  $\mathcal{Z} \leftarrow MAX(\{Y \setminus \{x_1, \ldots, x_n\} | x_i \in X_i \in \mathcal{X}, i = 1, \ldots, |\mathcal{X}|\})$

  return $\mathcal{Z}$

end

Efektywniejszy, rekurencyjny algorytm implementowany przez funkcję $\texttt{Max}\Delta(\mathcal{X}, Y)$ zaproponowano w [19]. W pierwszym kroku wykonywana jest redukcja argumentów wykorzystująca następującą obserwację. Każdy niepusty podzbiór Y zawierający pozycję x taką, że $\{x\} \in \mathcal{X}$, jest nadzbiorem $\{x\} \in \mathcal{X}$, a więc nie należy do $\texttt{Max}\Delta(\mathcal{X}, Y)$. Poszukiwane podzbiory Y nie mogą więc zawierać pozycji, które należą do dowolnego wzorca jednoelementowego $X \in \mathcal{X}$. W związku z tym ze zbioru Y usuwane są wszystkie takie pozycje. Jednocześnie ze zbioru $\mathcal{X}$ usuwane są wszystkie wzorce o niepustej części wspólnej z dowolnym wzorcem jednoelementowym $X \in \mathcal{X}$, ponieważ zredukowany wzorzec Y nie jest nadzbiorem żadnego z nich.

Funkcja 5.14: $\mathrm{Max}\Delta(\mathcal{X}, Y)$

assert: $\mathcal{X}$ jest antyłańcuchem, którego wszystkie elementy są podzbiorami Y

begin
- $Z \leftarrow \bigcup\{X \in \mathcal{X} || X| = 1\}$
- $Y \leftarrow Y \setminus Z$
- $\mathcal{X} \leftarrow \{X \in \mathcal{X} | Z \cap X = \emptyset\}$
- **if** $\mathcal{X} = \{\}$ **then**
  - $Max\Delta = \{Y\}$
- **end**
- **else**
  - $a \leftarrow$ dowolny element w $\bigcup \mathcal{X}$
  - $Max\Delta_{-a} \leftarrow$`Max`$\Delta$`(`$\{X \in \mathcal{X} | \{a\} \cap X = \emptyset\}, Y \setminus \{a\}$`)`
  - $Max\Delta_{a} \leftarrow$`Max`Δ`(`$\{X \setminus \{a\} | X \in \mathcal{X}\}, Y$`)`
  - **foreach** $Z \in Max\Delta_a$ **do**
    - $\mathcal{V} \leftarrow \{Z' \in Max\Delta_{-a} | Z' \subset Z\}$
    - $Max\Delta_{-a} \leftarrow Max\Delta_{-a} \setminus \mathcal{V}$
  - **end**
  - $Max\Delta = Max\Delta_{-a} \cup Max\Delta_a$
- **end**
- **return** $Max\Delta$

end

Jeżeli po wykonaniu tych operacji zbiór $\mathcal{X}$ jest pusty, wynikiem działania algorytmu jest zredukowany wzorzec Y. W przeciwnym przypadku wybierana jest dowolna pozycja z dowolnego wzorca $X \in \mathcal{X}$, a następnie problem wyszukiwania podzbiorów Y dzielony jest na dwa mniejsze problemy rozwiązywane rekurencyjnie. Wzorce $Max\Delta_{-a}$ będące wynikiem pierwszego wywołania rekurencyjnego nie zawierają pozycji a, natomiast wzorce $Max\Delta_a$ zwrócone przez drugie wywołanie rekurencyjne zawierają ją. Z tego względu ze zbioru $Max\Delta_{-a}$ usuwane są wzorce, dla których istnieją nadzbiory w $Max\Delta_a$. Wynikiem działania algorytmu jest suma teoriomnogościowa zredukowanego zbioru $Max\Delta_{-a}$ i zbioru $Max\Delta_a$.

W pracy zbadano także własności algorytmu `ExpansionMax`Δ opisanego jako *Algorytm 3.2.1: FindMinima* w [13].

Wyznaczanie dowolnych warstw

Funkcja 5.15: Stratum-BotOriented($\mathcal{X}, \mathcal{Y}$)

assert: $\mathcal{X}$ jest antyłańcuchem, którego wszystkie elementy mają nadzbiory w $\mathcal{Y}$

assert: $\mathcal{Y}$ jest antyłańcuchem, którego wszystkie elementy mają podzbiory w $\mathcal{X}$

begin
  if $\mathcal{X} = \{\emptyset\}$ **then**
    return `BotStratum`$(\{\emptyset\}, \mathcal{Y})$
  end
  else
    $\mathcal{Z} \leftarrow$ `Max`$\Delta(\mathcal{X}, \mathcal{Y})$
    $\mathcal{S}1 \leftarrow$ `BotStratum`$(\{\emptyset\}, \mathcal{Y})$
    $\mathcal{S}2 \leftarrow$ `BotStratum`$(\{\emptyset\}, \mathcal{Z})$
    return $\mathcal{S}1 \setminus \mathcal{S}2$
  end
end

Funkcja 5.16: LatticeStratum($\mathcal{X}, \mathcal{Y}$)

assert: $\mathcal{X}$ jest antyłańcuchem, którego wszystkie elementy mają nadzbiory w $\mathcal{Y}$

assert: $\mathcal{Y}$ jest antyłańcuchem, którego wszystkie elementy mają podzbiory w $\mathcal{X}$

begin
  $result \leftarrow \{\}$
  foreach $Y \in \mathcal{Y}$ **do**
    $\mathcal{Z} \leftarrow \{X \in \mathcal{X} | X \subseteq Y\}$
    foreach $X \in \mathcal{Z}$ **do**
      $result \leftarrow result \cup$ `Lattice(`X, Y`)`
    end
  end
  return *result*
end

Rozdział 6

Zrealizowane oprogramowanie analizy danych

W bieżącym rozdziale przedstawiono zaprojektowane i stworzone przez autora pracy oprogramowanie *$\mathcal{K}$-miner*. Oprogramowanie to jest przeznaczone dla systemu operacyjnego Microsoft Windows i stanowi niezależny, uniwersalny i łatwo rozszerzalny system analizy danych z graficznym interfejsem użytkownika, umożliwiający dołączanie dowolnych algorytmów w postaci zewnętrznych bibliotek komunikujących się z aplikacją przez ściśle określony interfejs API (ang. *application programming interface*).

Autor zaimplementował także i dołączył do oprogramowania *$\mathcal{K}$-miner* wszystkie algorytmy omówione w poprzednich rozdziałach, a także szereg procedur pomocniczych służących do wstępnego przetwarzania i obróbki zbiorów danych. Wykaz wszystkich zaimplementowanych procedur zamieszczono w dodatku A.

6.1 Założenia projektowe

Architektura

Ogólna architektura oprogramowania została zaczerpnięta ze standardu Java Data Mining 1.0 [11]. Uzyskano dzięki temu bardzo wysoki stopień odseparowania poszczególnych modułów, na które składają się: aplikacja główna, moduł danych, moduł procedur abstrakcyjnych, moduł algorytmów elementarnych, moduł procedur pomocniczych oraz niezależne biblioteki dynamiczne implementujące poszczególne algorytmy. Warto zauważyć, że aplikacja

K-miner nie posiada żadnej szczególnej wiedzy o udostępnianych użytkownikowi algorytmach, ponieważ wszystkie niezbędne metadane przekazywane są do niej w abstrakcyjnej postaci w trakcie automatycznego dołączania bibliotek. Tym samym możliwe jest dodanie do aplikacji kolejnych, niezależnie implementowanych algorytmów. Jednocześnie aplikacja dostarcza szereg usług w postaci bibliotek statycznych, takich jak zarządzanie danymi czy metody komunikacji.

Wykorzystane technologie

Oprogramowaniem *K-miner* zostało napisane w całości w języku C++ w środowisku Microsoft Visual Studio. W celu efektywnej realizacji operacji przetwarzania danych, takich jak wczytywanie, zapisywanie, implementacja algorytmów i funkcji pomocniczych, wykorzystywana jest standardowa biblioteka szablonów STL. Do implementacji elementów interfejsu użytkownika wykorzystano bibliotekę programistyczną MFC (Microsoft Foundation Classes) działającą w trybie MDI (ang. *multiple-document interface*). Ponadto, dla uzyskania bardziej przyjaznego dla użytkownika wyglądu aplikacji, w tym możliwości zmiany tematów graficznych, skorzystano z biblioteki Prof-UIS Freeware. Moduł implementujący algorytmy obliczania liczności warstw omówione w rozdziale 5 wykorzystuje natomiast bibliotekę MPIR (Multiple Precision Integers and Rationals) umożliwiającą operowanie na liczbach całkowitych o dowolnej wielkości. Wszystkie pozostałe funkcjonalności, takie jak analizator składniowy skryptów użytkownika, dynamicznie generowany interfejs graficzny projektów użytkownika czy wszystkie struktury danych i algorytmy zostały w całości zaprojektowane i zaimplementowane przez autora pracy.

Koncepcja interfejsu użytkownika

Od strony użytkowej założono, że system ma umożliwiać wygodne przeprowadzanie złożonych analiz danych, a w szczególności pomagać automatyzować wiele czynności składających się na cały proces analizy. Z tego względu zdecydowano o zaimplementowaniu dwóch alternatywnych metod specyfikowania zadań, z których jedna polega na zdefiniowaniu skryptu, a druga na stworzeniu projektu graficznego. Korzystanie ze skryptów umożliwia większe zautomatyzowanie całego procesu, niemniej graficzny projekt jest zdecydowanie bardziej intuicyjny dla mniej zaawansowanych użytkowników. W obu przypadkach system zapewnia pomoc w formie krótkiej, ale wyczerpującej informacji o wszystkich dostępnych algorytmach i ich parametrach.

Dla potrzeb realizacji założeń interfejsu użytkownika autor pracy zaproponował prosty język skryptowy $\mathcal{K}$, który został opisany w punkcie 6.2.2.

Przyjęty model implementacji analizy, na którą składa się wiele mniejszych, definiowanych przez użytkownika etapów, został zainspirowany istniejącym komercyjnym oprogramowaniem SAS. W szczególności podział analizy na etapy odpowiada *krokom* w analizach SAS, poszczególne etapy zaimplementowano w postaci *procedur*, dane między kolejnymi procedurami przekazywane są w postaci zewnętrznych zbiorów (w SAS są to tabele relacyjnych baz danych, zaś w obecnej implementacji $\mathcal{K}$*-miner* pliki tekstowe), zaś składnia skryptów $\mathcal{K}$ jest wzorowana na języku SAS 4GL.

Powyższy model interakcji wymagał rozwiązania problemu wynikającego ze specyfiki analizowanych w ramach pracy algorytmów, która polega na bardzo częstych alokacjach i dealokacjach niewielkich obszarów pamięci. W sytuacji, gdy sterta procesu zarządzana była w sposób standardowy, zanotowano spadek wydajności aplikacji wraz uruchamianiem kolejnych procedur w ramach analiz działających w jednej sesji. Z tego względu zdecydowano się na wykorzystanie niestandardowego algorytmu zarządzania stertą dostępnego w systemie Microsoft Windows XP i nowszych, a mianowicie algorytmu LFH (ang. *low-fragmentation heap*). Przełączenie obsługi sterty na algorytm LFH wyeliminowało wspomniany problem.

6.2 Podstawowe elementy interfejsu użytkownika

Jak wspomniano we wcześniejszym punkcie, oprogramowanie dostarcza użytkownikowi dwie metody specyfikowania procesu analizy danych. Pierwsza polega na zdefiniowaniu skryptu w języku $\mathcal{K}$ opisującego zadania do wykonania, zaś druga na stworzeniu projektu z użyciem edytora graficznego. Oprócz tych dwóch elementów interfejsu użytkownika należy także wymienić moduł służący do graficznej prezentacji danych, a także okno komunikatów, przez które aplikacja informuje użytkownika o wykonywanych czynnościach oraz przekazuje informacje wysyłane przez biblioteki implementujące poszczególne algorytmy.

Wszystkie elementy interfejsu użytkownika zostały zaimplementowane w module aplikacji głównej.

6.2.1 Prezentacja danych

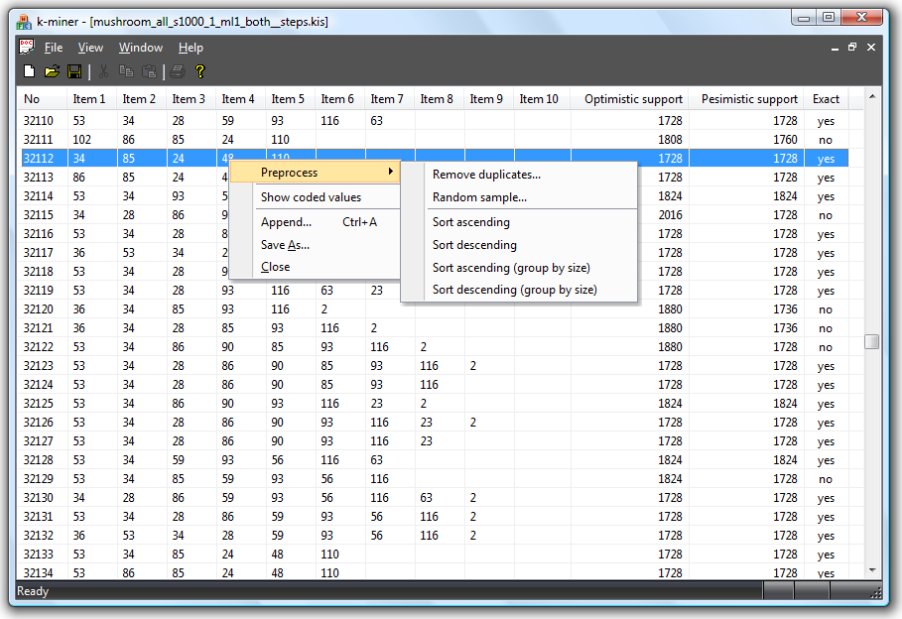
Pierwszym elementem interfejsu użytkownika jest edytor prezentujący dane przechowywane w plikach tekstowych. Zadaniem tego edytora jest udostępnienie użytkownikowi możliwości przyjrzenia się danym źródłowym, a także

dostarczenie elementarnych operacji wstępnej obróbki danych. Edytor umożliwia otwarcie plików źródłowych baz danych w formacie `*.csv` oraz `*.dat`, a także plików zawierających wzorce wraz z informacją o wsparciach w formacie `*.kis`. Formaty plików zostały opisane w dodatku A.

Wszystkie polecenia dostępne w tym edytorze zgromadzone są menu kontekstowym. Należą do nich polecenia:

- usuwania duplikatów rekordów,
- wyboru losowej próby rekordów zgodnie z rozkładem jednostajnym,
- sortowania rekordów,
- łączenia wielu plików danych w jeden.

Możliwe jest także przełączenie edytora do trybu, w którym zamiast oryginalnych wartości tekstowych poszczególnych pozycji prezentowane są przyporządkowane im przez aplikację, wykorzystywane wewnętrznie wartości całkowite. Warto w tym miejscu zauważyć, że bardziej zaawansowane opcje losowania dostarcza procedura `sample` przedstawiona w dodatku A.



Rysunek 6.1: Okno prezentacji danych

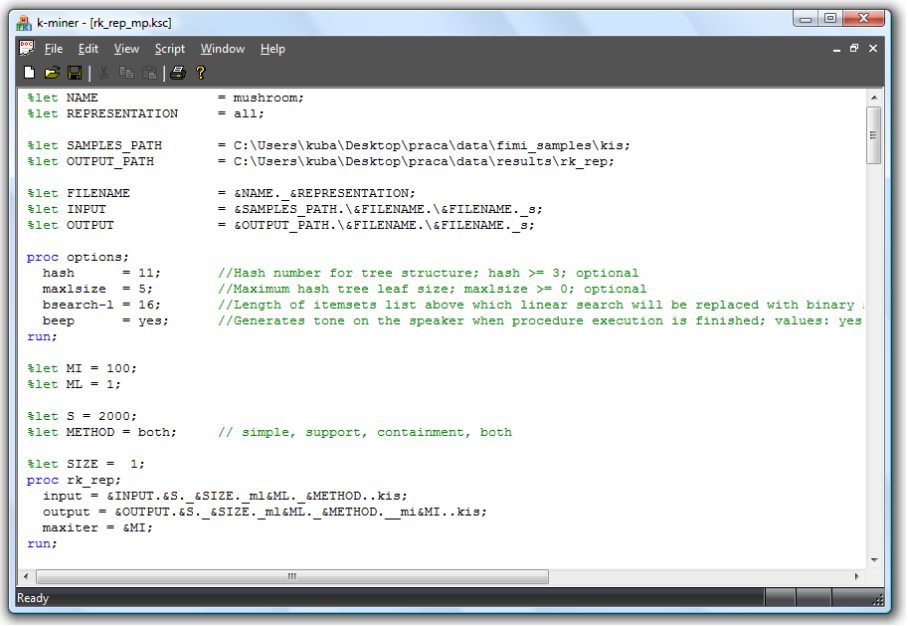
Wygląd okna prezentacji danych z otwartym plikiem w formacie `*.kis` przedstawiono na rysunku 6.1.

6.2.2 Edytor skryptów $\mathcal{K}$

Jednym z dwóch najważniejszych elementów interfejsu użytkownika jest edytor skryptów $\mathcal{K}$ obsługujący pliki tekstowe o rozszerzeniu `*.ksc`. Edytor ten umożliwia:

- definiowanie całego procesu przetwarzania danych z wykorzystaniem wszystkich aktualnie wczytanych do aplikacji procedur oraz procedur wbudowanych,
- wykonanie całego lub wybranego fragmentu zdefiniowanego procesu.

Wygląd okna edytora skryptów uwidoczniono na rysunku 6.2. Edytor na bieżąco analizuje zawartość wprowadzanego przez użytkownika tekstu i zaznacza kolorami wyróżnione elementy języka $\mathcal{K}$.



Rysunek 6.2: Okno tekstowego edytora skryptów

W celu efektywnego przetwarzania skryptów zaimplementowano rekurencyjny analizator składniowy działający metodą analizy zstępującej (ang. *top-down*).

Składnia języka $\mathcal{K}$

Wszystkie polecenia języka $\mathcal{K}$ muszą być zakończone znakiem średnika, zaś samodzielny średnik traktowany jest jako polecenie puste. Białe znaki inter-

pretowane są jak w większości języków programowania, tzn. nie ma narzuconych szczególnych ograniczeń co do ich liczby i formatowania (tak jak np. w języku *Python*). Ponadto wyróżniono trzy podstawowe elementy języka:

1. definicje makrozmiennych,
2. wywołania procedur,
3. komentarze.

Makrozmienne umożliwiają zapisanie pod wybraną nazwą symboliczną pewnego tekstu, a następnie odwoływanie się do niego przy użyciu nazwy symbolicznej. Możliwe jest rekurencyjne definiowanie makrozmiennych, tzn. definiowanie makrozmiennej przy użyciu innej, wcześniej zadeklarowanej makrozmiennej. Analizator składniowy rozwija makrozmienne momencie ich napotkania, a nie faktycznego użycia, zatem muszą być definiowane zawsze przed pierwszym odwołaniem.

Definicja makrozmiennej rozpoczyna się od słowa kluczowego `%let`, po którym następuje jej nazwa symboliczna, znak równości, przypisywana wartość tekstowa i średnik. Przykładowo definicja makrozmiennej wskazującej na folder z danymi źródłowymi przyjąć następującą postać:

```
%let NAME           = mushroom;
%let REPRESENTATION = maximal;
%let SAMPLES_PATH   = C:\data\fimi_samples\kis;
```

Nazwa symboliczna może składać się wyłącznie ze znaków alfabetu łacińskiego, cyfr i znaku podkreślenia.

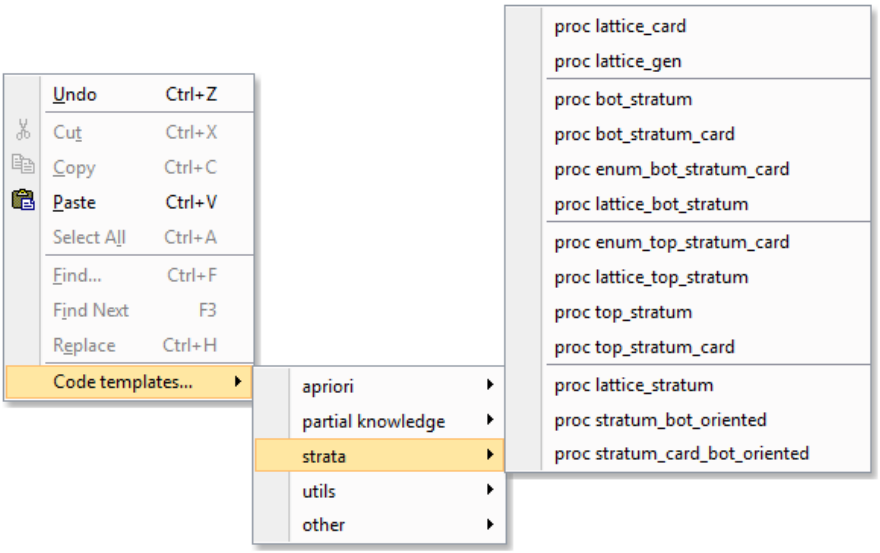
Odwołanie do makrozmiennej następuje poprzez poprzedzenie jej nazwy znakiem `&`. Jeżeli makrozmenna jest częścią większego napisu i nie występuje za nią znak biały lub średnik, należy zakończyć jej nazwę kropką. Np.:

```
%let INPUT = &SAMPLES_PATH.&NAME._&REPRESENTATION;
```

Istotną własnością makrozmiennych jest to, że są one definiowane globalnie na poziomie całej aplikacji, a nie tylko aktualnego skryptu. Oznacza to, że możliwe jest np. stworzenie jednego pliku z często wykorzystywanymi makrozmiennymi i wykorzystywanie ich definicji w innych skryptach.

Polecenie **wywołania procedury** umożliwia uruchomienie dowolnej dostępnej w aplikacji procedur lub funkcji pomocniczej, dlatego stanowi najsilniejszy element języka $\mathcal{K}$. Warto podkreślić, że użytkownik może w ten sposób odwołać się do wszystkich algorytmów dołączanych przez biblioteki dynamiczne. Z poziomu menu kontekstowego edytora skryptów możliwe

jest także zapoznanie się ze wszystkimi dostępnymi procedurami i ich parametrami, uzyskanie informacji o dopuszczalnych wartościach poszczególnych parametrów, ich wymagalności oraz (opcjonalnie) wartości domyślnej. Przykładowe menu prezentujące listę procedur implementujących algorytmy z rozdziału 5 przedstawiono na rysunku 6.3.



Rysunek 6.3: Menu dostępnych procedur

Wywołanie procedury rozpoczyna się od słowa kluczowego `proc` i jej nazwy. W dalszej kolejności podawane są wartości wszystkich parametrów wymaganych bez wartości domyślnej oraz dowolnych pozostałych parametrów. Wywołanie procedury kończy się słowem kluczowym `run`. Przykładowo wywołanie procedury wyznaczającej liczność warstwy uwiązanej do podstawy może wyglądać następująco:

```
proc bot_stratum_card;
  inputx     = &EMPTY;
  inputy     = &BOUNDARY.&S..kis;
  makechain  = no;
  progress   = yes;
run;
```

Ostatnim elementem języka $\mathcal{K}$ są komentarze, które są filtrowane przed przekazaniem skryptu dalej do analizatora składniowego. Komentarz rozpoczyna się od znaku podwójnego ukośnika `//` i kończy wraz z końcem bieżącej linii. Np.:

```
hash = 5;  //Hash number for tree structure
```

Jakkolwiek zaproponowany język skryptowy jest bardzo prosty i składa się zaledwie z trzech elementów, w praktyce jest silnym narzędziem umożliwiającym znaczne uproszczenie i zautomatyzowanie eksperymentów z wykorzystaniem wszystkich dostępnych w aplikacji algorytmów. Ponadto dzięki temu, że skrypt zapisywany jest w tekstowych plikach `*.ksc`, istnieje możliwość wygenerowania go przy użyciu zewnętrznych narzędzi (np. automatyzacja tworzenia skryptów konfiguracyjnych zawierających makrozmienne).

Uruchamianie skryptów

Domyślnie uruchomienie skryptu przyciskiem F2 lub odpowiednią pozycją w menu głównym powoduje sekwencyjne uruchomienie wszystkich zapisanych w nim poleceń. Ponadto istnieje możliwość uruchomienia wyłącznie wybranego fragmentu skryptu, np. w celu powtórzenia wybranych, a nie wszystkich etapów procesu po zmianie parametrów obliczeń. W takiej sytuacji do przetwarzania wybierana jest wyłącznie zaznaczona część skryptu.

6.2.3 Graficzny edytor projektów

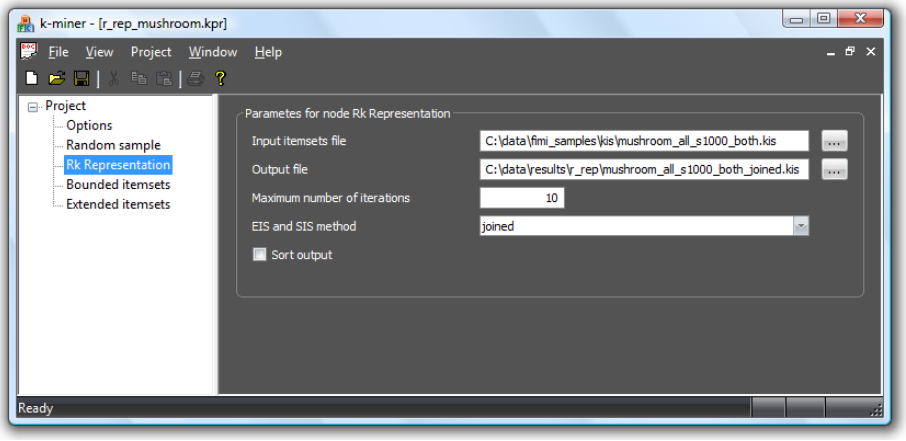
Omówiony w poprzednim punkcie edytor skryptów $\mathcal{K}$ jest bardzo elastycznym narzędziem, niemniej zgodnie z założeniami interfejsu aplikacji *$\mathcal{K}$-miner* zaimplementowano także alternatywną metodę definiowania zadań jaką jest graficzny edytor projektów. Edytor ten obsługuje pliki binarne o rozszerzeniu `*.kpr` i od strony funkcjonalnej dostarcza zasadniczą część funkcjonalności edytora skryptów jaką jest uruchamianie procedur w formie bardziej przystępnej dla mniej zaawansowanych użytkowników.

Okno edytora skryptów wraz z wczytanym przykładowym projektem przedstawiono na rysunku 6.4. Składa się ono z dwóch części:

- po lewej stronie: lista węzłów projektu,
- po prawej stronie: definicja wybranego węzła.

Lista węzłów projektu jest odpowiednikiem sekwencji wywołań procedur w edytorze skryptów[1]. W przykładowym projekcie najpierw ustawiane są odpowiednio opcje aplikacji, następnie pobierana jest próbka danych źródłowych, wyznaczana reprezentacja $\mathcal{R}^*$ oraz zbiory ograniczone i rozszerzone.

[1] Przyjęto nazwę *węzeł*, ponieważ w ogólności struktura projektu mogłaby być drzewem (po dodaniu instrukcji warunkowych).



Rysunek 6.4: Okno graficznego edytora projektów

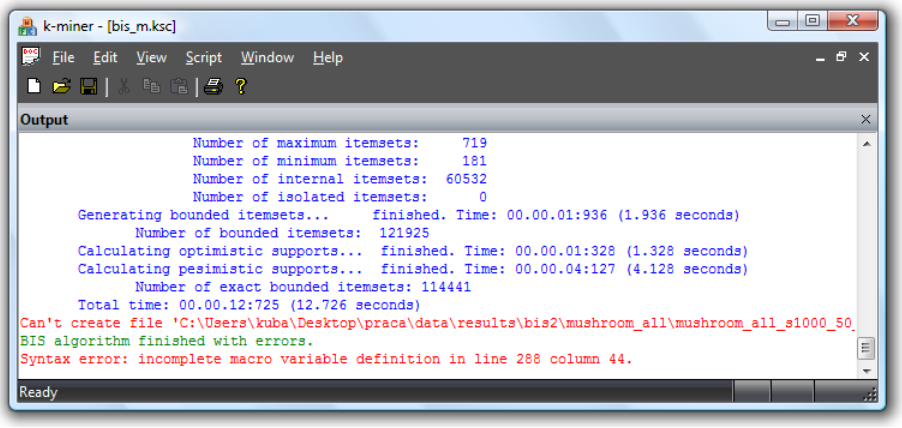
Na **definicję wybranego węzła** składają się natomiast wszystkie parametry reprezentowanej przez niego procedury. W przeciwieństwie do edytora skryptów, parametry w edytorze projektu są prezentowane w formie odpowiednio dobranych graficznych kontrolek, których rodzaj zależy od typu parametru:

- pola edycyjne z możliwością wyboru pliku: parametry tekstowe określające dane wejściowe lub wyjściowe procedury,
- pola edycyjne: parametry liczbowe i tekstowe bez predefiniowanych wartości,
- lista rozwijana typu *combobox*: parametry tekstowe z predefiniowaną listą wartości,
- przycisk typu *checkbox*: parametry przyjmujące wartość logiczną.

Edytor graficzny umożliwia uruchomienie wszystkich poleceń sekwencyjnie lub jednego wybranego polecenia. Istnieje także możliwość wyeksportowania całego projektu do tekstowego formatu skryptu $\mathcal{K}$.

6.2.4 Okno komunikatów

Czwartym wartym zaprezentowania elementem interfejsu użytkownika jest okno komunikatów przedstawione na rysunku 6.5. Okno to wraz z kilkoma dodatkowym funkcjonalnościami dostarcza wyczerpującą informację o stanie aplikacji.



Rysunek 6.5: Okno komunikatów

Tak jak wspomniano we wstępie do rozdziału, wszelkie dane wejściowe i wyjściowe dla algorytmów przekazywane są z wykorzystaniem zewnętrznych plików tekstowych. Aplikacja dostarcza jednak usługę odbioru i prezentacji w oknie komunikatów wartościowych informacji dostarczanych przez poszczególne procedury, takich jak stopień zaawansowania algorytmu, zmierzony czas wykonywania poszczególnych etapów itd. W oknie tym prezentowane są także komunikaty, których źródłem jest sama aplikacja lub jej poszczególne moduły, np. informacja o wykryciu błędów składniowych w przetwarzanym skrypcie $\mathcal{K}$. Trzecim i ostatnim przeznaczeniem okna komunikatów jest prezentacja wyniku działania specjalnych, wbudowanych w aplikację procedur `options` (zarządzanie opcjami) i `macros` (zarządzanie makrami).

Okno komunikatów wyróżnia trzy typy napisów: zwykłe wyświetlane w kolorze niebieskim, ostrzeżenia wyświetlane w kolorze zielonym i błędy wyświetlane w kolorze czerwonym.

Przy okazji omawiania sposobu przekazywania informacji od aplikacji *$\mathcal{K}$-miner* do użytkownika warto wspomnieć także o następujących funkcjonalnościach:

- powiadamianie dźwiękowe o zakończeniu procedury (zależnie od opcji aplikacji),
- alarm dźwiękowy po zakończeniu całego skryptu lub projektu (zależnie od opcji aplikacji),
- wskazywanie fragmentu w skrypcie lub projekcie, w którym nastąpiło jego nieoczekiwane przerwanie w wyniku nieprawidłowego zakończenia procedury, błędu składniowego itd.

Rozdział 7

Przeprowadzone badania i omówienie wyników

Przeprowadzone badania obejmowały analizę skuteczności wszystkich omówionych w pracy algorytmów wnioskowania z wiedzy niepełnej oraz ich wydajności. Ponadto przeanalizowano zwięzłość reprezentacji $\mathcal{R}$ i $\mathcal{R}^*$ oraz wpływ ich wykorzystania na całkowity czas obliczeń. W ostatnim punkcie przebadano wydajność algorytmów wyznaczania liczności warstw wzorców.

Wszystkie badania zostały wykonane na komputerze PC z procesorem Intel Core 2 Duo 2GHz (T7250) wyposażonym w 2GB pamięci operacyjnej i działającym pod kontrolą systemu operacyjnego Windows Vista Home Premium SP2. Z uwagi na jednowątkowość części obliczeniowej aplikacji, a tym samym wykonywanie obliczeń na jednym z dwóch dostępnych rdzeni, rzeczywiste wykorzystanie procesora wynosiło do 50%.

Mając na uwadze zwięzłość opracowania, w dalszej części rozdziału przedstawiono najistotniejsze wyniki eksperymentów. Bardziej szczegółowe dane, w tym pomiary poszczególnych etapów algorytmów, a także wyniki dodatkowych eksperymentów, można znaleźć na dołączonej do pracy płycie CD.

7.1 Badanie algorytmów wnioskowania z wiedzy niepełnej

Głównym celem badania algorytmów wnioskowania z wiedzy niepełnej było zweryfikowanie ich praktycznej *skuteczności*, przez którą należy rozumieć zdolność do odtwarzania informacji o wzorcach nieznanych na podstawie pewnej liczby wzorców znanych. Z tego powodu badania przeprowadzone zostały na zbiorach wzorców utworzonych w sposób symulujący sytuację

świadomego lub przypadkowego przekazania niepełnej informacji, która następnie zostaje poddana analizie.

Do badania wykorzystano dwa zbiory danych pobrane z repozytorium FIMI[1]:

- zbiór *mushroom*: 119 pozycji, 8124 transakcje, średnia długość transakcji 23,
- zbiór *kosarak*: 41270 pozycji, 990002 transakcje, średnia długość transakcji 8,1.

Korzystając z zaimplementowanego w aplikacji $\mathcal{K}$-*miner* algorytmu Apriori [3] wygenerowano wszystkie wzorce częste dla wybranych eksperymentalnie wartości minimalnego wsparcia[2]: 2000 dla zbioru *mushroom* ($|\mathcal{F}| = 6548$) i 1500 dla zbioru *kosarak* ($|\mathcal{F}| = 218766$). Następnie z tak przygotowanych zbiorów wzorców częstych utworzono testowe zbiory wzorców znanych jako próby losowe o wielkości od 1% do 50%, które zostały wykorzystane jako źródło danych dla badanych algorytmów. Próby losowe pobrano metodą bez zwracania na cztery sposoby:

1. losowanie proste - losowanie wzorców zgodnie z rozkładem jednostajnym,
2. losowanie ze wsparciem - jak w przypadku 1, przy czym dla każdego wzorca dodatkowo losowany był drugi wzorzec spośród wzorców o tym samym wsparciu; jeżeli takie wzorce nie istniały, do próby dołączany był tylko pierwszy wzorzec,
3. losowanie z zawieraniem - jak w przypadku 1, przy czym dla każdego wzorca dodatkowo losowany był drugi wzorzec spośród wzorców będących jego nadzbiorami lub podzbiorami właściwymi; jeżeli takie wzorce nie istniały, do próby dołączany był tylko pierwszy wzorzec,
4. losowanie z zawieraniem i wsparciem - jak w przypadku 1, przy czym dla każdego wzorca dodatkowo losowany był drugi wzorzec spośród wzorców będących jego nadzbiorami lub podzbiorami właściwymi o tym samym wsparciu; jeżeli takie wzorce nie istniały, do próby dołączany był tylko pierwszy wzorzec.

Warto zauważyć, że wybrane zbiory danych *mushroom* i *kosarak* istotnie się różnią. Pierwszy zbiór składa się z niewielkiej liczby pozycji i transakcji

[1] Frequent Itemset Mining Dataset Repository, `http://fimi.ua.ac.be/data`.

[2] Wartość minimalnego wsparcia wybrano w taki sposób, aby czas wykonania jednej serii eksperymentów każdego algorytmu nie przekroczył sześciu godzin.

w stosunku do drugiego zbioru, a jednocześnie średnia długość transakcji jest prawie prawie trzykrotnie większa niż w drugim zbiorze. Należy się zatem spodziewać, że wzorce znane będące próbą losową wszystkich wzorców częstych zbioru *mushroom* będą znacznie bardziej skorelowane niż wzorce znane wylosowane ze wszystkich wzorców częstych zbioru *kosarak*. Przez stopień korelacji należy rozumieć względną liczbę wzorców będących w relacji zawierania lub o tej samej wartości wsparcia. W efekcie algorytmy wnioskowania z wiedzy niepełnej powinny dawać lepsze rezultaty dla zbioru *mushroom*. Teza ta zostanie zweryfikowana w dalszej części rozdziału.

W kolejnych punktach w tabelach umieszczono wyniki uzyskane dla czwartego sposobu pobierania próby losowej, natomiast na wykresach przedstawiono porównanie wyników dla poszczególnych metod. Dla zbioru *kosarak* nie przeprowadzono eksperymentów dla trzeciej metody losowania, ponieważ zaimplementowany algorytm losujący wzorce nie umożliwiał pobrania prób losowych tym sposobem dla wartości minimalnego wsparcia równej 1500 ($|\mathcal{F}| = 218766$) ze względu na zbyt duże zapotrzebowanie na pamięć operacyjną.

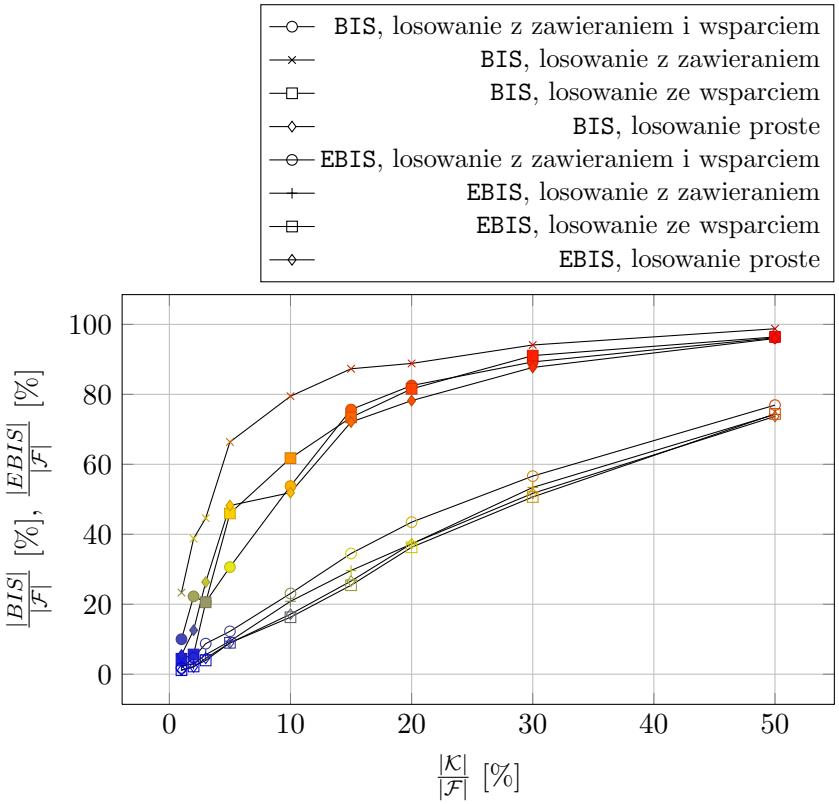
7.1.1 Badanie algorytmu BIS

W pierwszej kolejności przeanalizowano skuteczność działania algorytmu `BIS`. Zarówno w przypadku zbioru *mushroom*, jak i *kosarak*, uzyskane wyniki wskazują na dużą zdolność algorytmu do odtwarzania przybliżonej informacji o nieznanych wzorcach częstych. Dla zbioru *mushroom* (tabela 7.1), 1% wzorców częstych wystarczył do wyprowadzenia przybliżonej informacji o 9,988% wszystkich wzorców częstych, podczas gdy dla zbioru *kosarak*

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert BIS\rvert$	$\lvert EBIS\rvert$	$\frac{\lvert BIS\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\frac{\lvert EBIS\rvert}{\lvert\mathcal{F}\rvert}$ [%]
1	65	654	117	9.988	1.787
2	131	1457	308	22.251	4.704
3	196	1350	572	20.617	8.736
5	327	2005	802	30.62	12.248
10	655	3522	1511	53.787	23.076
15	982	4954	2260	75.657	34.514
20	1309	5403	2846	82.514	43.464
30	1964	5845	3708	89.264	56.628
50	3274	6297	5037	96.167	76.924

Tabela 7.1: Skuteczność algorytmu BIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

(tabela 7.2) przy tej samej wielkości próby udało się odtworzyć ponad 50% wszystkich wzorców w sposób przybliżony. W obu przypadkach znajomość połowy wszystkich wzorców częstych umożliwiła odtworzenie ponad 96% z nich wraz z oszacowaniem wartości wsparcia. Jak widać na wykresach 7.1 i 7.2, wraz ze wzrostem liczności próby maleje stosunek liczby wzorców znanych do liczby utworzonych na ich podstawie wzorców ograniczonych. Ponadto dla zbioru *mushroom* największą skuteczność algorytmu uzyskano w przypadku losowania prób z uwzględnieniem relacji zawierania, co jest zgodne z oczekiwaniami, ponieważ definicja wzorców ograniczonych bazuje na tej relacji.

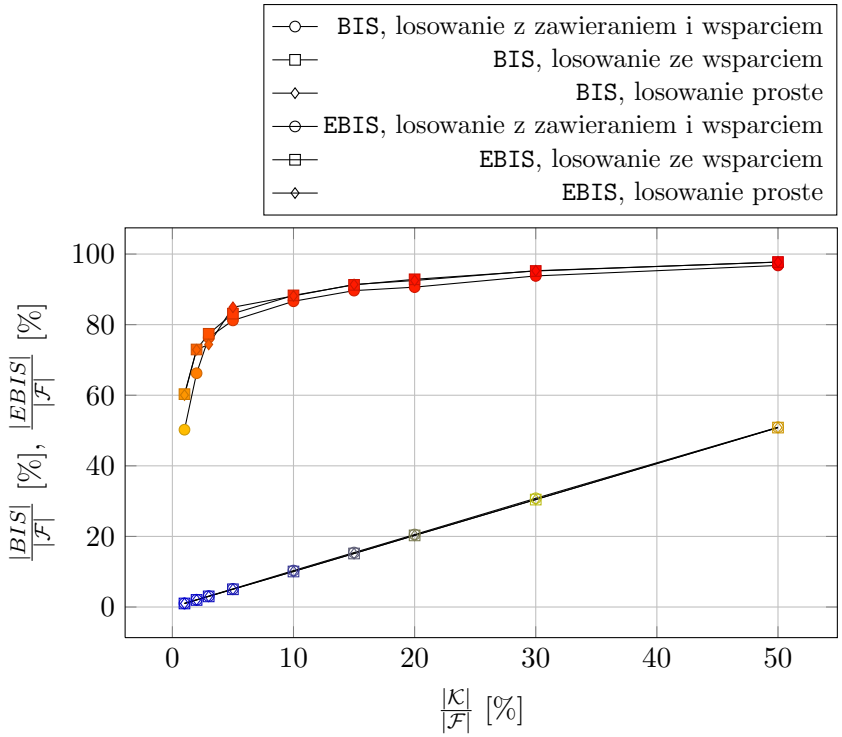


Rysunek 7.1: Skuteczność algorytmu BIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)

Analizując wyniki działania algorytmu `EBIS` dla zbioru *mushroom* widać, że podobnie jak w przypadku algorytmu `BIS`, skuteczność algorytmu `EBIS` dla tego zbioru jest również wysoka. Liczba zbiorów dokładnie ograniczonych w zależności od wielkości próby jest od półtora do prawie trzech razy większa od liczby wzorców znanych. Przykładowo dla próby o wielkości

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert BIS\rvert$	$\lvert EBIS\rvert$	$\frac{\lvert BIS\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\frac{\lvert EBIS\rvert}{\lvert\mathcal{F}\rvert}$ [%]
1	2188	109893	2252	50.233	1.029
2	4375	144899	4511	66.235	2.062
3	6563	167014	6758	76.344	3.089
5	10939	177633	11245	81.198	5.14
10	21877	189494	22589	86.62	10.326
15	32816	196103	33767	89.641	15.435
20	43754	198253	44929	90.623	20.538
30	65631	205208	67396	93.803	30.807
50	109385	211704	111422	96.772	50.932

Tabela 7.2: Skuteczność algorytmu BIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem



Rysunek 7.2: Skuteczność algorytmu BIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

3%, liczba zbiorów dokładnie ograniczonych to ponad 8,7% wszystkich wzorców częstych, zaś dla próby 50% algorytm był w stanie odzyskać dokładną informację o prawie 77% wzorców częstych.

W odróżnieniu od zbioru *mushroom*, w przypadku zbioru *kosarak* liczba wyprowadzonych nowych wzorców dokładnie ograniczonych jest stosunkowo niewielka. Próbę o wielkości 3% algorytm `EBIS` był w stanie powiększyć do 3,089% wszystkich wzorców częstych, zaś próbę 50% do 50,932%. Potwierdza to tezę postawioną we wstępie do tego punktu, że stopień korelacji wzorców częstych dla zbioru *kosarak* jest mniejszy niż w przypadku wzorców częstych zbioru *mushroom*, co przekłada się istotnie na uzyskiwane rezultaty.

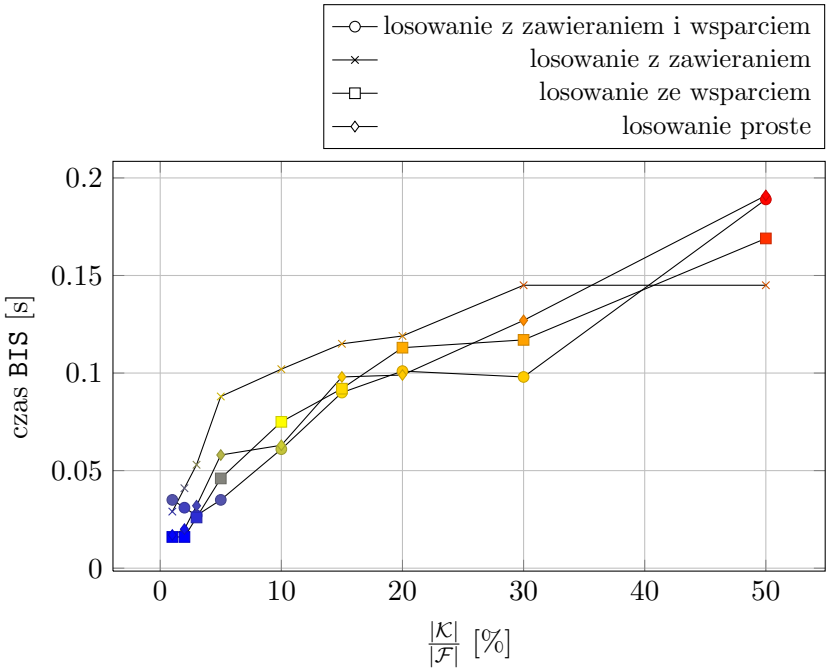
Na wykresie 7.1 widać ponadto, że w przypadku algorytmu `EBIS` dla zbioru *mushroom* najlepsze wyniki uzyskuje się przy losowaniu próby z uwzględnieniem wartości wsparć i relacji zawierania. Podobna relacja zachodzi dla zbioru *kosarak*, choć w mniejszym stopniu.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	czas `OSup` [s]	czas `PSup` [s]	czas `BIS` [s]
1	0.004	0.004	0.035
2	0.007	0.008	0.031
3	0.005	0.005	0.027
5	0.007	0.008	0.035
10	0.012	0.014	0.061
15	0.017	0.024	0.09
20	0.017	0.022	0.101
30	0.015	0.022	0.098
50	0.016	0.028	0.189

Tabela 7.3: Wydajność algorytmu BIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

W tabelach 7.3 i 7.4 przedstawiono całkowity czas obliczeń algorytmu `BIS`, a także czas wyznaczania wartości minimalnego i maksymalnego wsparcia[3]. Na podstawie tych danych oraz wyników eksperymentów dla innych wartości minimalnego wsparcia załączonych na płycie CD można stwierdzić, że w większości przypadków czas wyznaczania wszystkich wzorców ograniczonych stanowi od około 10% do 40% całkowitego czasu obliczeń algorytmu `BIS`. Pozostała część to szacowanie wartości wsparć.

[3]Nie podano czasu obliczeń algorytmu `EBIS`, ponieważ algorytm ten różni się od `BIS` jedynie tym, że w ostatnim kroku ze wszystkich wzorców ograniczonych wybierane są wyłącznie wzorce dokładnie ograniczone. Operacja ta praktycznie nie ma wpływu na całkowity czasu obliczeń.

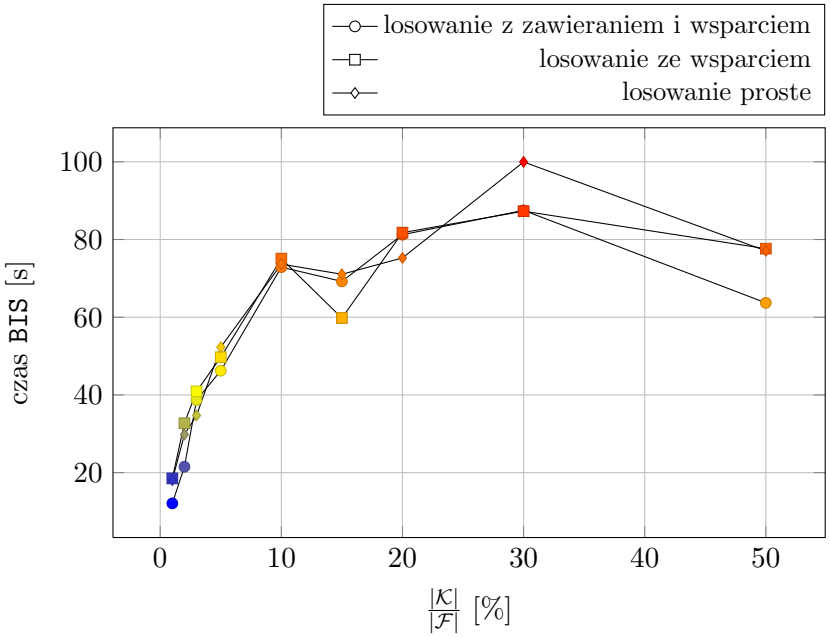


Rysunek 7.3: Wydajność algorytmu BIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	czas `OSup` [s]	czas `PSup` [s]	czas `BIS` [s]
1	5.119	5.382	12.091
2	8.145	8.588	21.518
3	15.349	16.212	38.683
5	16.588	18.082	46.241
10	21.369	24.994	72.893
15	21.285	26.24	69.217
20	20.854	28.058	81.243
30	20.217	30.49	87.507
50	14.173	31.315	63.693

Tabela 7.4: Wydajność algorytmu BIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem

Warto przy tym zauważyć, że dla większych prób uwidacznia się większa złożoność obliczeniowa algorytmu `PSup` w stosunku do algorytmu `OSup`. Algorytm `PSup` dla każdego zbioru ograniczonego wyszukuje wśród zbiorów ograniczonych jego podzbiory o jeden krótsze, podczas gdy algorytm `OSup` wykonuje tę operację tylko dla zbiorów, które nie są dokładnie ograniczone. Dzięki tej własności zwiększenie wielkości próby może spowodować przyspieszenie działania algorytmu `OSup` i w efekcie skrócenie całkowitego czasu obliczeń `BIS`, podczas gdy algorytm `PSup` zwiększa czas działania wraz ze zwiększeniem próby (z dokładnością do precyzji pomiaru). Opisywany efekt zaobserwowano dla zbioru danych *kosarak* (tabela 7.4).



Rysunek 7.4: Wydajność algorytmu BIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

7.1.2 Badanie algorytmu EIS

Algorytm `EIS`, służący do generowania wzorców rozszerzonych, umożliwia wyprowadzenie nowych wzorców tylko wtedy, gdy wśród wzorców znanych znajdują się wzorce będące w relacji zawierania, a ponadto wartości ich wsparć są równe. W związku z tym, podobnie jak w przypadku algorytmu `EBIS`, skuteczność algorytmu `EIS` powinna być większa dla próbek pobranych ze zbioru wszystkich wzorców częstych *mushroom* niż ze zbioru wszystkich wzorców częstych *kosarak*.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert EIS\rvert$	$\frac{\lvert EIS\rvert}{\lvert\mathcal{F}\rvert}$ [%]
1	65	76	1.161
2	131	182	2.78
3	196	286	4.368
5	327	492	7.514
10	655	1095	16.723
15	982	1896	28.955
20	1309	2612	39.89
30	1964	3656	55.834
50	3274	5141	78.513

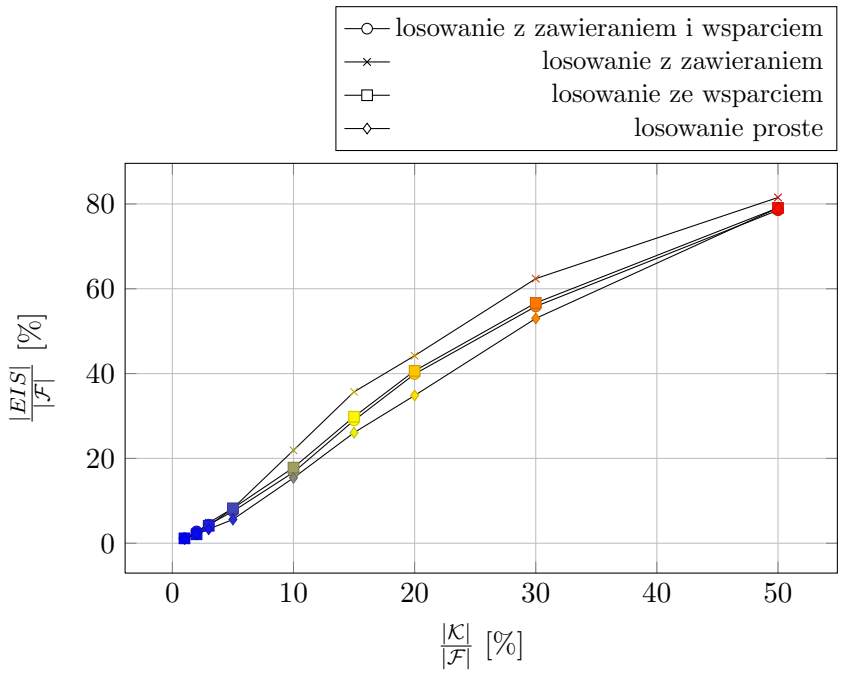
Tabela 7.5: Skuteczność algorytmu EIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

W tabeli 7.5 przedstawiono wyniki uzyskane dla zbioru *mushroom*. Skuteczność algorytmu `EIS` dla tego zbioru jest wysoka, szczególnie dla prób wielkości od 15% do 30%, dla których algorytm jest w stanie niemal podwoić liczbę wzorców. W porównaniu do algorytmu `EBIS` widać, że o ile `EIS` dla mniejszych prób wyprowadza mniej nowych wzorców, o tyle dla próby wielkości 50% liczba nowych wzorców rozszerzonych przekroczyła liczbę nowych wzorców dokładnie ograniczonych. Algorytm `EIS` w tym przypadku był w stanie odtworzyć pełną informację o ponad 78% wzorców częstych.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert EIS\rvert$	$\frac{\lvert EIS\rvert}{\lvert\mathcal{F}\rvert}$ [%]
1	2188	2268	1.037
2	4375	4703	2.15
3	6563	6995	3.198
5	10939	11884	5.432
10	21877	23535	10.758
15	32816	35232	16.105
20	43754	46631	21.316
30	65631	69166	31.616
50	109385	112854	51.587

Tabela 7.6: Skuteczność algorytmu EIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem

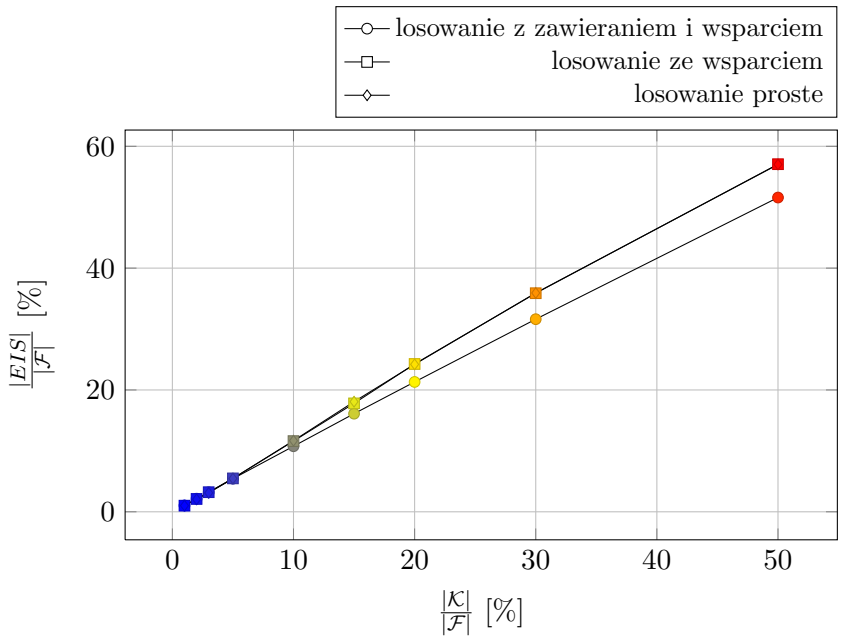
Wyniki dla zbioru *kosarak* zestawiono w tabeli 7.6. W tym przypadku skuteczność algorytmu jest zdecydowanie mniejsza. Największy przyrost



Rysunek 7.5: Skuteczność algorytmu EIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)

wzorców równy 1,616% wszystkich wzorców częstych uzyskano dla próby wielkości 30%. Warto zauważyć, że w odróżnieniu od wyników dla zbioru *mushroom*, dla zbioru *kosarak* dla każdej wielkości próby skuteczność algorytmu `EIS` jest większa od skuteczności algorytmu `EBIS`.

Istotną cechą charakterystyczną algorytmu `EIS` jest zdolność do wyprowadzania nowych wzorców o długości równej lub większej od długości najdłuższego wzorca znanego, a więc wzorców nie będących wzorcami ograniczonymi przez wzorce znane. W tabelach 7.7 i 7.8 przedstawiono porównanie długości i liczby najdłuższych wzorców znanych z długością i liczbą najdłuższych wzorców rozszerzonych. Widać, że w przypadku obu testowych zbiorów algorytm `EIS` wyprowadził wzorce rozszerzone nie będące podzbiorami żadnego wzorca znanego. Przykładowo dla zbioru *mushroom* i prób 3%, 5% oraz 10% długość najdłuższego wzorca zwiększyła się z 10 wśród wzorców znanych do 11 wśród wzorców rozszerzonych. W przypadku zbioru *kosarak* algorytm dla kilku wielkości prób wyprowadził nowe wzorce rozszerzone o długości równej długości najdłuższego wzorca znanego.



Rysunek 7.6: Skuteczność algorytmu EIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	maksymalna długość wzorców w $\mathcal{K}$	liczba wzorców $\mathcal{K}$ maksymalnej długości	maksymalna długość wzorców w EIS	liczba wzorców EIS maksymalnej długości
1	9	1	9	3
2	11	1	11	1
3	10	2	11	1
5	10	1	11	1
10	10	3	11	1
15	11	1	11	1
20	11	1	11	1
30	11	1	11	1
50	11	1	11	1

Tabela 7.7: Rozszerzanie najdłuższych wzorców: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

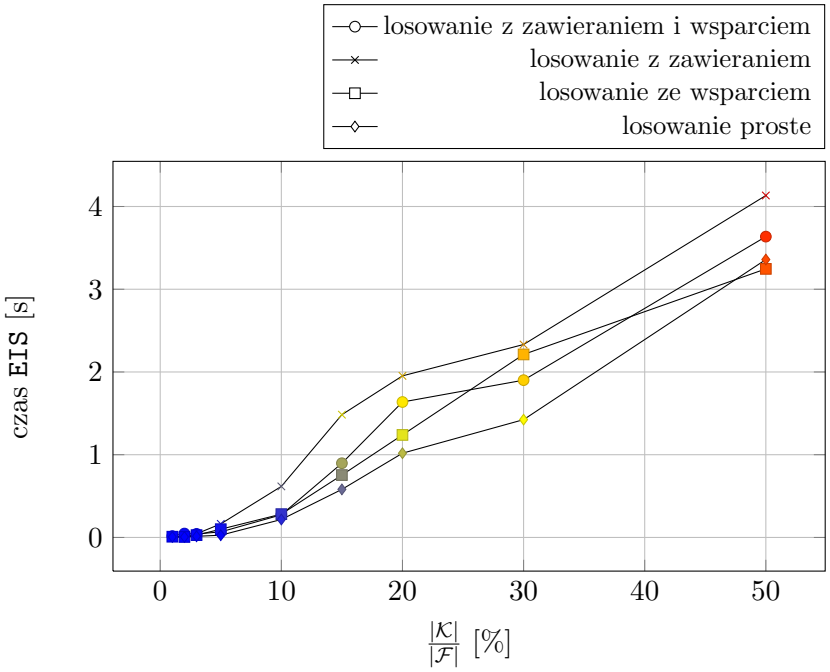
$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	maksymalna długość wzorców w $\mathcal{K}$	liczba wzorców $\mathcal{K}$ maksymalnej długości	maksymalna długość wzorców w EIS	liczba wzorców EIS maksymalnej długości
1	15	3	15	3
2	15	3	15	5
3	16	3	16	4
5	16	3	16	4
10	16	1	16	1
15	16	4	16	4
20	16	2	16	3
30	16	6	16	7
50	17	1	17	1

Tabela 7.8: Rozszerzanie najdłuższych wzorców: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	liczba iteracji	średni czas iteracji [s]	czas `EIS` [s]
1	2	0.002	0.016
2	3	0.01	0.047
3	3	0.012	0.043
5	3	0.019	0.067
10	4	0.066	0.274
15	4	0.203	0.897
20	4	0.38	1.637
30	3	0.593	1.901
50	3	1.189	3.636

Tabela 7.9: Wydajność algorytmu EIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

Analizując wyniki pomiaru wydajności algorytmu `EIS` przedstawione w tabelach 7.9 oraz 7.10 widać, że zależnie od wielkości próby algorytm wykonywał różną liczbę iteracji, co miało wpływ na całkowity czas obliczeń. Z tego względu podano także średni czas wykonania jednej iteracji. Porównując wydajność algorytmu `EIS` z wydajnością algorytmu `BIS` można zauważyć, że złożoność obliczeniowa algorytmu `EIS` jest większa od złożoności algorytmu `BIS`. Dla niewielkich prób wyznaczanie wzorców rozszerzonych



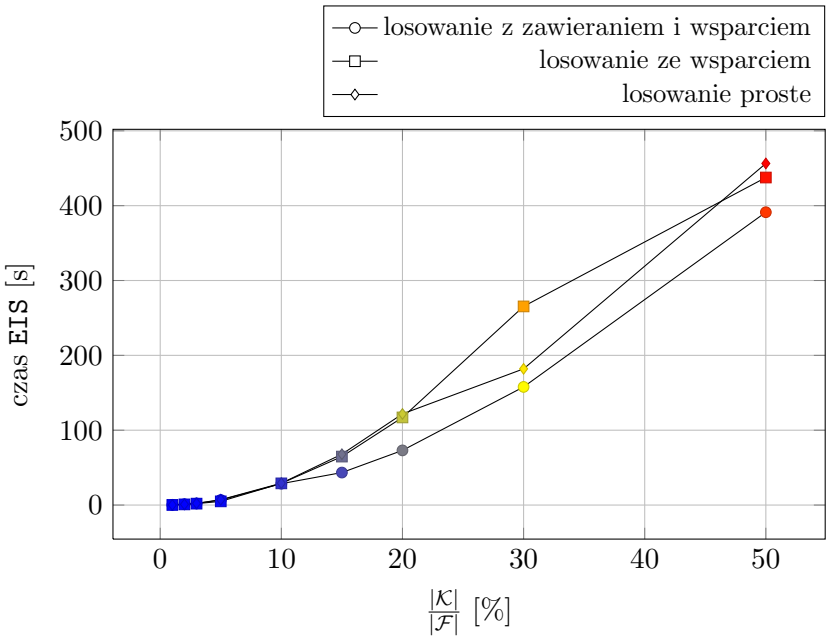
Rysunek 7.7: Wydajność algorytmu EIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)

trwa krócej niż wyznaczanie wzorców ograniczonych, niemniej wraz ze wzrostem liczby wzorców znanych czas obliczeń algorytmu `EIS` wzrasta szybciej niż czas obliczeń algorytmu `BIS`. W przypadku wybierania prób losowych ze zbioru wszystkich wzorców częstych wygenerowanych dla zbioru *mushroom* dla minimalnego wsparcia równego 1600 (wyniki na załączonej płycie CD), czas obliczeń algorytmu `BIS` dla każdej próby losowej nie przekroczył 7 sekund, podczas gdy obliczenia algorytmu `EIS` dla próby wielkości 5% trwały ponad 3428 sekund, zaś dla kolejnego progu 10% zostały przerwane po około sześciu godzinach.

Istotnym problemem obliczeniowym algorytmu `EIS` jest przetwarzanie list nadzbiorów o tym samym wsparciu wszystkich podzbiorów znanych każdego zbioru znanego. Liczba podzbiorów i nadzbiorów w przypadku mocno skorelowanych wzorców zbioru *mushroom* może być bardzo duża, co powoduje opisane różnice w wydajności `EIS` i `BIS` dla mniejszych wartości minimalnego wsparcia. W przypadku mało skorelowanych wzorców zbioru *kosarak*, algorytmu `EIS` również wykonywał się dłużej od algorytmu `BIS`, jednak różnica była znacznie mniejsza dla każdej przebadanej próby losowej wzorców.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	liczba iteracji	średni czas iteracji [s]	czas `EIS` [s]
1	3	0.09	0.273
2	3	0.362	1.092
3	3	0.717	2.157
5	3	2.241	6.724
10	4	7.142	28.566
15	3	14.468	43.406
20	3	24.34	73.019
30	3	52.609	157.828
50	3	130.41	391.23

Tabela 7.10: Wydajność algorytmu EIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem



Rysunek 7.8: Wydajność algorytmu EIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

7.1.3 Badanie algorytmu SIS

Wyniki eksperymentów obrazujące skuteczność algorytmu `SIS`, służącego do wyprowadzania wzorców skurczonych, przedstawiono w tabelach 7.11 i 7.12.

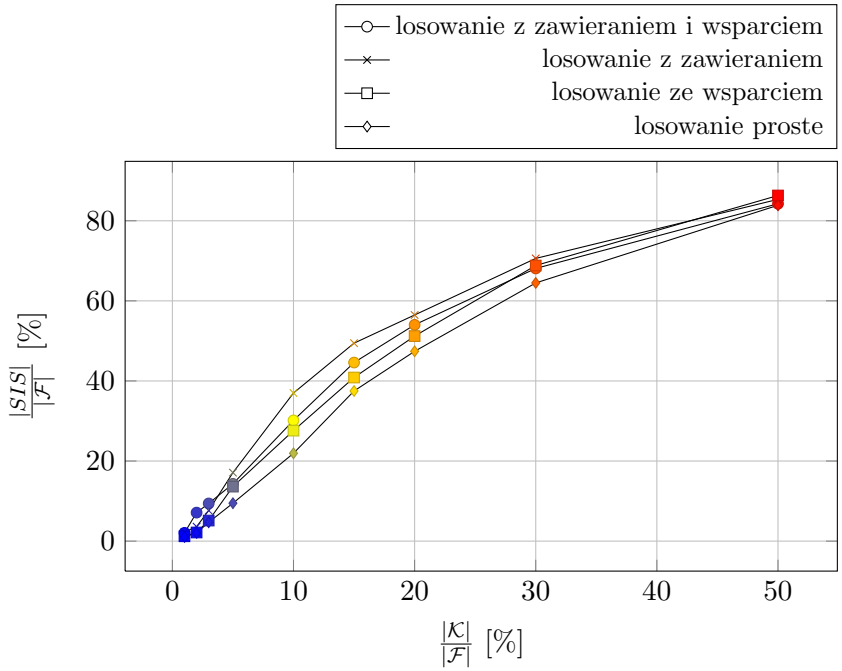
$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert SIS\rvert$	$\frac{\lvert SIS\rvert}{\lvert\mathcal{F}\rvert}$ [%]
1	65	135	2.062
2	131	466	7.117
3	196	616	9.408
5	327	937	14.31
10	655	1974	30.147
15	982	2920	44.594
20	1309	3537	54.017
30	1964	4459	68.097
50	3274	5517	84.255

Tabela 7.11: Skuteczność algorytmu SIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

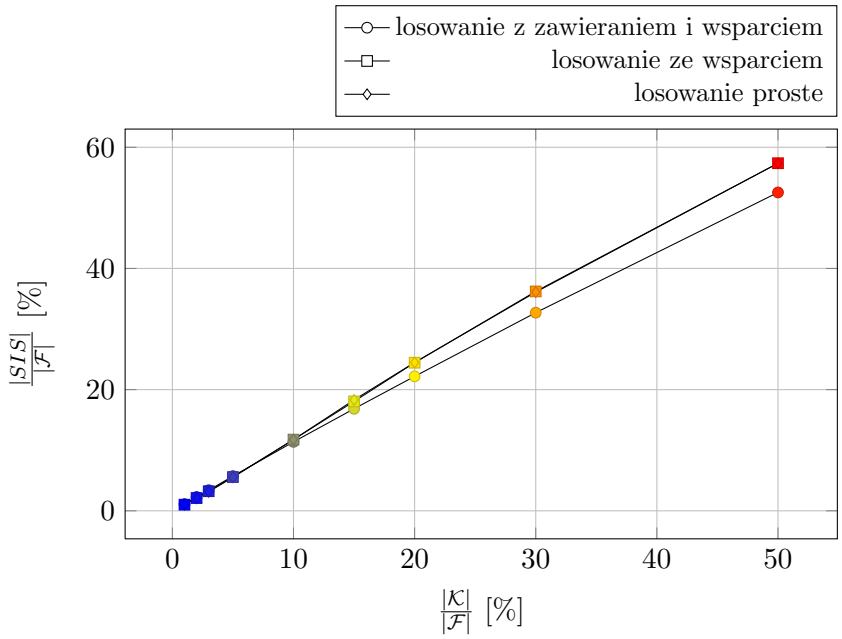
Podobnie jak w przypadku omówionych wcześniej algorytmów `EBIS` i `EIS`, algorytm `SIS` okazał się znacznie skuteczniejszy dla zbioru *mushroom* niż dla zbioru *kosarak*. W przypadku pierwszego zbioru, liczba wzorców skurczonych dla próby wielkości 10% wynosiła ponad 30,147% wszystkich wzorców częstych, zaś dla próby 50% osiągnęła wartość 84,255% wzorców częstych. Dla każdej wielkości próby algorytm `SIS` wyprowadził istotnie więcej nowych wzorców niż algorytmy `EBIS` oraz `EIS`.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert SIS\rvert$	$\frac{\lvert SIS\rvert}{\lvert\mathcal{F}\rvert}$ [%]
1	2188	2394	1.094
2	4375	4971	2.272
3	6563	7427	3.395
5	10939	12519	5.723
10	21877	24889	11.377
15	32816	36839	16.84
20	43754	48528	22.183
30	65631	71532	32.698
50	109385	114941	52.541

Tabela 7.12: Skuteczność algorytmu SIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem



Rysunek 7.9: Skuteczność algorytmu SIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)



Rysunek 7.10: Skuteczność algorytmu SIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

Liczba nowych wzorców wyprowadzonych przez algorytm SIS dla zbioru *kosarak* w najlepszym przypadku osiągnęła 2,698% wszystkich wzorców częstych. Podobnie jak w przypadku zbioru *mushroom*, liczba wyprowadzonych wzorców skurczonych była dla każdej wielkości próby większa niż liczba wyprowadzonych wzorców dokładnie ograniczonych oraz wzorców rozszerzonych.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	minimalna długość wzorców w $\mathcal{K}$	liczba wzorców $\mathcal{K}$ minimalnej długości	minimalna długość wzorców w SIS	liczba wzorców SIS minimalnej długości
1	1	1	1	1
2	2	4	2	4
3	2	4	2	4
5	1	1	1	1
10	1	2	1	2
15	1	4	1	4
20	1	12	1	12
30	1	8	1	8
50	1	17	1	17

Tabela 7.13: Skracanie najkrótszych wzorców: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

Cechą szczególną algorytmu SIS jest fakt, że nowo wyprowadzone wzorce skurczone mogą mieć długość równą lub mniejszą od długości najkrótszego wzorca znanego, a tym samym mogą nie być ograniczone przez wzorce znane. W tabelach 7.13 oraz 7.14 zestawiono długość i liczbę najdłuższych wzorców znanych z długością i liczbą najdłuższych wzorców skurczonych. Dla żadnego z przebadanych przypadków algorytm SIS nie wyprowadził wzorców nie będących wzorcami ograniczonymi.

W przypadku algorytmu SIS istotne jest zwrócenie uwagi na jego wydajność. W poprzednim punkcie zauważono, że złożoność obliczeniowa algorytmu EIS jest większa od złożoności obliczeniowej algorytmu BIS, a różnica między oboma algorytmami zależy przede wszystkim od stopnia korelacji wzorców znanych. Dla algorytmu SIS sytuacja jest analogiczna, przy czym jego złożoność jest większa niż algorytmu EIS. Porównując dane z tabel 7.3, 7.9 oraz 7.15 widać, że dla największej próby 50% czas obliczeń BIS wynosił 0,189 sekundy, czas obliczeń EIS 3,636 sekundy, podczas gdy wykonanie algorytmu SIS wymagało ponad 151 sekund. W przypadku pobierania prób

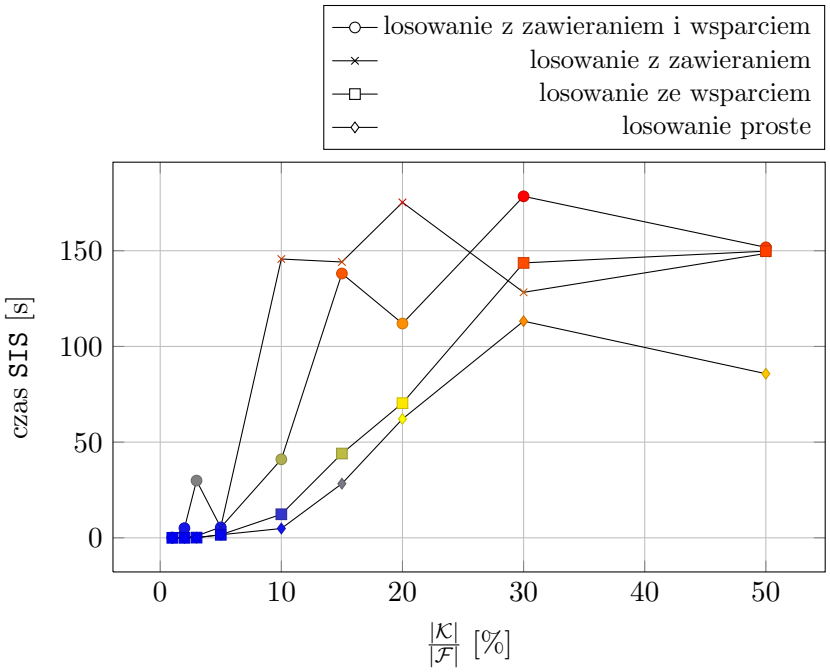
$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	minimalna długość wzorców w $\mathcal{K}$	liczba wzorców $\mathcal{K}$ minimalnej długości	minimalna długość wzorców w SIS	liczba wzorców SIS minimalnej długości
1	1	5	1	5
2	1	5	1	5
3	1	22	1	22
5	1	29	1	29
10	1	64	1	64
15	1	95	1	95
20	1	117	1	117
30	1	178	1	178
50	1	314	1	314

Tabela 7.14: Skracanie najkrótszych wzorców: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	liczba iteracji	średni czas iteracji [s]	czas `SIS` [s]
1	4	0.028	0.131
2	6	0.817	5.06
3	4	7.465	29.902
5	5	1.054	5.417
10	6	6.812	41.021
15	5	27.542	138.101
20	4	27.964	111.971
30	4	44.52	178.433
50	3	50.523	151.786

Tabela 7.15: Wydajność algorytmu SIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

losowych dla progu minimalnego wsparcia równego 1800, dla próby wielkości 5% algorytm `SIS` zakończył obliczenia po prawie czterech godzinach, zaś dla próby wielkości 10% obliczenia zostały przerwane po około sześciu godzinach. Dla tych samych parametrów obliczeń algorytmy `BIS` i `EIS` wykonywały się czasie od kilku do kilkudziesięciu sekund.

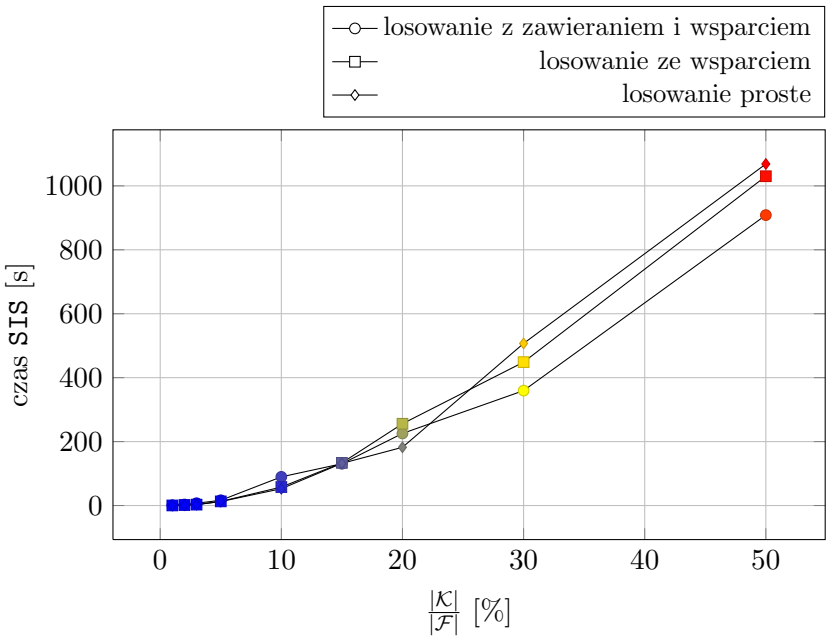


Rysunek 7.11: Wydajność algorytmu SIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)

Analogicznie jak dla algorytmu `EIS`, duży czas obliczeń algorytmu `SIS` dla zbioru *mushroom* wynika z konieczności przetwarzania długich list nadzbiorów i podzbiorów o tym samym wsparciu wszystkich wzorców znanych. Dla zbioru *kosarak* algorytm `SIS` jest również wolniejszy od algorytmu `BIS` i `EIS`, niemniej różnica w wydajności jest znacznie mniejsza niż w przypadku zbioru *mushroom*.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	liczba iteracji	średni czas iteracji [s]	czas `SIS` [s]
1	4	0.175	0.702
2	4	0.776	3.108
3	4	1.787	7.149
5	4	4.094	16.377
10	5	17.939	89.697
15	4	32.689	130.755
20	4	56.371	225.484
30	3	119.82	359.462
50	3	302.84	908.52

Tabela 7.16: Wydajność algorytmu SIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem



Rysunek 7.12: Wydajność algorytmu SIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

7.1.4 Badanie algorytmu EBIS*

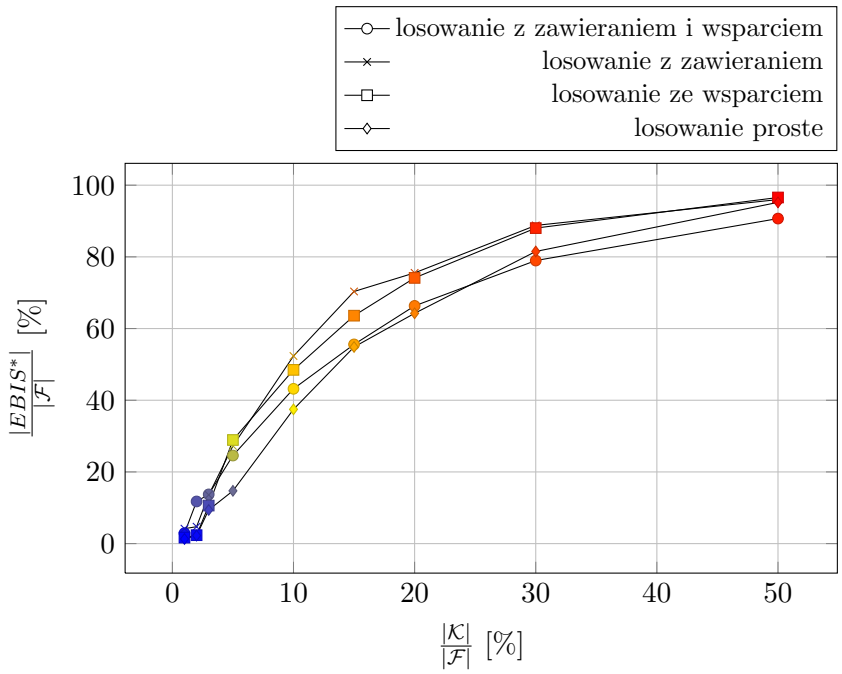
Algorytm `EBIS*` wykorzystuje własności wszystkich przeanalizowanych wyżej algorytmów, a tym samym jego skuteczność powinna być nie mniejsza od skuteczności każdego z nich z osobna. Potwierdzają to wyniki eksperymentów przedstawione w tabelach 7.17 i 7.18. Dla wszystkich przeanalizowanych przypadków, liczba wzorców wyprowadzonych przez algorytm `EBIS*` jest większa niż liczba wzorców wyprowadzonych osobno przez algorytmy `EBIS`, `EIS` oraz `SIS`.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert EBIS\rvert$	$\lvert EBIS^*\rvert$	$\frac{\lvert EBIS^*\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\frac{\lvert EBIS^*\rvert}{\lvert EBIS\rvert}$ [%]
1	65	117	190	2.902	162.393
2	131	308	770	11.759	250
3	196	572	899	13.729	157.168
5	327	802	1609	24.572	200.623
10	655	1511	2828	43.189	187.161
15	982	2260	3639	55.574	161.018
20	1309	2846	4343	66.326	152.6
30	1964	3708	5171	78.971	139.455
50	3274	5037	5938	90.684	117.888

Tabela 7.17: Skuteczność algorytmu EBIS* i porównanie z EBIS: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

Z tych samych przyczyn co w przypadku omówionych wcześniej algorytmów, skuteczność algorytmu `EBIS*` jest znacznie większa dla zbioru *mushroom* niż dla zbioru *kosarak*. Dla niewielkiej próby wielkości 2%, liczba wzorców wyprowadzonych przy użyciu algorytmu `EBIS*` dla zbioru *mushroom* wynosi prawie 12% wszystkich wzorców częstych, podczas gdy duża próba 50% umożliwia odzyskanie dokładnej informacji o ponad 90% wszystkich wzorców częstych. Dla zbioru *kosarak* wyniki są wyraźnie słabsze, choć również w każdym przypadku algorytm `EBIS*` był w stanie wyprowadzić nowe wzorce. Największy przyrost uzyskano dla próby o wielkości 30% i wynosił on 3,977% wszystkich wzorców częstych.

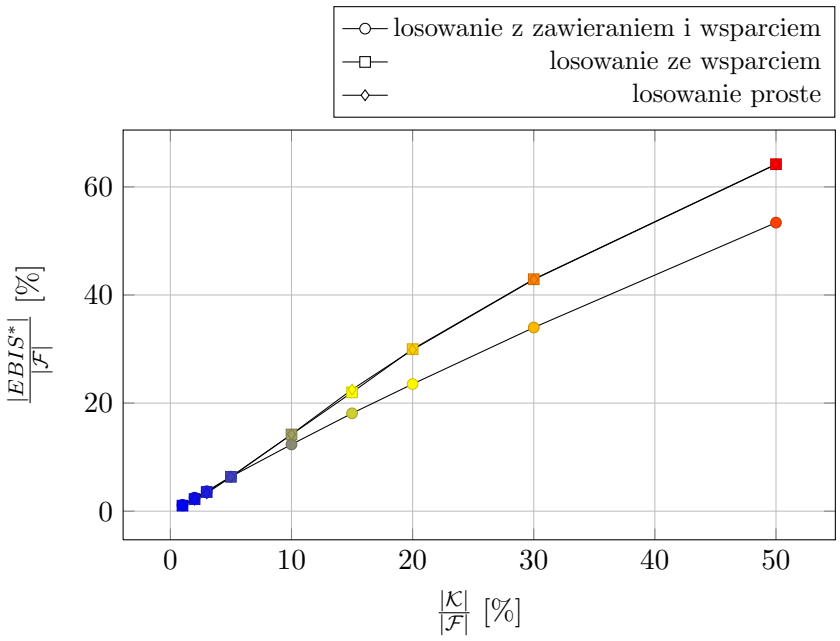
Podobnie do algorytmów `EIS` i `SIS`, algorytm `EBIS*` powinien być w stanie wyprowadzić nowe wzorce nie będące wzorcami ograniczonymi przez wzorce znane. Wyniki rozszerzania najdłuższych i skracania najkrótszych wzorców znanych uzyskane dla algorytmu `EBIS*` są identyczne z wynikami uzyskanymi kolejno dla algorytmów `EIS` (tabele 7.7 i 7.8) oraz `SIS` (tabele 7.13 i 7.14).



Rysunek 7.13: Skuteczność algorytmu EBIS*: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert EBIS\rvert$	$\lvert EBIS^*\rvert$	$\frac{\lvert EBIS^*\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\frac{\lvert EBIS^*\rvert}{\lvert EBIS\rvert}$ [%]
1	2188	2252	2535	1.159	112.567
2	4375	4511	5463	2.497	121.104
3	6563	6758	8078	3.693	119.532
5	10939	11245	13971	6.386	124.242
10	21877	22589	27016	12.349	119.598
15	32816	33767	39609	18.106	117.301
20	43754	44929	51462	23.524	114.541
30	65631	67396	74331	33.977	110.29
50	109385	111422	116800	53.39	104.827

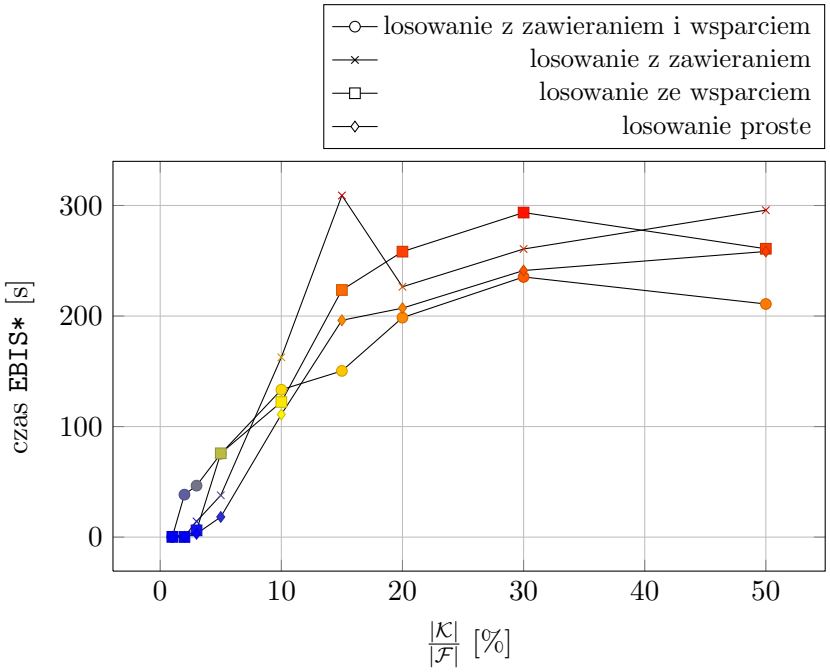
Tabela 7.18: Skuteczność algorytmu EBIS* i porównanie z EBIS: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem



Rysunek 7.14: Skuteczność algorytmu EBIS*: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	liczba iteracji	średni czas iteracji [s]	czas `EBIS*` [s]
1	4	0.063	0.274
2	6	6.399	38.419
3	5	9.308	46.56
5	5	15.075	75.392
10	6	22.216	133.322
15	5	30.069	150.367
20	5	39.71	198.572
30	5	47.065	235.349
50	4	52.716	210.884

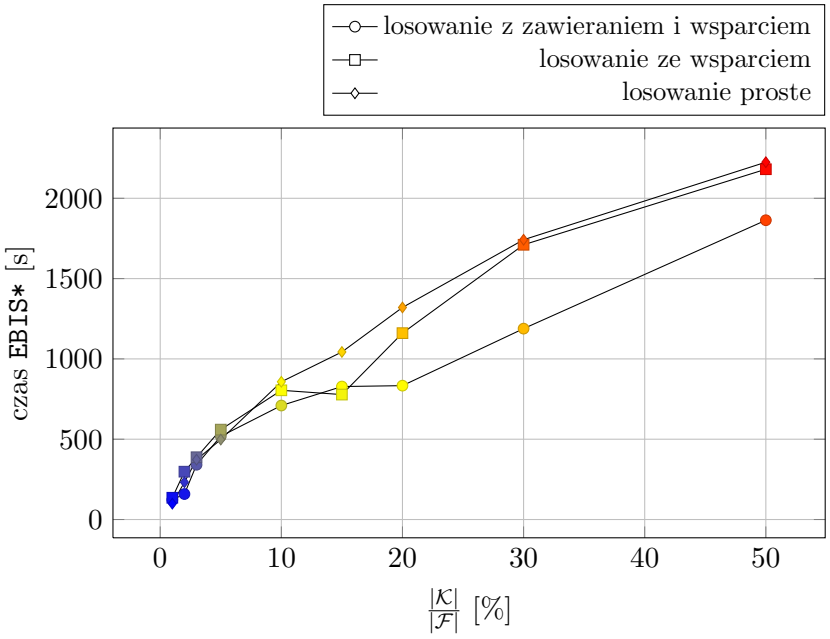
Tabela 7.19: Wydajność algorytmu EBIS*: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem



Rysunek 7.15: Wydajność algorytmu EBIS*: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)

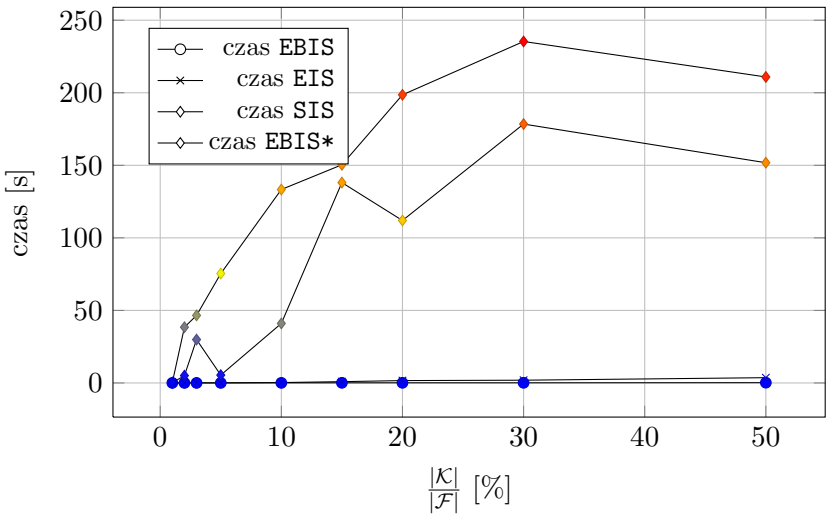
$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	liczba iteracji	średni czas iteracji [s]	czas EBIS* [s]
1	7	17.592	123.147
2	5	31.727	158.636
3	6	56.839	341.041
5	7	74.045	518.313
10	6	118.29	709.745
15	6	137.989	827.941
20	5	166.693	833.471
30	5	237.697	1188.495
50	5	372.669	1863.359

Tabela 7.20: Wydajność algorytmu EBIS*: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem

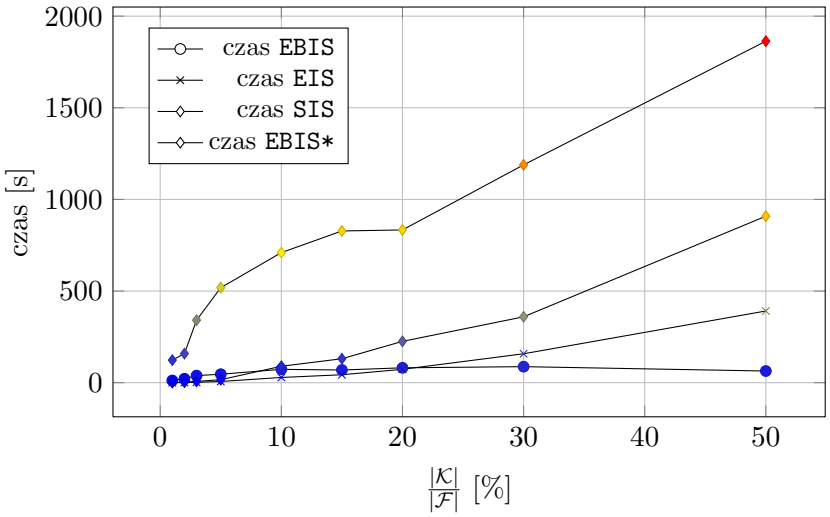


Rysunek 7.16: Wydajność algorytmu EBIS*: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

Największa skuteczność algorytmu `EBIS*` ze wszystkich dotychczas omówionych okupiona jest jego najmniejszą wydajnością. Wyniki pomiarów przedstawiono w tabelach 7.19 oraz 7.20. Z uwagi na fakt, że algorytm `EBIS*` jest złożeniem algorytmów `EBIS`, `EIS` i `SIS`, czas wykonywania obliczeń jest większy od czasu poszczególnych algorytmów składowych dla obu zbiorów i wszystkich wielkości prób, co zostało uwidocznione na wykresach 7.17 i 7.18. Dla wzorców znanych o dużym stopniu korelacji losowanych ze wszystkich wzorców częstych zbioru *mushroom*, czas wyznaczania wzorców skurczonych dla większych wielkości prób ma dominujący wpływ na całkowity czas obliczeń algorytmu `EBIS*`. W przypadku mniej skorelowanych wzorców zbioru *kosarak*, wpływ ten nadal jest duży, choć istotny jest także czas pracy algorytmu `EIS`. W obu przypadkach wyznaczanie wzorców dokładnie ograniczonych to niewielki ułamek czasu algorytmu `EBIS*`.



Rysunek 7.17: Porównanie wydajności algorytmów EBIS, EIS, SIS i EBIS*: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem



Rysunek 7.18: Porównanie wydajności algorytmów EBIS, EIS, SIS i EBIS*: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem

7.2 Badanie algorytmów wyznaczania zwięzłych reprezentacji

Podstawowym celem stosowania zwięzłych reprezentacji do opisu wiedzy jest zapisanie jej w zwartym formacie wymagającym mniejszej ilości pamięci, a także przyspieszenie obliczeń. W skrajnych przypadkach wykorzystanie zwięzłych reprezentacji może umożliwić odkrycie wiedzy, której bez ich zastosowania nie dałoby się wyprowadzić ze względu na nieakceptowalny czas obliczeń lub niemożliwe do zaspokojenia zapotrzebowanie na pamięć. W niniejszym rozdziale przedstawiono wyniki eksperymentów, które miały zweryfikować w jakim stopniu oba te aspekty są realizowane przez reprezentacje wiedzy niepełnej R i R^*.

Do badania zwięzłych reprezentacji wykorzystano identyczne zbiory wzorców jak w punkcie 7.1.

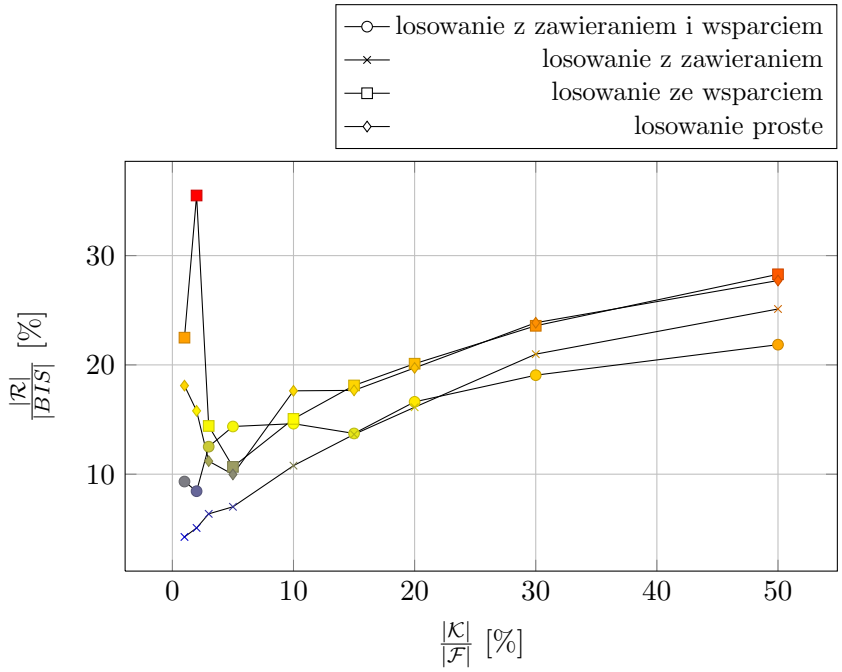
7.2.1 Badanie algorytmu R

Wyniki eksperymentów badających zwięzłość reprezentacji R przedstawiono w tabelach 7.21 oraz 7.22.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert BIS\rvert$	$\frac{\lvert\mathcal{R}\rvert}{\lvert BIS\rvert}$ [%]
1	65	61	654	9.327
2	131	123	1457	8.442
3	196	169	1350	12.519
5	327	288	2005	14.364
10	655	515	3522	14.622
15	982	680	4954	13.726
20	1309	898	5403	16.62
30	1964	1114	5845	19.059
50	3274	1376	6297	21.852

Tabela 7.21: Zwięzłość reprezentacji $\mathcal{R}$: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

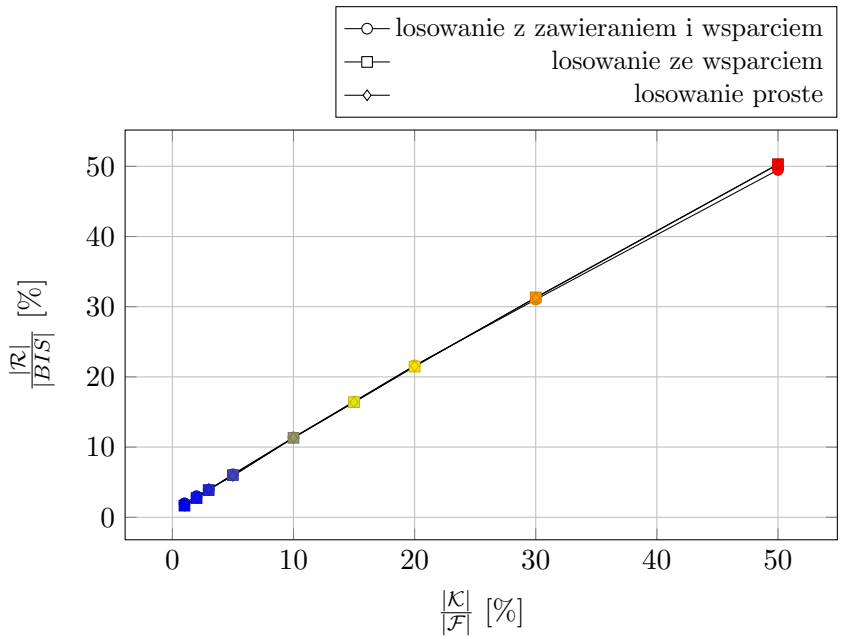
Dla zbioru *mushroom* wielkość reprezentacji R stanowi od około 8,5% wszystkich wzorców ograniczonych do prawie 22% w zależności od wielkości próby, podczas gdy reprezentacja wzorców losowanych ze zbioru *kosarak* to od około 2% do 50% wzorców ograniczonych. Oceniając natomiast zwięzłość reprezentacji względem liczby wzorców znanych można zauważyć, że w przypadku zbioru *mushroom* jest ona zauważalnie większa niż dla zbioru



Rysunek 7.19: Zwięzłość reprezentacji $\mathcal{R}$: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert BIS\rvert$	$\frac{\lvert\mathcal{R}\rvert}{\lvert BIS\rvert}$ [%]
1	2188	2184	109893	1.987
2	4375	4359	144899	3.008
3	6563	6539	167014	3.915
5	10939	10888	177633	6.13
10	21877	21556	189494	11.376
15	32816	32340	196103	16.491
20	43754	42880	198253	21.629
30	65631	63658	205208	31.021
50	109385	104767	211704	49.488

Tabela 7.22: Zwięzłość reprezentacji $\mathcal{R}$: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem



Rysunek 7.20: Zwięzłość reprezentacji $\mathcal{R}$: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

kosarak. Wynika to z faktu większej korelacji wzorców zbioru *mushroom*, a tym samym mniejszej relatywnej liczby generatorów i zbiorów zamkniętych. Dla zbioru *kosarak* wielkość reprezentacji R jest niewiele mniejsza od liczby wzorców znanych.

W przypadku obu zbiorów wraz ze wzrostem wielkości próby zwięzłość reprezentacji maleje, co widać na wykresach 7.19 i 7.20. Zależność ta dla zbioru *kosarak* jest prawie liniowa niezależnie od przyjętej metody losowania, podczas gdy w przypadku zbioru *mushroom* zmniejszanie zwięzłości następuje powoli.

Porównanie czasu wyznaczania reprezentacji R do czasu wyznaczania wzorców ograniczonych przedstawiono w tabelach 7.23 i 7.24 (wartość 0 wynika z zaokrąglenia zmierzonego czasu do trzech miejsc po przecinku). Wyznaczanie reprezentacji jest znacznie mniej złożone czasowo od wyznaczania wzorców ograniczonych, co wynika z istotnie różnej konstrukcji algorytmów `BIS` i `R`. Podczas gdy algorytm `BIS` wyznacza wzorce ograniczone oraz szacuje ich wsparcia, algorytm `R` wyszukuje generatory i zbiory zamknięte wśród wzorców znanych.

Warto także zauważyć, że korzyść czasową z zastosowania reprezentacji R widać szczególnie w przypadku mało skorelowanych wzorców zbioru *ko-*

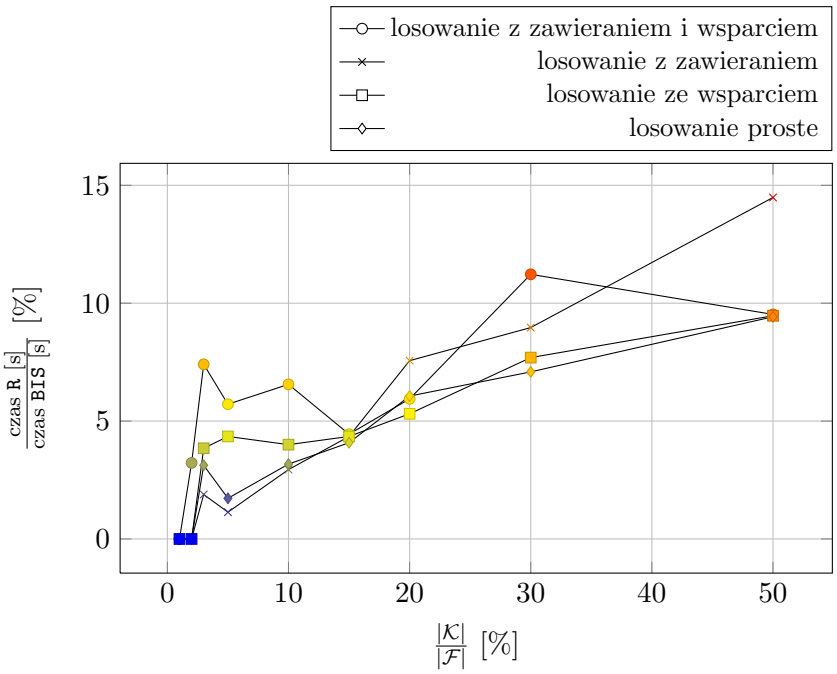
sarak. Jest tak, ponieważ jak zauważono w punkcie 7.1.1, wyznaczanie przybliżonych wartości wsparć dla zbioru wzorców ograniczonych o niewielkiej korelacji (tzn. zbiorów ograniczonych, których tylko niewielka część jest dokładnie ograniczona), jest bardziej czasochłonne niż w przypadku wzorców silnie skorelowanych. Jednocześnie czas szacowania wsparć to istotna część algorytmu `BIS`. Z tego względu zysk z wykorzystania zwięzłej reprezentacji dla zbioru *kosarak* jest zazwyczaj większy niż dla zbioru *mushroom*.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	czas `R` [s]	czas `BIS` [s]	$\frac{\text{czas } \texttt{R} \text{ [s]}}{\text{czas } \texttt{BIS} \text{ [s]}}$ [%]
1	0	0.035	0
2	0.001	0.031	3.226
3	0.002	0.027	7.407
5	0.002	0.035	5.714
10	0.004	0.061	6.557
15	0.004	0.09	4.444
20	0.006	0.101	5.941
30	0.011	0.098	11.225
50	0.018	0.189	9.524

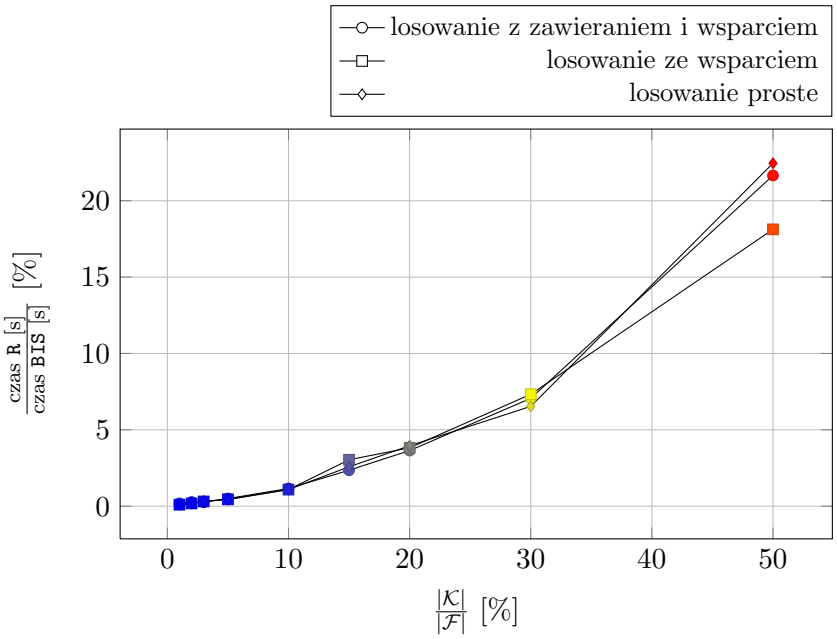
Tabela 7.23: Wydajność algorytmu R: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	czas `R` [s]	czas `BIS` [s]	$\frac{\text{czas } \texttt{R} \text{ [s]}}{\text{czas } \texttt{BIS} \text{ [s]}}$ [%]
1	0.021	12.091	0.174
2	0.057	21.518	0.265
3	0.1	38.683	0.259
5	0.231	46.241	0.5
10	0.844	72.893	1.158
15	1.63	69.217	2.355
20	2.957	81.243	3.64
30	6.179	87.507	7.061
50	13.792	63.693	21.654

Tabela 7.24: Wydajność algorytmu R: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem



Rysunek 7.21: Wydajność algorytmu R: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)



Rysunek 7.22: Wydajność algorytmu R: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

7.2.2 Badanie algorytmu R*

W tabelach 7.25 i 7.26 przedstawiono wyniki eksperymentów badających zwięzłość reprezentacji R^*. Dla zbioru wzorców mocno skorelowanych *mushroom* zwięzłość rozumiana jako stosunek wielkości reprezentacji R^* do liczby wszystkich wzorców dokładnie ograniczonych $EBIS^*$, pomijając najmniejsze wielkości prób, jest względnie stała i wynosi około 20%. W przypadku wzorców *kosarak* zwięzłość jest dużo mniejsza, a ponadto zmienia się wraz ze zmianą wielkości próby. Różnice widać wyraźnie na wykresach 7.23 i 7.24.

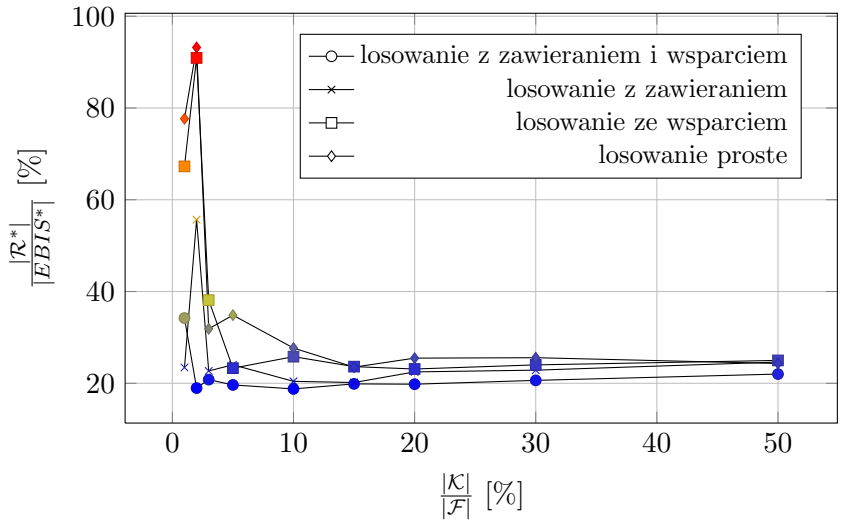
$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{R}^*\rvert$	$\lvert EBIS^*\rvert$	$\frac{\lvert\mathcal{R}^*\rvert}{\lvert EBIS^*\rvert}$ [%]
1	65	65	190	34.211
2	131	146	770	18.961
3	196	187	899	20.801
5	327	316	1609	19.64
10	655	531	2828	18.777
15	982	723	3639	19.868
20	1309	860	4343	19.802
30	1964	1067	5171	20.634
50	3274	1307	5938	22.011

Tabela 7.25: Zwięzłość reprezentacji $\mathcal{R}^*$: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

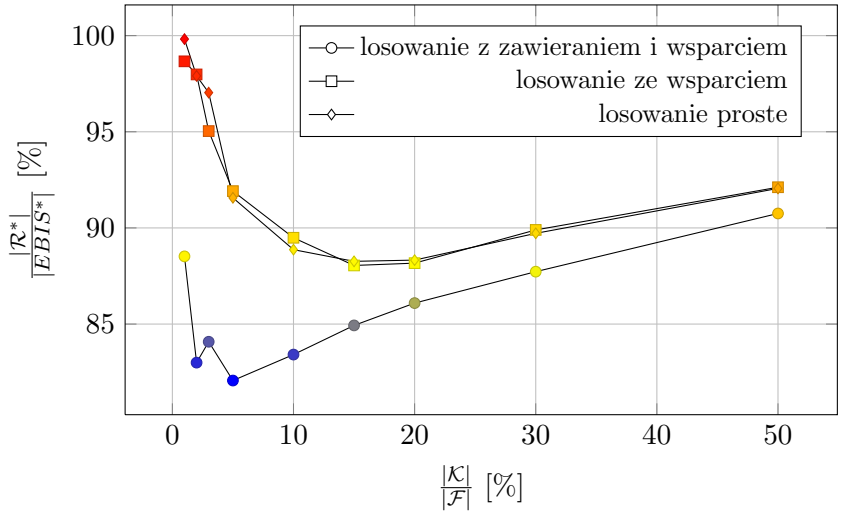
$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{R}^*\rvert$	$\lvert EBIS^*\rvert$	$\frac{\lvert\mathcal{R}^*\rvert}{\lvert EBIS^*\rvert}$ [%]
1	2188	2244	2535	88.521
2	4375	4534	5463	82.995
3	6563	6792	8078	84.08
5	10939	11465	13971	82.063
10	21877	22535	27016	83.414
15	32816	33640	39609	84.93
20	43754	44304	51462	86.091
30	65631	65207	74331	87.725
50	109385	105998	116800	90.752

Tabela 7.26: Zwięzłość reprezentacji $\mathcal{R}^*$: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem

Warto ponadto zauważyć (wykres 7.24), że dla zbioru *kosarak* oraz metody losowania wzorców uwzględniającej relację zawierania i wartości wspar-



Rysunek 7.23: Zwięzłość reprezentacji $\mathcal{R}^*$: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)



Rysunek 7.24: Zwięzłość reprezentacji $\mathcal{R}^*$: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

cia, zwięzłość reprezentacji R^* była wyraźnie większa niż dla prób losowanych pozostałymi metodami.

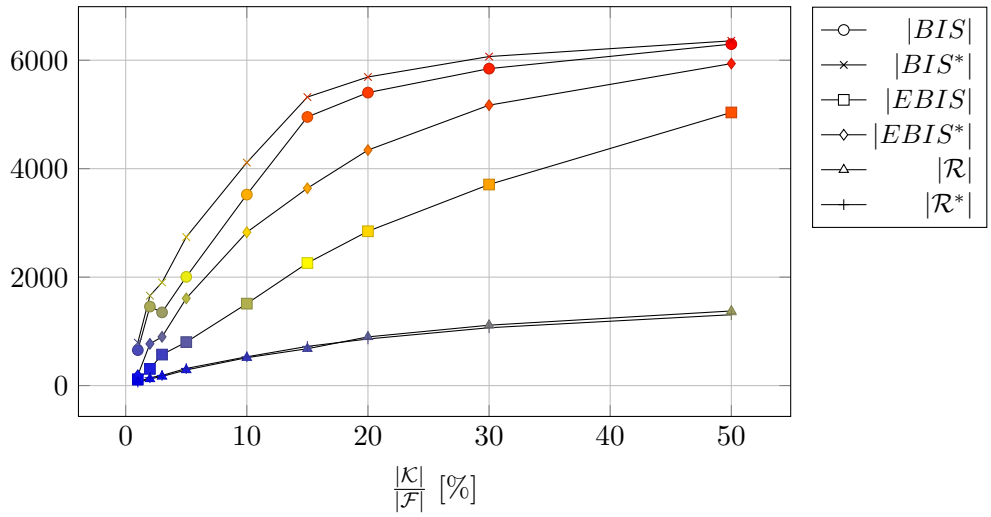
W tabelach 7.27 i 7.28 oraz na wykresach 7.25 i 7.26 przestawiono porównanie liczności zbiorów $\mathcal{K}$, $BIS(\mathcal{K})$, $BIS^*(\mathcal{K})$, $EBIS(\mathcal{K})$, $EBIS^*(\mathcal{K})$, $\mathcal{R}(\mathcal{K})$ oraz $\mathcal{R}^*(\mathcal{K})$.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert BIS\rvert$	$\lvert BIS^*\rvert$	$\lvert EBIS\rvert$	$\lvert EBIS^*\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{R}^*\rvert$
1	65	654	792	117	190	61	65
2	131	1457	1657	308	770	123	146
3	196	1350	1903	572	899	169	187
5	327	2005	2739	802	1609	288	316
10	655	3522	4113	1511	2828	515	531
15	982	4954	5321	2260	3639	680	723
20	1309	5403	5691	2846	4343	898	860
30	1964	5845	6068	3708	5171	1114	1067
50	3274	6297	6358	5037	5938	1376	1307

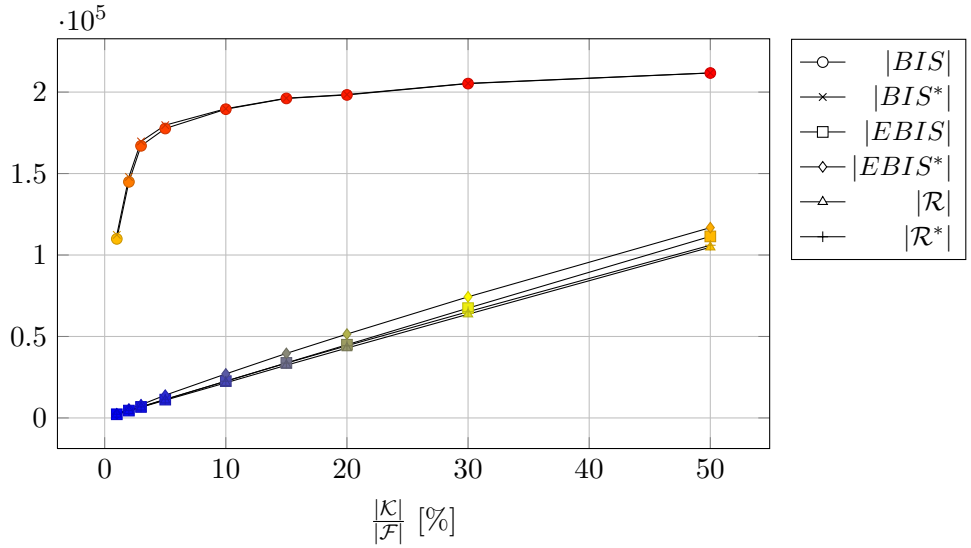
Tabela 7.27: Porównanie algorytmów wnioskowania i zwięzłości reprezentacji: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert BIS\rvert$	$\lvert BIS^*\rvert$	$\lvert EBIS\rvert$	$\lvert EBIS^*\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{R}^*\rvert$
1	2188	109893	112345	2252	2535	2184	2244
2	4375	144899	147909	4511	5463	4359	4534
3	6563	167014	169704	6758	8078	6539	6792
5	10939	177633	179688	11245	13971	10888	11465
10	21877	189494	189787	22589	27016	21556	22535
15	32816	196103	196248	33767	39609	32340	33640
20	43754	198253	198483	44929	51462	42880	44304
30	65631	205208	205350	67396	74331	63658	65207
50	109385	211704	211763	111422	116800	104767	105998

Tabela 7.28: Porównanie algorytmów wnioskowania i zwięzłości reprezentacji: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem



Rysunek 7.25: Porównanie algorytmów wnioskowania i zwięzłości reprezentacji: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem



Rysunek 7.26: Porównanie algorytmów wnioskowania i zwięzłości reprezentacji: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem

Analizując wyniki zamieszczone w tabelach 7.29 i 7.30 można zauważyć, że czas obliczeń algorytmu `R*` jest krótszy od czasu obliczeń algorytmu `EBIS*`. W tym przypadku korzyść z zastosowania reprezentacji jest szczegól-

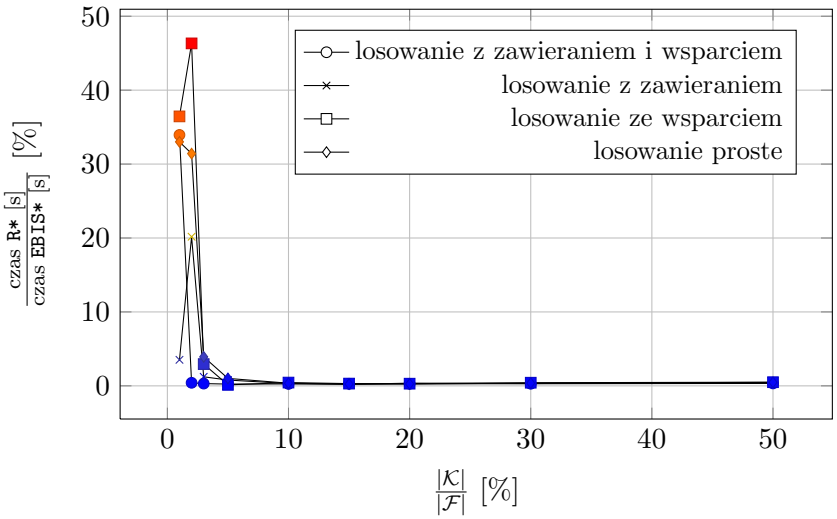
nie widoczna dla zbioru wzorców *mushroom*, gdzie poza próbą o wielkości 1%, wyznaczanie reprezentacji R^* to ułamek procenta czasu wymaganego na wyznaczenie wszystkich wzorców dokładnie ograniczonych $EBIS^*$. Korzyść dla zbioru *kosarak* jest mniejsza, choć dla niewielkich wielkości prób algorytm `R*` wykonuje się ponad 50 razy szybciej od algorytmu `EBIS*`.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	czas `R*` [s]	czas `EBIS*` [s]	$\frac{\text{czas R* [s]}}{\text{czas EBIS* [s]}}$ [%]
1	0.093	0.274	33.942
2	0.159	38.419	0.414
3	0.143	46.56	0.307
5	0.182	75.392	0.241
10	0.348	133.322	0.261
15	0.426	150.367	0.283
20	0.56	198.572	0.282
30	0.675	235.349	0.287
50	0.693	210.884	0.329

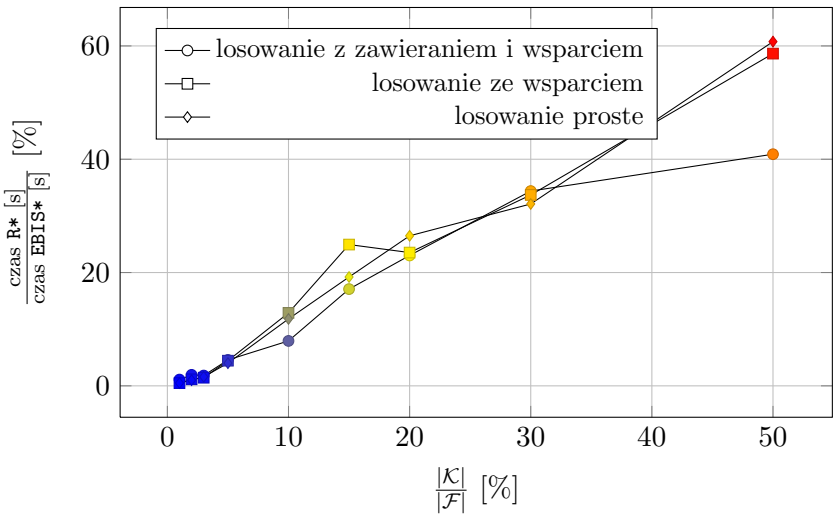
Tabela 7.29: Wydajność algorytmu R*: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$), losowanie z zawieraniem i wsparciem

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	czas `R*` [s]	czas `EBIS*` [s]	$\frac{\text{czas R* [s]}}{\text{czas EBIS* [s]}}$ [%]
1	1.363	123.147	1.107
2	3.083	158.636	1.943
3	6.199	341.041	1.818
5	23.705	518.313	4.574
10	56.287	709.745	7.931
15	141.389	827.941	17.077
20	191.798	833.471	23.012
30	408.647	1188.495	34.384
50	761.852	1863.359	40.886

Tabela 7.30: Wydajność algorytmu R*: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem



Rysunek 7.27: Wydajność algorytmu R*: zbiór *mushroom*, $minSup = 2000$ ($|\mathcal{F}| = 6548$)



Rysunek 7.28: Wydajność algorytmu R*: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$)

W punkcie 7.1 wykazano, że względna wydajność algorytmów `EBIS`, `EIS` oraz `SIS` jest bardzo różna dla zbiorów mocno skorelowanych wzorców. W szczególności algorytm `EIS` nie był w stanie wyznaczyć w ciągu sześciu godzin wzorców rozszerzonych dla próby wielkości 10% pobranej ze wszystkich wzorców częstych zbioru *mushroom* przy minimalnym wsparciu równym

1600 ($|\mathcal{F}| = 53838$). Proces wyznaczania wzorców skurczonych z wykorzystaniem algorytmu `SIS` dla jeszcze mniejszej próby, tzn. 10% wzorców częstych *mushroom* przy minimalnym wsparciu 1800 ($|\mathcal{F}| = 13692$), również został przerwany po około sześciu godzinach. Dla tych samych warunków obliczeń, wyznaczenie wzorców ograniczonych za pomocą algorytmu `BIS` trwało nie więcej niż kilka sekund. Ponieważ algorytm `EBIS*` jest złożeniem algorytmów `EBIS`, `EIS` oraz `SIS`, w ustalonym sześciogodzinnym limicie czasowym nie było możliwości wyznaczenia wzorców $EBIS^*$ ani BIS^* dla wyżej podanych warunków obliczeń.

W niniejszym punkcie postanowiono sprawdzić, czy wykorzystanie zwięzłej reprezentacji R^* umożliwi wyznaczenie wzorców $EBIS^*$ oraz BIS^*, przy czym do testu wybrano większe próby niż wymienione w poprzednim akapicie, tzn. losowano wzorce częste spośród wszystkich wzorców częstych zbioru *mushroom* wyznaczonych dla minimalnego wsparcia równego 1000 ($|\mathcal{F}| = 122864$). Dla tak wyznaczonych prób wyznaczono reprezentacje R^*, a następnie wygenerowano wzorce $EBIS^*$ i BIS^* jako odpowiednio $EBIS(R^*)$ oraz $BIS(R^*)$. Wyniki zamieszczono w tabelach 7.31 oraz 7.32. Czas całkowity, na który składało się wyznaczanie reprezentacji oraz wyznaczanie wzorców (dokładnie) ograniczonych, w zależności od wielkości próby wynosił od około 12 do niespełna 30 sekund. Widać zatem, że skorzystanie ze zwięzłej reprezentacji R^* umożliwiło sprawne przeprowadzenie obliczeń, których nie dało się wykonać poprzez bezpośrednie zastosowanie algorytmu `EBIS*` do wzorców znanych.

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$\lvert\mathcal{K}\rvert$	$\lvert BIS\rvert$	$\lvert BIS(\mathcal{R}^*)\rvert$	$\lvert EBIS\rvert$	$\lvert EBIS(\mathcal{R}^*)\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{R}^*\rvert$
1	1229	61723	68259	44253	56735	800	1118
2	2457	79480	84806	60844	72368	1178	1590
3	3686	87232	93733	65671	80404	1751	2313
5	6143	97877	103170	74124	89727	2393	2716
10	12286	110397	114726	84463	101023	3893	3858
15	18429	113656	116535	92530	107596	5082	4694
20	24573	118333	120799	97490	112551	5902	5122
30	36859	120656	121898	105487	116848	7020	5978
50	61432	121925	122544	114441	120671	8262	7087

Tabela 7.31: Porównanie algorytmów wnioskowania i zwięzłości reprezentacji: zbiór *mushroom*, $minSup = 1000$ ($|\mathcal{F}| = 122864$), losowanie z zawieraniem i wsparciem

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	czas `R*` [s]	czas `BIS(R*)` [s]	czas całkowity [s]
1	7.541	4.518	12.059
2	8.732	5.162	13.894
3	14.094	5.433	19.527
5	10.715	5.295	16.01
10	12.708	5.664	18.372
15	16.336	5.928	22.264
20	17.195	5.515	22.71
30	16.176	6.07	22.246
50	23.355	5.663	29.018

Tabela 7.32: Czas wykonania algorytmów R* i BIS: zbiór *mushroom*, $minSup = 1000$ ($|\mathcal{F}| = 122864$), losowanie z zawieraniem i wsparciem

7.3 Badanie algorytmów wyznaczania liczności warstw

Odmiennie niż w przypadku algorytmów przebadanych w poprzednich punktach, gdzie wybierano próby losowe z warstwy wszystkich wzorców częstych przy ustalonym minimalnym progu wsparcia, do analizy algorytmów wyznaczających liczności warstw posłużono się zbiorami wzorców definiującymi istotnie różniące się od siebie warstwy. Analogicznie jak wcześniej wykorzystano zbiory danych *mushroom* i *kosarak*, niemniej warstwy zostały zdefiniowane na trzy różne sposoby w zależności od typu:

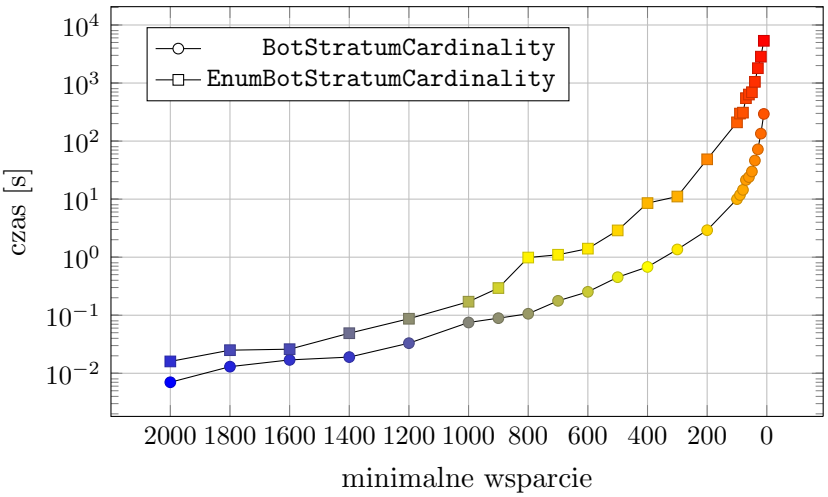
- dla warstw przywiązanych do podstawy:
  - granica dolna: zbiór pusty,
  - granica górna: wszystkie maksymalne wzorce częste wyznaczone dla kilkunastu wartości minimalnego wsparcia: od 2000 do 10 dla zbioru *mushroom* oraz od 8000 do 800 dla zbioru *kosarak*,
- dla warstw przywiązanych do szczytu:
  - granica dolna: wszystkie minimalne wzorce rzadkie wyznaczone dla kilkunastu wartości minimalnego wsparcia: od 8000 do 1000 dla zbioru *mushroom* oraz od 8000 do 1000 dla zbioru *kosarak*,
  - granica górna: zbiór reprezentujący uniwersum bazy danych: 119 pozycji dla zbioru *mushroom* oraz 41270 pozycji dla zbioru *kosarak*,
- dla warstw dowolnych:
  - granica dolna: wszystkie maksymalne wzorce częste wyznaczone dla ustalonej wartości minimalnego wsparcia: 2000 dla zbioru *mushroom* oraz 6000 dla zbioru *kosarak*,
  - granica górna: tylko te maksymalne wzorce częste wyznaczone dla kilkunastu wartości minimalnego wsparcia: od 1800 do 10 dla zbioru *mushroom* oraz od 5500 do 800 dla zbioru *kosarak*, dla których istnieje co najmniej jeden podzbiór w granicy dolnej.

Zbiory maksymalnych wzorców częstych wygenerowano przy użyciu dostępnej w repozytorium FIMI implementacji algorytmu LCM ver. 2 [34]. Do utworzenia zbiorów minimalnych wzorców rzadkich wykorzystano algorytm GR-Apriori [18] zaimplementowany w aplikacji $\mathcal{K}$-*miner*.

7.3.1 Badanie algorytmów wyznaczania liczności warstw przywiązanych do podstawy

Wyniki eksperymentów badających algorytmy wyznaczania liczności warstw przywiązanych do podstawy zebrano w tabelach 7.33 i 7.34. We wszystkich przypadkach algorytmy `BotStratumCardinality` oraz `EnumBotStratumCardinality` zwróciły tę samą liczność równą faktycznej liczbie wszystkich wzorców częstych zbiorów *mushroom* i *kosarak* dla podanych wartości minimalnego wsparcia.

Dla zbioru *mushroom* algorytm `BotStratumCardinality` obliczał liczność znacznie szybciej od algorytmu `EnumBotStratumCardinality`. Odwrotnie zależność zachodziła w przypadku zbioru *kosarak*. Pomijając mało liczne warstwy, dla których obliczenia obu algorytmów trwały podobnie długo, dla większych warstw `EnumBotStratumCardinality` wyznaczał ich liczność w krótszym czasie niż `BotStratumCardinality`. W szczególności dla minimalnego wsparcia równego 800, algorytm `BotStratumCardinality` nie wyznaczył liczności warstwy w ciągu 10 godzin, po których obliczenia zostały przerwane.



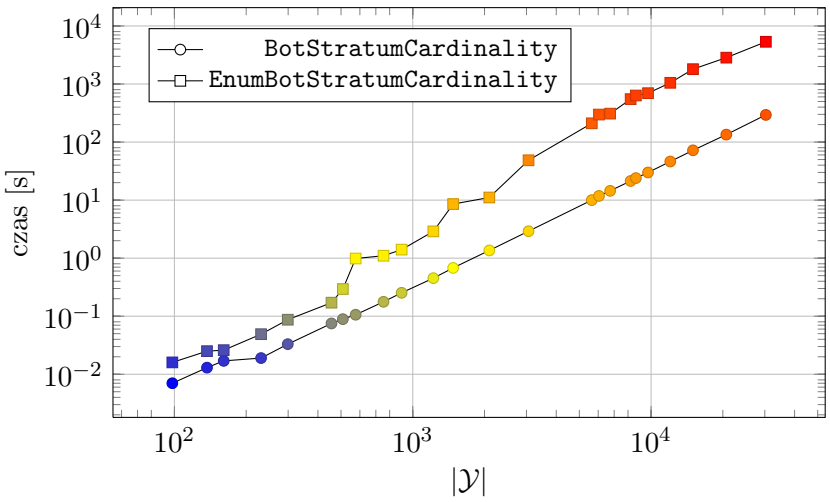
Rysunek 7.29: Zależność czasu wyznaczania $|[\{\emptyset\}, \mathcal{Y}]|$ od minimalnego wsparcia: zbiór *mushroom*

W ogólności na podstawie uzyskanych wyników można zaobserwować, że złożoność obliczeniowa obu algorytmów jest w przybliżeniu liniowa względem liczby wzorców stanowiących górną granicę warstwy. Fakt ten zobrazowano na wykresach 7.30 i 7.32.

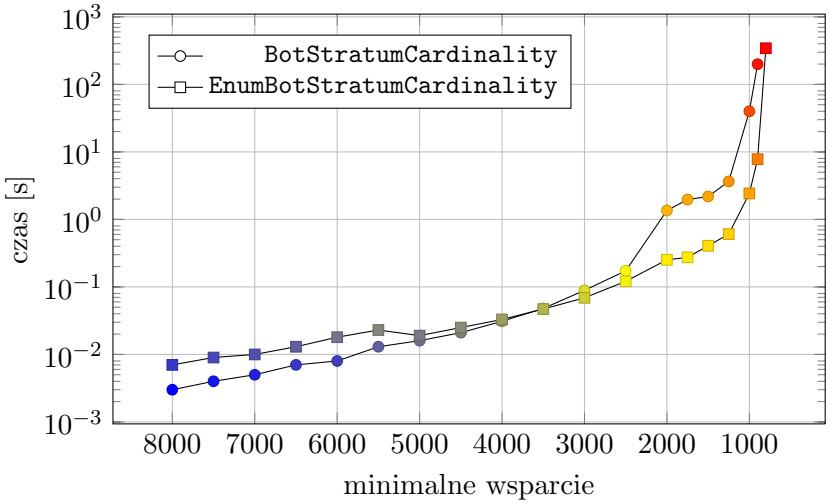
Wśród przedstawionych w rozdziale 5 motywacji wyznaczania liczno-

minimalne wsparcie	$\lvert\mathcal{Y}\rvert$	$\lvert[\{\emptyset\},\mathcal{Y}]\rvert$	czas `BotStratum Cardinality` [s]	czas `EnumBotStratum Cardinality` [s]
2000	98	6548	0.007	0.016
1800	137	13692	0.013	0.025
1600	161	53838	0.017	0.026
1400	231	59344	0.019	0.049
1200	299	100378	0.033	0.087
1000	456	122864	0.075	0.171
900	509	163302	0.089	0.294
800	576	575420	0.106	0.986
700	754	633172	0.177	1.101
600	898	941526	0.253	1.403
500	1220	1441722	0.452	2.892
400	1477	3757180	0.679	8.548
300	2095	5251450	1.359	11.091
200	3061	18006022	2.915	48.554
100	5639	65867804	10.001	210.412
90	6044	89812162	11.804	297.733
80	6733	91273270	14.446	308.93
70	8208	153262492	21.369	549.059
60	8634	176436682	23.938	634.098
50	9709	197924358	29.925	695.132
40	12037	294138132	46.208	1049.348
30	14990	505019936	72.1	1802.602
20	20681	779945280	134.444	2846.572
10	30306	1661196716	293.728	5341.418

Tabela 7.33: Wyznaczanie liczności warstw $[\{\emptyset\}, \mathcal{Y}]$: zbiór *mushroom*



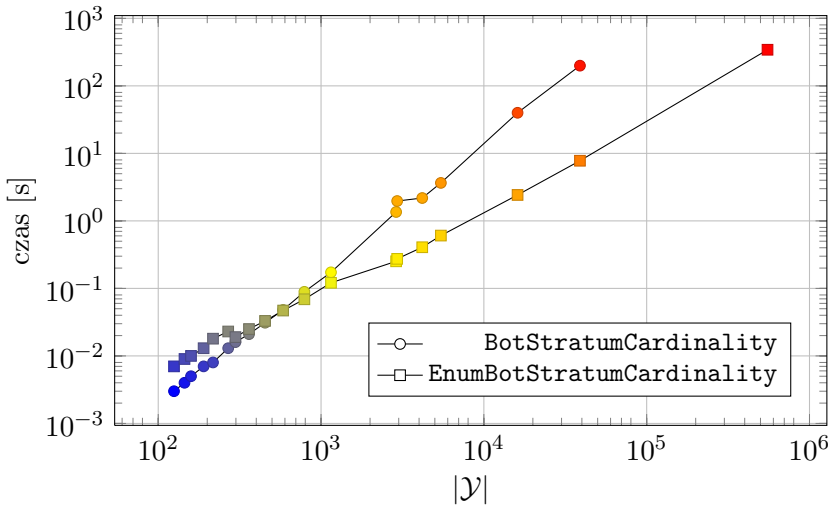
Rysunek 7.30: Zależność czasu wyznaczania $|[\{\emptyset\}, \mathcal{Y}]|$ od $|\mathcal{Y}|$: zbiór *mushroom*



Rysunek 7.31: Zależność czasu wyznaczania $|[\{\emptyset\}, \mathcal{Y}]|$ od min. wsparcia: zbiór *kosarak*

minimalne wsparcie	$\lvert\mathcal{Y}\rvert$	$\lvert[\{\emptyset\},\mathcal{Y}]\rvert$	czas `BotStratum Cardinality` [s]	czas `EnumBotStratum Cardinality` [s]
8000	125	576	0.003	0.007
7500	145	660	0.004	0.009
7000	159	771	0.005	0.01
6500	190	917	0.007	0.013
6000	217	1111	0.008	0.018
5500	269	1345	0.013	0.023
5000	298	1592	0.016	0.019
4500	361	1957	0.021	0.025
4000	453	2470	0.031	0.033
3500	585	3379	0.048	0.047
3000	790	4889	0.089	0.069
2500	1151	8554	0.173	0.121
2000	2892	34201	1.355	0.252
1750	2935	145180	1.969	0.274
1500	4183	218766	2.181	0.407
1250	5445	326906	3.653	0.606
1000	16110	706428	39.941	2.422
900	38926	1577386	199.159	7.781
800	552307	36538731	NaN	342.77

Tabela 7.34: Wyznaczanie liczności warstw $[\{\emptyset\},\mathcal{Y}]$: zbiór *kosarak*



Rysunek 7.32: Zależność czasu wyznaczania $\lvert[\{\emptyset\},\mathcal{Y}]\rvert$ od $\lvert\mathcal{Y}\rvert$: zbiór *kosarak*

ści warstw podano między innymi możliwość obliczenia liczby wszystkich wzorców częstych przy danych częstych wzorach maksymalnych, a także ocenę zwięzłości reprezentacji opartej na zbiorach zamkniętych. Z tego względu przeprowadzono eksperyment porównawczy, mający na celu zweryfikowanie, czy zaimplementowane algorytmy `BotStratumCardinality` oraz `EnumBotStratumCardinality` umożliwiają wykonanie obu tych operacji w czasie krótszym od czasu wyznaczania wszystkich wzorców częstych. W tym celu zmierzono czas wyznaczania maksymalnych, zamkniętych oraz wszystkich wzorców częstych przy pomocy jednego z najszybszych dostępnych algorytmów LCM ver. 2 [34][4] oraz czas wyznaczania wszystkich wzorców częstych za pomocą algorytmu Apriori [3] zaimplementowanym w aplikacji *$\mathcal{K}$-miner*.

Wyniki dla zbioru *mushroom* zestawiono w tabeli 7.35. Dla wszystkich przebadanych wartości minimalnego wsparcia algorytm `BotStratumCardinality` (szybszy dla zbioru *mushroom* od `EnumBotStratumCardinality`) obliczał liczbę wszystkich wzorców częstych w oparciu o zbiór częstych wzorców maksymalnych w czasie krótszym niż czas wyznaczania tych wzorców. Algorytm Apriori wykrywał wzorce częste stosunkowo wolno. Różnica między algorytmem `BotStratumCardinality` i LCM ver. 2 była natomiast mniejsza i malała wraz ze zmniejszaniem wartości wsparcia, niemniej w całym przebadanym zakresie udało się osiągnąć omówiony w poprzednim akapicie cel.

Podobnie jak dla zbioru *mushroom*, w przypadku zbioru *kosarak* algorytm Apriori każdorazowo generował wszystkie wzorce częste wolniej niż trwało obliczanie ich liczności za pomocą algorytmu `EnumBotStratumCardinality` (szybszym od `BotStratumCardinality` dla zbioru *kosarak*), co widać w tabeli 7.36. Czas pracy algorytmu LCM ver. 2 był również dłuższy od czasu obliczeń algorytmu `EnumBotStratumCardinality`, ale tylko do minimalnej wartości wsparcia równej 1000. Poniżej tej wartości czas algorytmu `EnumBotStratumCardinality` wzrasta szybciej od LCM ver. 2, w efekcie czego obliczenia tego pierwszego dla małych wartości wsparcia trwają dłużej[5].

[4]Pomiary algorytmu LCM ver. 2 wykonano korzystając z dostępnej w repozytorium FIMI implementacji na tym samym komputerze PC co wszystkie pozostałe eksperymenty, ale pod kontrolą systemu operacyjnego Ubuntu 10.04.

[5]Interesujące byłoby porównanie wyników dla wartości wsparć mniejszych od 800. Przeprowadzenie takiego eksperymentu nie było jednak możliwe, ponieważ pobrana z repozytorium FIMI implementacja LCM ver. 2 przy tak niskich wartościach wsparcia przerywała obliczenia z błędem.

minimalne wsparcie	czas `BotStratum Cardinality` [s]	czas Apriori [s]	czas LCM v.2 $\mathcal{F}$ [s]	czas LCM v.2 $MAX(\mathcal{F})$ [s]	czas LCM v.2 $\mathcal{FC}$ [s]
2000	0.007	6.268	0.033	0.034	0.033
1800	0.013	11.966	0.038	0.041	0.037
1600	0.017	39.241	0.05	0.045	0.039
1400	0.019	39.858	0.062	0.054	0.047
1200	0.033	63.407	0.085	0.07	0.057
1000	0.075	83.982	0.086	0.094	0.069
900	0.089	111.325	0.092	0.096	0.071
800	0.106	277.004	0.192	0.104	0.076
700	0.177	309.13	0.686	0.117	0.09
600	0.253	411.773	1.006	0.129	0.094
500	0.452	620.152	1.159	0.166	0.114
400	0.679	1180.386	3.884	0.188	0.129
300	1.359	1500.079	6.247	0.222	0.148
200	2.915	NaN	20.976	0.286	0.214
100	10.001	NaN	88.801	0.42	0.268
90	11.804	NaN	90.856	0.447	0.288
80	14.446	NaN	92.171	0.466	0.299
70	21.369	NaN	106.059	0.506	0.325
60	23.938	NaN	114.443	0.527	0.337
50	29.925	NaN	117.596	0.559	0.351
40	46.208	NaN	161.15	0.647	0.384
30	72.1	NaN	195.754	0.696	0.43
20	134.444	NaN	248.563	0.755	0.471
10	293.728	NaN	486.728	0.884	0.588

Tabela 7.35: Porównanie czasu wyznaczania $|[\{\emptyset\}, MAX(\mathcal{F})]|$ oraz $\mathcal{F}$, $\mathcal{FC}$ i $MAX(\mathcal{F})$: zbiór *mushroom*

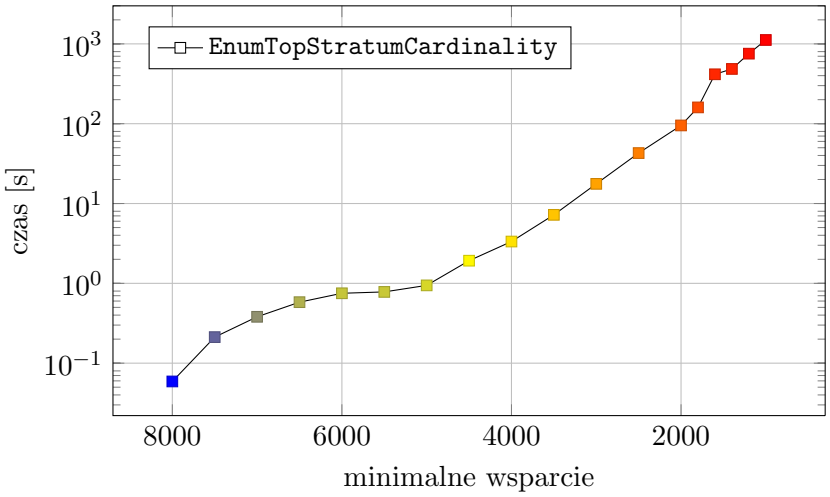
minimalne wsparcie	czas `EnumBotStratum Cardinality` [s]	czas Apriori [s]	czas LCM v.2 $\mathcal{F}$ [s]	czas LCM v.2 $MAX(\mathcal{F})$ [s]	czas LCM v.2 $\mathcal{FC}$ [s]
8000	0.007	6.106	2.622	1.972	1.896
7500	0.009	6.598	1.792	2.054	1.961
7000	0.01	7.433	1.824	2.153	2.056
6500	0.013	8.167	1.909	2.226	2.118
6000	0.018	9.39	1.921	2.332	2.134
5500	0.023	11.284	1.927	2.46	2.323
5000	0.019	13.402	2.151	2.64	2.431
4500	0.025	16.46	2.256	2.81	2.52
4000	0.033	21.425	2.26	2.949	2.627
3500	0.047	29.544	2.362	3.227	2.789
3000	0.069	40.429	2.52	3.548	3.029
2500	0.121	65.375	2.672	3.994	3.277
2000	0.252	153.577	3.082	4.853	3.784
1750	0.274	363.551	3.417	5.645	4.413
1500	0.407	510.24	3.692	6.469	4.898
1250	0.606	723.303	4.098	7.696	5.516
1000	2.422	1294.952	5.049	10.026	7.209
900	7.781	NaN	6.419	12.003	9.129
800	342.77	NaN	81.073	25.633	44.683

Tabela 7.36: Porównanie czasu wyznaczania $|[\{\emptyset\}, MAX(\mathcal{F})]|$ oraz $\mathcal{F}$, $\mathcal{FC}$ i $MAX(\mathcal{F})$: zbiór *kosarak*

7.3.2 Badanie algorytmów wyznaczania liczności warstw przywiązanych do szczytu

W niniejszym punkcie przedstawiono wyniki eksperymentów wyznaczania liczności warstw przywiązanych do szczytu, których górnym ograniczeniem jest wzorzec będący uniwersum, zaś dolnym wszystkie minimalne rzadkie wzorce dla różnych wartości wsparć. Na warstwy te składają się więc wszystkie wzorce rzadkie, których liczba dla badanych przypadków jest bardzo duża.

Poniżej przedstawiono wyniki działania algorytmu `EnumTopStratumCardinality` dla warstw skonstruowanych w oparciu o wzorce zbioru *mushroom*. Pozostałe eksperymenty okazały się niemożliwe do przeprowadzenia przy przyjętym sposobie konstruowania warstw testowych. Obliczenia algorytmu `TopStratumCardinality` dla zbioru *mushroom* i wartości minimalnego wsparcia równej 8000 zostały przerwane po około ośmiu godzinach. Próba wyznaczenia liczności warstwy dla minimalnego wsparcia równego 7500 również nie zakończyła się sukcesem. W przypadku warstw wzorców wyznaczonych ze zbioru *kosarak*, oba algorytmy nie były w stanie wyznaczyć liczności warstw. W tym przypadku problemem okazało się bardzo szybkie wykorzystanie całej dostępnej pamięci operacyjnej, a w konsekwencji przerwanie obliczeń.



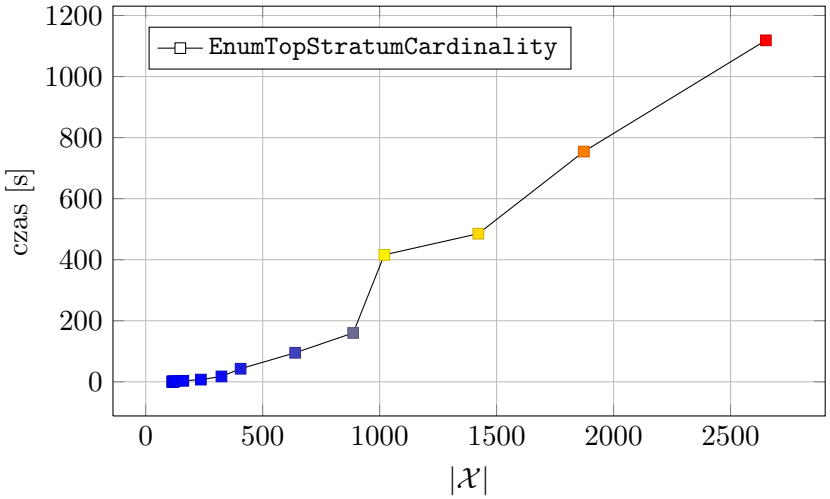
Rysunek 7.33: Zależność czasu wyznaczania $|[\mathcal{X},\{I\}]|$ od min. wsparcia: zbiór *mushroom*

Wyniki uzyskane dla zbioru *mushroom* zestawiono w tabeli 7.37. Algorytm `EnumTopStratumCardinality` poprawnie wyznaczył liczbę wszystkich wzorców rzadkich w każdym testowanym przypadku. Warto zauważyć,

minimalne wsparcie	$\lvert\mathcal{X}\rvert$	$\lvert[\mathcal{X},\{I\}]\rvert$	czas `TopStratum Cardinality` [s]	czas `EnumTopStratum Cardinality` [s]
8000	118	66461399789245793645190353014017228 6	NaN	0.059
7500	116	66461399789245793645190353014017228 0	NaN	0.212
7000	115	66461399789245793645190353014017227 2	NaN	0.381
6500	115	66461399789245793645190353014017226 4	NaN	0.581
6000	114	66461399789245793645190353014017225 6	NaN	0.751
5500	117	66461399789245793645190353014017225 4	NaN	0.781
5000	119	66461399789245793645190353014017224 6	NaN	0.943
4500	128	66461399789245793645190353014017219 0	NaN	1.922
4000	161	66461399789245793645190353014017212 0	NaN	3.34
3500	236	66461399789245793645190353014017191 8	NaN	7.216
3000	324	66461399789245793645190353014017135 6	NaN	17.638
2500	406	66461399789245793645190353014016993 0	NaN	42.855
2000	639	66461399789245793645190353014016574 0	NaN	95.094
1800	887	66461399789245793645190353014015859 6	NaN	160.019
1600	1020	66461399789245793645190353014011845 0	NaN	415.947
1400	1422	66461399789245793645190353014011294 4	NaN	485.256
1200	1873	66461399789245793645190353014007191 0	NaN	754.234
1000	2651	66461399789245793645190353014004942 4	NaN	1118.619

Tabela 7.37: Wyznaczanie liczności warstw $\lvert[\mathcal{X},\{I\}]\rvert$: zbiór *mushroom*

że czas obliczeń rośnie wraz ze zmniejszaniem się liczności badanej warstwy, choć faktyczna zależność między czasem obliczeń a licznością warstwy jest trudna do określenia, ponieważ względna różnica między licznością największej i najmniejszej analizowanej warstwy jest bardzo mała. Widać jednak, że w przebadanym zakresie złożoność obliczeniowa algorytmu `EnumTopStratumCardinality` zależy w dużym stopniu od liczności dolnej granicy warstwy (rysunek 7.34).



Rysunek 7.34: Zależność czasu wyznaczania $|[\mathcal{X},\{I\}]|$ od $|\mathcal{X}|$: zbiór *mushroom*

Alternatywna metoda wyznaczania liczby wzorców częstych i rzadkich

Przy znajomości uniwersum, problem wyznaczania liczności wszystkich wzorców częstych w oparciu o zbiór maksymalnych wzorców częstych jest tożsamy problemowi wyznaczania wszystkich wzorców rzadkich w oparciu o zbiór minimalnych wzorców rzadkich. Wynika to z następującej równości:

$$|[\{\emptyset\},\{I\}]| = |\{\emptyset\}, MAX(\mathcal{F})| + |[GBd^{-},\{I\}]|.$$

Liczności warstw $|\{\emptyset\}, MAX(\mathcal{F})|$ wyznaczane były w punkcie 7.3.1, zaś warstw $|[GBd^{-},\{I\}]|$ w punkcie 7.3.2. Korzystając z danych uzyskanych w tych punktach dla zbioru *mushroom*, w tabeli 7.38 przedstawiono czas wyznaczania liczby wszystkich wzorców częstych za pomocą algorytmu `BotStratumCardinality` oraz alternatywną metodą z wykorzystaniem algorytmu `TopStratumCardinality`. Analogiczne porównanie dla problemu

wyznaczania wszystkich wzorców rzadkich pokazano w tabeli 7.39. Czas wyznaczania $|[\{\emptyset\}, \{I\}]|$ z użyciem funkcji `LatticeCardinality` po zaokrągleniu do trzech miejsc po przecinku wynosił 0.

minimalne wsparcie	$\lvert\{\emptyset\}, MAX(\mathcal{F})\rvert$ czas [s]	$\lvert[\{\emptyset\}, \{I\}]\rvert - \lvert[GBd^-, \{I\}]\rvert$ czas [s]
2000	0.007	95.094
1800	0.013	160.019
1600	0.017	415.947
1400	0.019	485.256
1200	0.033	754.234
1000	0.075	1118.619

Tabela 7.38: Czas wyznaczania liczby wszystkich wzorców częstych dla kilku wartości minimalnego wsparcia za pomocą algorytmu BotStratumCardinality i alternatywną metodą z użyciem algorytmu EnumTopStratumCardinality: zbiór *mushroom*

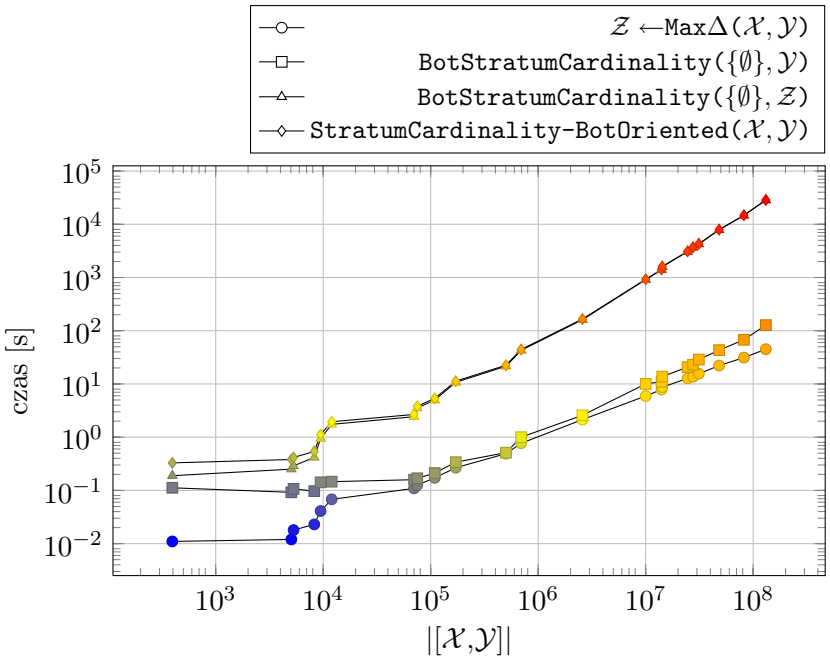
minimalne wsparcie	$\lvert[GBd^-, \{I\}]\rvert$ czas [s]	$\lvert[\{\emptyset\}, \{I\}]\rvert - \lvert\{\emptyset\}, MAX(\mathcal{F})\rvert$ czas [s]
2000	95.094	0.007
1800	160.019	0.013
1600	415.947	0.017
1400	485.256	0.019
1200	754.234	0.033
1000	1118.619	0.075

Tabela 7.39: Czas wyznaczania liczby wszystkich wzorców rzadkich dla kilku wartości minimalnego wsparcia za pomocą algorytmu EnumTopStratumCardinality i alternatywną metodą z użyciem algorytmu BotStratumCardinality: zbiór *mushroom*

Z powyższego zestawienia widać, że w przypadku małych wartości minimalnego wsparcia, problem wyznaczania liczby wszystkich wzorców rzadkich korzystniej jest przekształcić do problemu wyznaczania liczby wszystkich wzorców częstych.

7.3.3 Badanie algorytmów wyznaczania liczności dowolnych warstw

Wyniki eksperymentów badających algorytm wyznaczania liczności dowolnej warstwy `StratumCardinality-BotOriented` przedstawiono w tabelach 7.40 oraz 7.41. Oprócz całkowitego czasu obliczeń zaprezentowano także czas wykonywania najważniejszych etapów algorytmu, a także czas obliczeń funkcji $\texttt{Max}\Delta(\mathcal{X},\mathcal{Y})$ trzeba alternatywnymi metodami. Czasy $\texttt{BotStratumCardinality}(\mathcal{Y})$, $\texttt{BotStratumCardinality}(\mathcal{Z})$ oraz czas całkowity zmierzono przy wyznaczaniu wartości funkcji $\texttt{Max}\Delta(\mathcal{X},\mathcal{Y})$ z wykorzystaniem algorytmu $\texttt{ExpansionMax}\Delta$.



Rysunek 7.35: Zależność czasu wyznaczania $|[\mathcal{X},\mathcal{Y}]|$ od $|[\mathcal{X},\mathcal{Y}]|$: zbiór *mushroom*

Wyznaczanie wartości funkcji $\texttt{Max}\Delta(\mathcal{X},\mathcal{Y})$ najszybciej wykonywane jest przy pomocy algorytmu $\texttt{ExpansionMax}\Delta$, który jest nieznacznie bardziej wydajny od algorytmu $\texttt{Max}\Delta$. Metoda naiwna sprawdza się jedynie dla warstw o niewielkiej liczności, co szczególnie widać w wynikach dla zbioru *mushroom*.

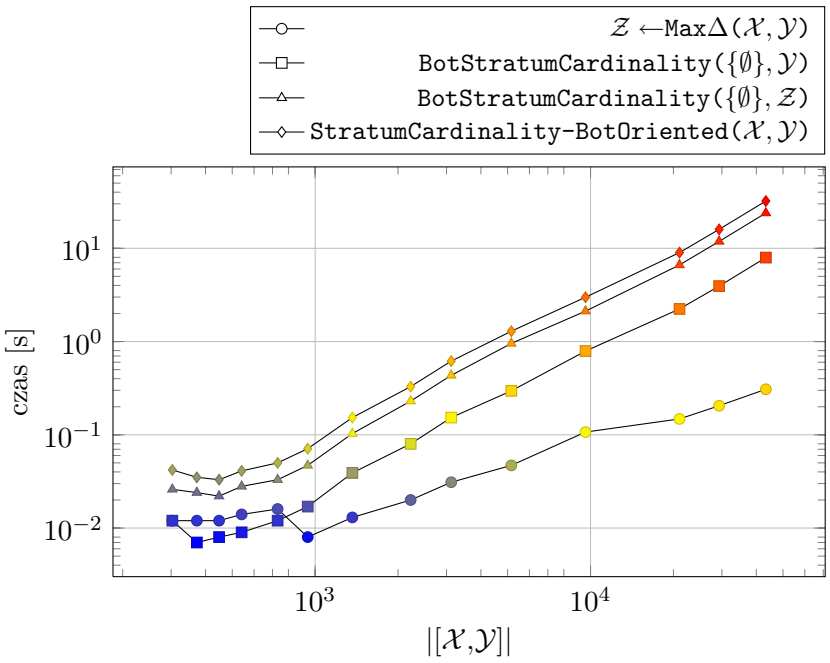
Jeżeli funkcja $\texttt{Max}\Delta(\mathcal{X},\mathcal{Y})$ nie jest wyznaczana metodą naiwną, czas jej obliczania stanowi niewielką część całkowitego czasu obliczeń algorytmu `StratumCardinality-BotOriented`. W takiej sytuacji dominują ope-

minimalne wsparcie dla $\mathcal{Y}$	$\lvert\mathcal{Y}\rvert$	`Max`Δ [s]	`Expansion` `Max`Δ [s]	`Naive` `Max`Δ [s]	$\lvert\mathcal{Z}\rvert$	`BotStratum` `Cardinality` $\mathcal{Y}$ [s]	`BotStratum` `Cardinality` $\mathcal{Z}$ [s]	$\lvert[\mathcal{X},\mathcal{Y}]\rvert$	czas całkowity [s]
1800	80	0.011	0.017	0.509	426	0.112	0.189	395	0.329
1600	84	0.012	0.009	NaN	445	0.092	0.251	5056	0.381
1400	124	0.018	0.013	NaN	620	0.107	0.289	5300	0.416
1200	149	0.023	0.017	NaN	748	0.097	0.417	8260	0.539
1000	297	0.041	0.032	NaN	1432	0.142	0.942	9459	1.122
900	368	0.068	0.052	NaN	2050	0.146	1.742	11996	1.952
800	388	0.109	0.083	NaN	2263	0.159	2.425	70045	2.674
700	541	0.126	0.104	NaN	3015	0.169	3.534	75056	3.817
600	659	0.172	0.137	NaN	3768	0.212	4.99	109117	5.356
500	953	0.267	0.225	NaN	5683	0.339	10.729	171239	11.3
400	1180	0.49	0.421	NaN	8016	0.513	21.55	501854	22.489
300	1770	0.769	0.65	NaN	12115	1	42.77	694436	44.491
200	2746	2.14	1.737	NaN	23522	2.58	161.208	2582591	165.594
100	5315	5.938	5.35	NaN	57231	10.039	911.182	10012188	926.603
90	5789	7.85	6.999	NaN	71401	11.004	1393.98	14051674	1412.035
80	6486	8.688	7.473	NaN	76518	13.78	1588.821	14274056	1610.124
70	7988	12.726	11.188	NaN	105549	20.709	3064.146	24547785	3096.109
60	8425	13.675	12.486	NaN	116379	23.007	3668.509	27583530	3704.065
50	9464	15.531	13.772	NaN	124995	28.715	4248.102	31247119	4290.658
40	11840	22.155	20.26	NaN	171280	43.161	7849.908	48260064	7913.41
30	14858	31.25	28.351	NaN	234067	67.623	14666.484	82047559	14762.654
20	20545	45.02	40.877	NaN	324196	127.452	28413.931	131254142	28594.424
10	30230	75.507	67.44	NaN	488068	299.632	NaN	NaN	NaN

Tabela 7.40: Wyznaczanie liczności warstw $\lvert[\mathcal{X},\mathcal{Y}]\rvert$: zbiór *mushroom*, minimalne wsparcie dla $\mathcal{X} = 2000$, $\lvert\mathcal{X}\rvert = 98$

minimalne wsparcie dla $\mathcal{Y}$	$\lvert\mathcal{Y}\rvert$	`Max`Δ [s]	`Expansion` `Max`Δ [s]	`Naive` `Max`Δ [s]	$\lvert\mathcal{Z}\rvert$	`BotStratum` `Cardinality` $\mathcal{Y}$ [s]	`BotStratum` `Cardinality` $\mathcal{Z}$ [s]	$\lvert[\mathcal{X},\mathcal{Y}]\rvert$	czas całkowity [s]
5500	210	0.012	0.003	0.004	375	0.012	0.026	303	0.042
5000	197	0.012	0.003	0.004	369	0.007	0.024	372	0.035
4500	193	0.012	0.004	0.005	358	0.008	0.022	448	0.033
4000	209	0.014	0.004	0.007	385	0.009	0.028	541	0.041
3500	236	0.016	0.005	0.012	431	0.012	0.033	731	0.05
3000	283	0.008	0.006	0.015	513	0.017	0.047	940	0.071
2500	456	0.013	0.011	0.025	804	0.039	0.103	1365	0.153
2000	653	0.02	0.018	0.049	1221	0.08	0.23	2223	0.329
1750	949	0.031	0.028	0.097	1776	0.153	0.434	3120	0.617
1500	1404	0.047	0.04	29.751	2625	0.296	0.955	5157	1.291
1250	2310	0.107	0.078	29.838	4295	0.791	2.119	9591	2.989
1000	4118	0.148	0.128	32.665	7494	2.237	6.63	21044	8.997
900	5664	0.205	0.181	33.95	10166	3.933	11.854	29316	15.97
800	8070	0.308	0.262	38.332	14249	7.951	23.91	43321	32.128

Tabela 7.41: Wyznaczanie liczności warstw $\lvert[\mathcal{X},\mathcal{Y}]\rvert$: zbiór *kosarak*, minimalne wsparcie dla $\mathcal{X} = 6000$, $\lvert\mathcal{X}\rvert = 217$



Rysunek 7.36: Zależność czasu wyznaczania $|[\mathcal{X},\mathcal{Y}]|$ od $|[\mathcal{X},\mathcal{Y}]|$: zbiór *kosarak*

racje wyznaczania za pomocą algorytmu `BotStratumCardinality` liczności warstw $[\{\emptyset\},\mathcal{Y}]$ oraz $[\{\emptyset\},\mathcal{Z}]$, a przede wszystkim tej drugiej. Wynika to z faktu, że liczność górnej granicy $\mathcal{Z}$ warstwy nadmiarowej w każdym przypadku była większa od liczności górnej granicy $\mathcal{Y}$ badanej warstwy, a jak pokazano w punkcie 7.3.1, czas obliczeń algorytmu `BotStratumCardinality` jest proporcjonalny do liczby wzorców definiujących górną granicę warstwy przywiązanej do podstawy.

Ze względu na to, że całkowity czas obliczeń jest zdominowany przez algorytm `BotStratumCardinality`, analogicznie jak `BotStratumCardinality` algorytm `StratumCardinality-BotOriented`, przy ustalonej dolnej granicy warstwy, wykazuje w przybliżeniu liniową złożoność obliczeniową względem liczby wzorców należących do jej górnej granicy.

Ostatnim przeprowadzonym eksperymentem było porównanie czasu wyznaczania liczby wzorców ograniczonych z użyciem algorytmu `BotStratumCardinality` oraz czasu wyznaczania tych wzorców przy pomocy algorytmu `BIS` dla warstw przebadanych w punkcie 7.1. Uzyskane wyniki przedstawiono w tabelach 7.42 i 7.43. Czas wyznaczania liczności warstwy każdorazowo był krótszy od czasu wyznaczania warstwy i stanowił od około 45% w najgorszym do niecałych 1,5% w najlepszym przypadku czasu

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$BIS(\mathcal{K})$ czas [s]	$BIS(\mathcal{R})$ czas [s]	$BIS(\mathcal{R}^*)$ czas [s]	$\lvert BIS(\mathcal{K})\rvert$ czas [s]	$\lvert BIS(\mathcal{R})\rvert$ czas [s]	$\lvert BIS(\mathcal{R}^*)\rvert$ czas [s]
1	4.048	3.832	4.518	1.37	1.347	0.974
2	4.946	5.191	5.162	0.742	0.802	0.655
3	8.016	7.329	5.433	1.123	1.135	0.745
5	6.563	6.014	5.295	0.546	0.589	0.472
10	6.402	5.257	5.664	0.413	0.449	0.219
15	8.195	7.117	5.928	0.436	0.46	0.259
20	8.462	6.419	5.515	0.223	0.234	0.08
30	8.771	6.93	6.07	0.221	0.238	0.099
50	13.877	6.152	5.663	0.268	0.279	0.102

Tabela 7.42: Porównanie czasów wyznaczania warstwy i czasu wyznaczania liczności warstwy dla danych z punktu 7.1: zbiór *mushroom*, $minSup = 1000$ ($|\mathcal{F}| = 122864$), losowanie z zawieraniem i wsparciem

$\frac{\lvert\mathcal{K}\rvert}{\lvert\mathcal{F}\rvert}$ [%]	$BIS(\mathcal{K})$ czas [s]	$BIS(\mathcal{R})$ czas [s]	$BIS(\mathcal{R}^*)$ czas [s]	$\lvert BIS(\mathcal{K})\rvert$ czas [s]	$\lvert BIS(\mathcal{R})\rvert$ czas [s]	$\lvert BIS(\mathcal{R}^*)\rvert$ czas [s]
1	12.091	12.924	12.856	5.653	5.715	4.866
2	21.518	21.248	23.552	5.549	6.108	4.167
3	38.683	43.33	44.367	2.9	3.289	2.076
5	46.241	49.504	55.205	4.978	4.898	2.195
10	72.893	67.424	69.199	3.494	3.638	3.363
15	69.217	66.182	69.945	2.274	2.299	2.376
20	81.243	74.576	74.968	3.18	3.088	2.6
30	87.507	83.523	76.422	2.327	2.391	2.072
50	63.693	58.483	60.625	2.324	2.404	2.22

Tabela 7.43: Porównanie czasów wyznaczania warstwy i czasu wyznaczania liczności warstwy dla danych z punktu 7.1: zbiór *kosarak*, $minSup = 1500$ ($|\mathcal{F}| = 218766$), losowanie z zawieraniem i wsparciem

wyznaczania warstwy.

Rozdział 8

Podsumowanie

W pracy zrealizowano oprogramowanie *$\mathcal{K}$-miner* implementujące wszystkie przedstawione algorytmy, w tym algorytmy wyznaczania wzorców ograniczonych, wzorców rozszerzonych i skurczonych, algorytmy wyznaczania zwięzłych i bezstratnych reprezentacji wiedzy niepełnej, a także algorytmy obliczające liczności warstw wzorców bez konieczności ich wyznaczania. Zaproponowano ponadto prosty język skryptowy $\mathcal{K}$ umożliwiający specyfikację procesu analizy danych w aplikacji *$\mathcal{K}$-miner*.

Badania algorytmów wnioskowania z wiedzy niepełnej wykazały, że za ich pomocą możliwe jest wyprowadzenie znacznej liczby nowych wzorców, a tym samym przeprowadzenia dalszego wnioskowania o charakterze niedostępnych danych źródłowych, przy czym liczba nowych wzorców zależy od stopnia korelacji wzorców znanych. Szczególnie silnym narzędziem jest mechanizm wyznaczania wzorców ograniczonych z wartością wsparcia wyznaczoną w sposób przybliżony, ponieważ jego wykorzystanie umożliwia wyprowadzenie od kilku do kilkunastu razy więcej wzorców niż liczba wzorców znanych. Liczba nowych wzorców dokładnie ograniczonych jest natomiast duża w przypadku zbiorów wzorców znanych o dużym stopniu korelacji. Wykorzystanie dodatkowo operatorów wyprowadzających wzorce rozszerzone i skurczone umożliwia dalsze zwiększenie skuteczności wnioskowania z wiedzy niepełnej, tj. zwiększenie liczby możliwych do wyprowadzenia nowych wzorców ograniczonych oraz dokładnie ograniczonych. Dla niektórych zbiorów wzorców znanych, wyprowadzone wzorce rozszerzone mogą nie być ograniczone przez wzorce znane. Podobna sytuacja może mieć miejsce w przypadku wzorców skurczonych, niemniej tego drugiego zjawiska nie zaobserwowano dla przebadanych zbiorów wzorców znanych.

Pewnym problemem jest duża złożoność obliczeniowa algorytmów `EIS` oraz `SIS`, a szczególnie algorytmu `SIS`, która uwidacznia się dla zbiorów

wzorców o dużym stopniu korelacji. Wynika to z konstrukcji tych algorytmów, tj. konieczności wyznaczenia dla wszystkich wzorców znanych list ich nadzbiorów, a w przypadku algorytmu `SIS` także podzbiorów, o tym samym wsparciu, a następnie przeanalizowania tych zależności. Z tego względu dalszy rozwój tych algorytmów powinien następować w kierunku ich optymalizacji dla zbiorów mocno skorelowanych wzorców.

Ważnym przebadanym aspektem wnioskowania z wiedzy niepełnej była ocena praktycznych korzyści z wykorzystania zwięzłych reprezentacji wiedzy niepełnej. Eksperymenty wykazały, że liczność reprezentacji wzorców ograniczonych oraz reprezentacji wszystkich wzorców dokładnie ograniczonych jest mniejsza od liczby reprezentowanych przez nie wzorców, przy czym zwięzłość reprezentacji zależy od stopnia korelacji wzorców znanych. Najważniejszą korzyścią jest jednak zysk polegający na znacznym skróceniu czasu obliczeń, objawiający się szczególnie dla zbiorów wzorców mocno skorelowanych. Wykorzystanie zwięzłych reprezentacji umożliwia przeprowadzenie wnioskowania, którego nie dałoby się przeprowadzić wykorzystując operatory wnioskowania z wiedzy niepełnej bezpośrednio na wzorcach znanych. Przedstawione reprezentacje redukują więc w znacznym stopniu problem niewielkiej efektywności algorytmów `EIS` i `SIS`, o którym wspomniano w poprzednim akapicie.

Istotnym poruszonym w pracy zagadnieniem było omówienie metod wydajnego wyznaczania liczności warstw wzorców w taki sposób, aby nie było konieczne wyznaczanie wszystkich wzorców stanowiących te warstwy. Przebadane algorytmy każdorazowo wyznaczały tę liczność poprawnie. W przypadku warstw przywiązanych do podstawy, wyznaczanie liczby wszystkich wzorców częstych przy danych maksymalnych wzorcach częstych i małych wartościach minimalnego wsparcia może zajmować mniej czasu niż wyznaczanie wszystkich wzorców częstych, przy czym zależnie od charakteru danych wyniki szybciej zwraca algorytm `BotStratumCardinality` lub `EnumBotStratumCardinality`. Problematyczne może być natomiast wyznaczanie liczby wzorców rzadkich dla niewielkich wartości minimalnego wsparcia przy pomocy algorytmów wyznaczania liczności warstw przywiązanych do szczytu `TopStratumCardinality` oraz `EnumTopStratumCardinality`. Dla przeanalizowanych warstw, część eksperymentów zakończyła się niepowodzeniem z uwagi na nieakceptowalny czas obliczeń lub zbyt duże zapotrzebowanie na pamięć operacyjną. Wyznaczanie liczności dowolnych warstw z wykorzystaniem algorytmu `StratumCardinality-BotOriented` umożliwia na ogół ocenę liczby wszystkich wzorców, które można wyprowadzić z danego zbioru wzorców znanych, w czasie krótszym niż wyprowadzanie tych wzorców.

Bibliografia

[1] Ramesh C. Agarwal, Charu C. Aggarwal, V. V. V. Prasad. A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, 61(3):350–371, 2001.

[2] Rakesh Agrawal, Tomasz Imielinski, Arun N. Swami. Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, strony 207–216, Washington, D.C., 26–28 1993.

[3] Rakesh Agrawal, Ramakrishnan Srikant. Fast algorithms for mining association rules. *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, strony 487–499. Morgan Kaufmann, 12–15 1994.

[4] Yonatan Aumann, Yehuda Lindell. A statistical theory for quantitative association rules. *Knowledge Discovery and Data Mining*, strony 261–270, 1999.

[5] Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme, Lotfi Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations*, 2(2):66–75, 2000.

[6] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. Joan Peckham, redaktor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, strony 255–264. ACM Press, 05 1997.

[7] Artur Bykowski, Christophe Rigotti. A condensed representation to find frequent patterns. *Symposium on Principles of Database Systems*, 2001.

[8] David S. Coppock. Data mining and modeling: Why lift? *DM Review Online (www.dmreview.com)*, 2002.

[9] Attila Gyenesei. A fuzzy approach for mining quantitative association rules. Raport instytutowy TUCS-TR-336, 17, 2000.

[10] Jiawei Han, Jian Pei, Yiwen Yin. Mining frequent patterns without candidate generation. Weidong Chen, Jeffrey Naughton, Philip A. Bernstein, redaktorzy, *2000 ACM SIGMOD Intl. Conference on Management of Data*, strony 1–12. ACM Press, 05 2000.

[11] Mark Hornic JSR-73 Expert Group. Data Mining API 1.0. `http://www.jcp.org/en/jsr/detail?id=73`, 2004.

[12] Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen, A. Inkeri Verkamo. Finding interesting rules from large sets of discovered association rules. Nabil R. Adam, Bharat K. Bhargava, Yelena Yesha, redaktorzy, *Third International Conference on Information and Knowledge Management (CIKM'94)*, strony 401–407. ACM Press, 1994.

[13] Marzena Kryszkiewicz. *Algorytmy redukcji wiedzy w systemach informacyjnych.* Praca doktorska, Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych, 1994.

[14] Marzena Kryszkiewicz. Representative association rules. *Second Pacific-Asia Conference on Knowledge Discovery and Data Mining, Melbourne, Australia*, strony 198–209, 1998.

[15] Marzena Kryszkiewicz. Inducing theory for the rule set. *Proceedings of the Second International Conference on Rough Sets and Current Trends in Computing, Banff, Canada*, strony 391–398. Springer-Verlag, 2000.

[16] Marzena Kryszkiewicz. Mining with cover and extension operators. *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, Lyon, France*, strony 476–482. Springer-Verlag, 2000.

[17] Marzena Kryszkiewicz. Concise representation of frequent patterns based on disjunction-free generators. *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, strony 305–312, Washington, DC, USA, 2001. IEEE Computer Society.

[18] Marzena Kryszkiewicz. *Concise Representations of Frequent Patterns and Association Rules.* Oficyna Wydawnicza Politechniki Warszawskiej, 2002.

[19] Marzena Kryszkiewicz. Converting generators representation and closed itemsets reptresentations of frequent patterns. Raport instytutowy ICS Research Report 27/02, Politechnika Warszawska, 2002.

[20] Marzena Kryszkiewicz. Inferring knowledge from frequent patterns. *Proceedings of the First International Conference on Computing in an Imperfect World, Belfast, Northern Ireland*, strony 247–262. Springer-Verlag, 2002.

[21] Marzena Kryszkiewicz. Stratification of frequent patterns. Raport instytutowy ICS Research Report 8/2003, Politechnika Warszawska, 2003.

[22] Marzena Kryszkiewicz. Calculating cardinalities of strata. Manuskrypt, 2005.

[23] Marzena Kryszkiewicz, Marcin Gajek. Concise representation of frequent patterns based on generalized disjunction-free generators. *PAKDD '02: Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, strony 159–171, London, UK, 2002. Springer-Verlag.

[24] Heikki Mannila, Hannu Toivonen, A. Inkeri Verkamo. Efficient algorithms for discovering association rules. *AAAI Workshop on Knowledge Discovery in Databases (KDD-94)*, strony 181–192, Seattle, Washington, 1994. AAAI Press.

[25] Jong Soo Park, Ming-Syan Chen, Philip S. Yu. An effective hash based algorithm for mining association rules. Michael J. Carey, Donovan A. Schneider, redaktorzy, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, strony 175–186, San Jose, California, 22–25 1995.

[26] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal. Pruning closed itemset lattices for association rules, 1998. Proceedings of the BDA French Conference on Advanced Databases, October 1998. To appear.

[27] Nicolas Pasquier, Yves Bastide, Rafik Taouil, Lotfi Lakhal. Discovering frequent closed itemsets for association rules. *Lecture Notes in Computer Science*, 1540:398–416, 1999.

[28] Nicolas Pasquier, Yves Bastide, Rafik Taouil, Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, 1999.

[29] J. Pei, G. Dong, W. Zou, J. Han. On computing condensed frequent pattern bases, 2002.

[30] Jian Pei, Jiawei Han, Runying Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, strony 21–30, 2000.

[31] Arnaud Soulet, Bruno Crémilleux. An efficient framework for mining flexible constraints. *PAKDD*, strony 661–671, 2005.

[32] Ramakrishnan Srikant, Rakesh Agrawal. Mining quantitative association rules in large relational tables. H. V. Jagadish, Inderpal Singh Mumick, redaktorzy, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, strony 1–12, Montreal, Quebec, Canada, 4–6 1996.

[33] Ramakrishnan Srikant, Rakesh Agrawal. Mining generalized association rules. *Future Generation Computer Systems*, 13(2–3):161–180, 1997.

[34] Takeaki Uno, Masashi Kiyomi, Hiroki Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. Roberto J. Bayardo Jr., Bart Goethals, Mohammed Javeed Zaki, redaktorzy, *FIMI*, wolumen 126 serii *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[35] Mohammed J. Zaki, Ching-Jui Hsiao. CHARM: An efficient algorithm for closed itemset mining.

Dodatek A

Informacje dla użytkowników oprogramowania

A.1 Wykaz zaimplementowanych procedur

Poniżej przedstawiono wszystkie procedury dostępne w aplikacji *K-miner* oraz dołączonych bibliotekach wraz z krótką informacją o implementowanych przez nie funkcjonalnościach. Dla zachowania zwięzłości, w zestawieniu pominięto opis parametrów procedur oraz dopuszczalnych wartości tych parametrów. Informacje te dostępne są w systemie pomocy aplikacji *K-miner*.

Dla każdej procedury przedstawiono nazwę pełną prezentowaną przez graficzny edytor projektów (6.2.3) oraz nazwę skróconą wykorzystywaną podczas definiowania skryptów w edytorze skryptów (6.2.2) .

Procedury pomocnicze modułu aplikacji głównej

W tabeli A.1 zestawiono procedury pomocnicze zaimplementowane w module aplikacji głównej.

Nazwa	Realizowana funkcjonalność
Options options	Służy do zarządzania opcjami aplikacji oraz drukowania aktualnych wartości opcji w oknie komunikatów. Wszystkie opcje zebrano w tabeli A.2.
macros	Procedura dostępna wyłącznie w edytorze skryptów (6.2.2). Służy do zarządzania zdefiniowanymi makrozmiennymi.
Random sample sample	Umożliwia wybór losowej próby wzorców z podanego zbioru wzorców. Dostępne są cztery metody losowania przedstawione w punkcie 7.1.
Boundaries boundaries	Dla podanej warstwy wzorców wyznacza górną, dolną lub obie granice właściwe.
Universum universum	Dla podanego zbioru wzorców wyznacza wzorzec reprezentujący uniwersum.
Compare compare	Oblicza podstawowe statystyki dwóch zbiorów wzorców oraz przeprowadza test na identyczność tych zbiorów.
RelationaToTransactional rel_to_trans	Umożliwia przekształcenie relacyjnego zbioru danych do transakcyjnego zbioru danych.

Tabela A.1: Procedury pomocnicze modułu aplikacji głównej

Nazwa	Znaczenie
`hash`	Liczba mieszająca drzew mieszających.
`maxlsize`	Maksymalna długość listy wzorców w liściu drzewa mieszającego. Po przekroczeniu tej wartości liść zostanie przekształcony w węzeł wewnętrzny.
`bsearch-l`	Wartość progowa długości list wzorców w drzewie mieszającym, powyżej której wzorce w listach wyszukiwane są algorytmem binarnym. Listy wzorców o długości nieprzekraczającej tej wartości przeszukiwane są algorytmem liniowym.
`bsearch-i`	Wartość progowa długości wzorców, powyżej której pozycje we wzorcach wyszukiwane są algorytmem binarnym. Wzorce o długości nieprzekraczającej tej wartości przeszukiwane są algorytmem liniowym.
`progress`	Określa czy aplikacja ma przetwarzać i prezentować dane o postępie obliczeń. Dotyczy wybranych procedur, które przekazują do aplikacji wymagane informacje.
`beep`	Steruje dźwiękowym sygnalizowaniem o zakończeniu procedury oraz całej analizy.

Tabela A.2: Dostępne opcje aplikacji $\mathcal{K}$-*miner*

Procedury biblioteki alg_apriori.dll

Biblioteka `alg_apriori.dll` zawiera procedury implementujące algorytmy wyznaczania wzorców częstych z bazy danych oraz reprezentacji generatorowej wzorców częstych. Procedury zestawiono w tabeli A.3.

Nazwa	Zaimplementowany algorytm
Apriori apriori	Algorytm Apriori [3] wyznaczania wzorców częstych.
GR-Apriori grapriori	Algorytm GR-Apriori [18] wyznaczania reprezentacji generatorowej wzorców częstych.

Tabela A.3: Procedury biblioteki alg_apriori.dll

Procedury biblioteki alg_partial_knowledge.dll

Biblioteka `alg_partial_knowledge.dll` zawiera procedury implementujące algorytmy wnioskowania z wiedzy niepełnej opisane w rozdziale 3 oraz algorytmy wyznaczania zwięzłych reprezentacji omówione w rozdziale 4. Procedury zestawiono w tabeli A.4.

Nazwa	Zaimplementowany algorytm
Bounded itemsets bis	Algorytm 3.1 wyznaczania wzorców ograniczonych i dokładnie ograniczonych.
Extended itemsets eis	Algorytm 3.8 wyznaczania wzorców rozszerzonych.
Shrunk itemsets sis	Algorytm 3.9 wyznaczania wzorców skurczonych.
Ek Exact bounded itemsets ek	Algorytm 3.13 wyznaczania wszystkich wzorców dokładnie ograniczonych.
R Representation r_rep	Algorytm 4.1 wyznaczania reprezentacji $\mathcal{R}$.
Rk Representation rk_rep	Algorytm 4.2 wyznaczania reprezentacji $\mathcal{R}^*$.

Tabela A.4: Procedury biblioteki alg_partial_knowledge.dll

Procedury biblioteki alg_strata.dll

Biblioteka `alg_strata.dll` zawiera procedury implementujące algorytmy wyznaczania liczności warstw opisane w rozdziale 5. Procedury zestawiono w tabeli A.5.

Nazwa	Zaimplementowany algorytm
LatticeCardinality lattice_card	Algorytm 5.1 wyznaczania liczności kraty.
LatticeGen lattice_gen	Algorytm 5.2 wyznaczania kraty.
BotStratumCardinality bot_stratum_card	Algorytm 5.3 wyznaczania liczności warstwy przywiązanej do podstawy.
EnumBotStratumCardinality enum_bot_stratum_card	Algorytm 5.4 wyznaczania liczności warstwy przywiązanej do podstawy.
BotStratum bot_stratum	Algorytm 5.5 wyznaczania warstwy przywiązanej do podstawy.
LatticeBotStratum lattice_bot_stratum	Algorytm 5.6 wyznaczania warstwy przywiązanej do podstawy.
TopStratumCardinality top_stratum_card	Algorytm 5.7 wyznaczania liczności warstwy przywiązanej do szczytu.
EnumTopStratumCardinality enum_top_stratum_card	Algorytm 5.8 wyznaczania liczności warstwy przywiązanej do szczytu.
TopStratum top_stratum	Algorytm 5.9 wyznaczania warstwy przywiązanej do szczytu.
LatticeTopStratum lattice_top_stratum	Algorytm 5.10 wyznaczania warstwy przywiązanej do szczytu.
StratumCardinalityBotOriented stratum_card_bot_oriented	Algorytm 5.11 wyznaczania liczności dowolnej warstwy.
StratumBotOriented stratum_bot_oriented	Algorytm 5.15 wyznaczania dowolnej warstwy.
LatticeStratum lattice_stratum	Algorytm 5.16 wyznaczania dowolnej warstwy.

Tabela A.5: Procedury biblioteki alg_strata.dll

A.2 Format plików wejściowych i wyjściowych

Aplikacja *K-miner* obsługuje pięć formatów plików służących do przechowywania czterech typów danych:

1. źródłowych baz danych: pliki `*.csv` i `*.dat`,
2. zbiorów wzorców: pliki `*.kis`,
3. definicji skryptów: pliki `*.ksc`,
4. definicji projektów graficznych: pliki `*.kpr`.

Format plików `*.kpr` jest formatem binarnym, natomiast wszystkie pozostałe są formatami tekstowymi.

Pliki źródłowych baz danych *.csv i *.dat

Każdy wiersz w pliku reprezentuje jeden rekord danych. Pola rekordów rozdzielane są:

- w plikach `*.csv` - znakiem przecinka,
- w plikach `*.dat` - dowolnym białym znakiem.

W drugim przypadku następujące po sobie białe znaki są traktowane jak jeden separator. Znakami zakazanymi są znaki separatora. Przykładowy fragment pliku `*.dat`:

```
39 11 27 1 40 6 41 42 43 44 45 46 47 3 48 7 49 50 51
82 6 83 7 84 85 86 87 88
1 6 139 140 90 141 142 143 144
11 6 56 57 58 59 60 61 62 63 64
```

Pliki zbiorów wzorców *.kis

Format plików wzorców `*.kis` jest analogiczny do plików baz danych `*.dat`, przy czym każdy wiersz reprezentuje jeden wzorzec. Ponadto ostatnie pole w wierszu musi zawierać wartość wsparcia wzorca zapisaną w nawiasach okrągłych. W przypadku wzorców, dla których znane jest jedynie oszacowanie wsparcia w postaci przedziału, wartości minimalnego i maksymalnego wsparcia muszą być podane kolejno jedna za drugą i rozdzielone dwukropkiem. Znak okrągłego nawiasu otwierającego jest znakiem zabronionym. Istnieje także możliwość dodawania komentarzy do pliku, ponieważ linie rozpoczynające się od znaku `%`, `#` lub `@` są ignorowane przez aplikację. Przykładowy fragment pliku `*.kis`:

```
% bis results
% number of itemsets: 99775 (exact: 94986)
% min length: 1, max length: 15
36 59 23 86 34 116 (1728)
36 59 23 86 34 67 (1784:1984)
36 59 23 86 34 98 (1344)
36 59 23 86 34 76 (1784:1984)
```

Pliki definicji skryptów *.ksc

Pliki `*.ksc` to pliki tekstowe zawierające treść skryptu zgodną ze składnią opisaną w punkcie 6.2.2.

Pliki definicji projektów graficznych *.kpr

Pliki `*.kpr` są tworzone i zapisywane przez aplikację *$\mathcal{K}$-miner* z wykorzystaniem mechanizmu serializacji biblioteki MFC w formacie binarnym. Zawierają one pełną definicję projektu graficznego w postaci listy procedur i wartości ich parametrów. Do otwarcia plików `*.kpr` wymagana jest zgodność wersji procedur zapisanych w pliku z wersjami procedur dostępnych w aplikacji *$\mathcal{K}$-miner*. W przypadku braku takiej zgodności plik nie zostanie wczytany. Przykładowo, jeśli w pliku projektu znajduje się odwołanie do algorytmu `BIS`, podczas gdy w aplikacji taki algorytm nie jest dostępny (np. z powodu braku biblioteki `alg_partial_knowledge.dll`) lub jest dostępny w niezgodnej wersji (np. z powodu innej wersji biblioteki `alg_partial_knowledge.dll`), to plik nie zostanie otwarty.