

Projekt MED-P3, algorytm GRM. Raport.

Przedmiot: Metody eksploracji danych w odkrywaniu wiedzy.

Michał Aniserowicz, Jakub Turek

1 Opis zadania

Celem projektu jest zaimplementowanie algorytmu wyznaczania reguł decyzyjnych o minimalnych poprzednikach, które są częstymi generatorami. Algorytm ten jest modyfikacją algorytmu odkrywania częstych generatorów (GRM), opisanego w [1].

2 Założenia

Projekt zrealizowano w oparciu o następujące założenia:

2.1 Niefunkcjonalne:

1. Użyty język programowania; platforma: C#; .NET Framework 3.5.
2. Obsługiwane systemy operacyjne: kompatybilne z .NET Framework 3.5¹ (aplikację testowano na systemie Microsoft Windows 7 Ultimate).
3. Rodzaj aplikacji: aplikacja konsolowa uruchamiana z wiersza poleceń.

2.2 Funkcjonalne:

1. Aplikacja pobiera dane z pliku (patrz sekcja 3.1).
2. Aplikacja zwraca wynik działania w dwóch formatach: “przyjaznym dla człowieka” i “excelowym” (patrz sekcja 4).
3. Aplikacja pozwala mierzyć czas wykonania poszczególnych kroków algorytmu.

2.3 Dotyczące danych wejściowych:

1. Dane wejściowe zawierają jedynie wartości atrybutów transakcji i ewentualnie nazwy atrybutów transakcji.
2. Wartości atrybutów w pliku wejściowym są oddzielone przecinkami.
3. Każda transakcja ma przypisaną decyzję.
4. Brakujące wartości atrybutów (tzn. wartości nieznane bądź nieustalone) są oznaczone jako wartości puste lub złożone z białych znaków.

3 Dane wejściowe

Aplikacja będąca wynikiem projektu przyjmuje jednocześnie dwa rodzaje danych wejściowych:

- plik zawierający dane transakcji,
- parametry podane przez użytkownika w wierszu poleceń.

¹Lista systemów kompatybilnych z .NET Framework 3.5 dostępna jest pod adresem: <http://msdn.microsoft.com/en-us/library/vstudio/bb882520%28v=vs.90%29.aspx>, sekcja “Supported Operating Systems”.

3.1 Plik wejściowy

Plik wejściowy powinien zawierać kolejne wartości atrybutów transakcji według reguł przedstawionych w sekcji 2.3. Przykładowy format pliku wejściowego:

```
a,b,c,d,e, ,g, ,+
a,b,c,d,e,f, , ,+
a,b,c,d,e, , ,h,+
a,b, ,d,e, , , ,+
a, ,c,d,e, , ,h,-
, b,c, ,e, , , , -
```

Opcjonalnie, pierwszy wiersz pliku wejściowego może zawierać nazwy atrybutów transakcji (nagłówki), na przykład:

```
Attr A,Attr B,Attr C,Attr D,Attr E,Attr F,Attr G,Attr H,Decision
a,b,c,d,e, ,g, ,+
, b,c, ,e, , , , -
```

Jeden z atrybutów transakcji musi reprezentować przypisaną jej decyzję. Domyślnie, aplikacja uznaje ostatni atrybut transakcji za “decyzyjny” - użytkownik może jednak samodzielnie wskazać odpowiedni atrybut (patrz sekcja 3.2).

3.2 Parametry wiersza poleceń

Aplikacja przyjmuje następujące parametry:

- **--help** - Powoduje wyświetlenie informacji o dostępnych parametrach i wyjście z programu.
- **-f, --file=VALUE** - Ścieżka do pliku wejściowego. Parametr wymagany.
- **--sup, --minSup=VALUE** - Próg (bezwzględny) wsparcia - wykryte zostaną reguły decyzyjne o poprzednikach cechujących się wsparciem większym **lub równym** progowi wsparcia. Parametr wymagany.
- **-h, --headers** - Flaga oznaczająca, że plik wejściowy zawiera nagłówki atrybutów. Parametr opcjonalny.
- **--dec, --decAttr=VALUE** - Pozycja atrybutu zawierającego wartości decyzji (1 - pierwszy atrybut, 2 - drugi atrybut itd.). Parametr opcjonalny (jeśli nie zostanie podany, ostatni atrybut zostanie uznany za decyzyjny).
- **--sort=VALUE** - strategia sortowania elementów (patrz sekcja 6.2). Parametr opcjonalny. Dopuszczalne wartości:
 - **AscendingSupport** (lub 0; wartość domyślna),
 - **DescendingSupport** (lub 1),
 - **Lexicographical** (lub 2).
- **--store=VALUE** - strategia przechowywania identyfikatorów transakcji (patrz sekcja 6.3). Parametr opcjonalny. Dopuszczalne wartości:
 - **TIDSets** (lub 0; wartość domyślna),
 - **DiffSets** (lub 1).
- **--supgen=VALUE** - Strategia przechowywania generatorów decyzji, a także wykrywania i usuwania ich nadgeneratorów (patrz sekcja 6.4). Parametr opcjonalny. Dopuszczalne wartości:
 - **InvertedLists** (lub 0; wartość domyślna),
 - **BruteForce** (lub 1).

- `--track=VALUE` - Poziom monitorowania wydajności programu (patrz sekcja 6.5). Parametr opcjonalny.
 - `NoTracking` (lub 0),
 - `Task` (lub 1; wartość domyślna).
 - `Steps` (lub 2).
 - `Substeps` (lub 3; Uwaga: może powodować znaczący spadek ogólnej wydajności programu).
- `-o, --output=VALUE` - Ścieżka plików wyjściowych. Parametr opcjonalny. Poprawna wartość jest ścieżką pliku z pominięciem jego rozszerzenia (np. *wyniki/wynik*. Wartość domyślna: *[ścieżka pliku wejściowego]_rules*.

4 Dane wyjściowe

Na wyjście aplikacji składają się trzy rodzaje danych:

- komunikaty diagnostyczne wypisywane na standardowe wyjście,
- plik zawierający znalezione reguły decyzyjne w formacie czytelny dla człowieka,
- plik zawierający znalezione reguły decyzyjne w formacie przystosowanym do dalszej obróbki.

4.1 Plik czytelny dla człowieka

Plik w formacie czytelny dla człowieka zawiera listy generatorów dla każdej decyzji. Pojedynczy generator jest przedstawiony jako lista wartości atrybutów wraz z ich nazwami. Przykład:

```
=====
Decision: '+' :
-----
Attr A (a), Attr B (b)
Attr B (b), Attr D (d)
```

Plik ten jest zapisywany pod ścieżką *[plik].txt*, gdzie *[plik]* to wartość parametru `output` (patrz sekcja 3.2).

4.2 Plik przystosowany do dalszej obróbki

Plik w formacie czytelny dla człowieka zawiera listę generatorów wraz generowanymi przez nie decyzjami. Pojedynczy generator jest przedstawiony jako lista wartości oddzielonych średnikami², z uwzględnieniem atrybutów niewystępujących w generatorze. Przykład:

```
Attr A;Attr B;Attr C;Attr D;Attr E;Attr F;Attr G;Attr H;Decision
a;b;;;;;+
;b;d;;;;+
```

Plik ten jest zapisywany pod ścieżką *[plik].csv*, gdzie *[plik]* to wartość parametru `output` (patrz sekcja 3.2).

²Patrz [http://pl.wikipedia.org/wiki/CSV_\(format_pliku\)](http://pl.wikipedia.org/wiki/CSV_(format_pliku)). W omawianej aplikacji zamiast przecinków użyto średników, aby plik wyjściowy mógł być odczytany przez program Microsoft Excel.

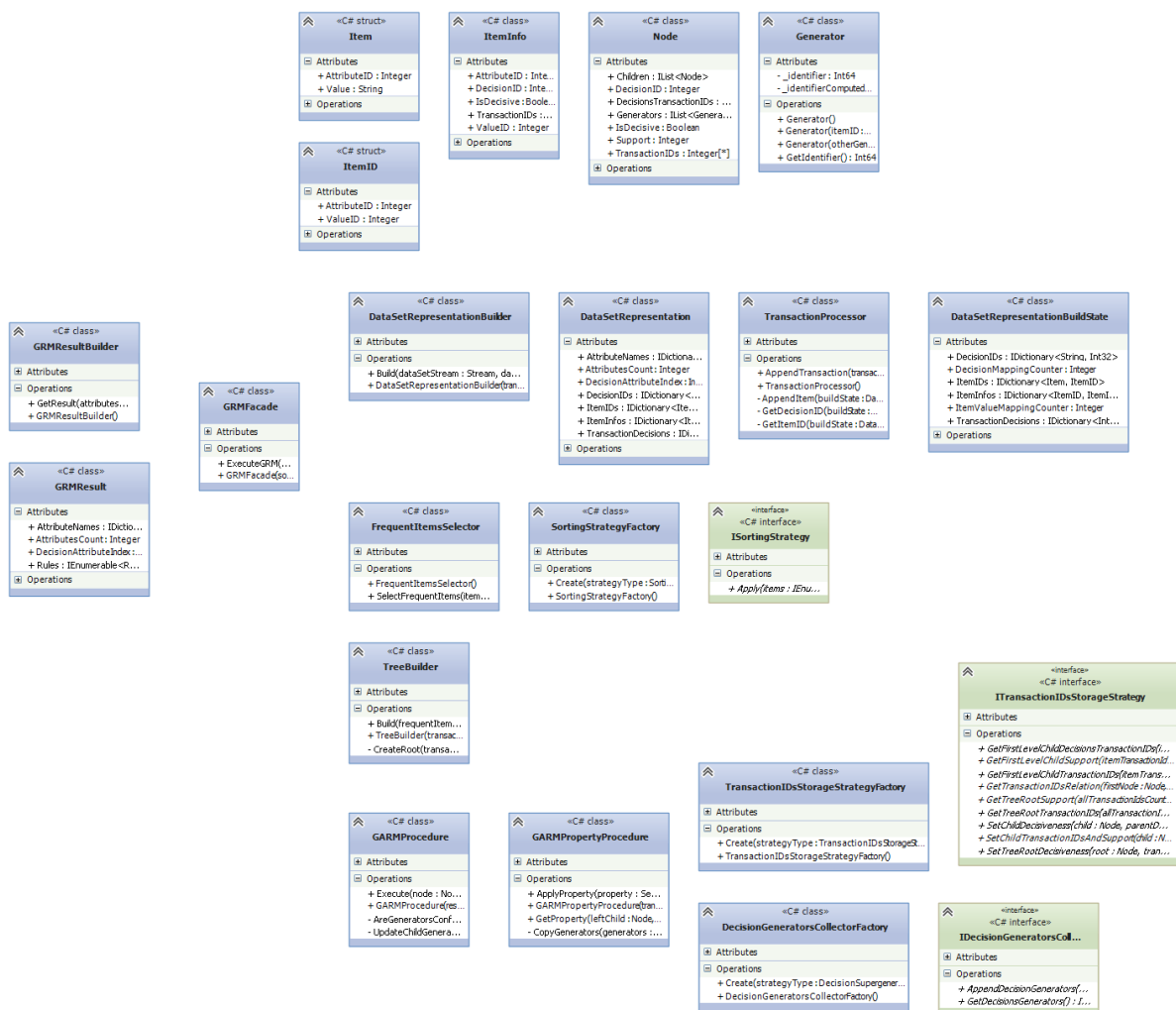
5 Architektura aplikacji

Aplikacja składa się z następujących modułów:

- **GRM.Logic** - zawiera implementację zmodyfikowanego algorytmu GRM,
- **GRM.Logic.UnitTests** - zawiera testy jednostkowe modułu **GRM.Logic**,
- **GRM.Logic.PerformanceTests** - zawiera testy wydajnościowe modułu **GRM.Logic**,
- **GRM.Presentation** - aplikacja konsolowa przetwarzająca argumenty wiersza poleceń i wywołująca algorytm GRM.

5.1 Moduł GRM.Logic

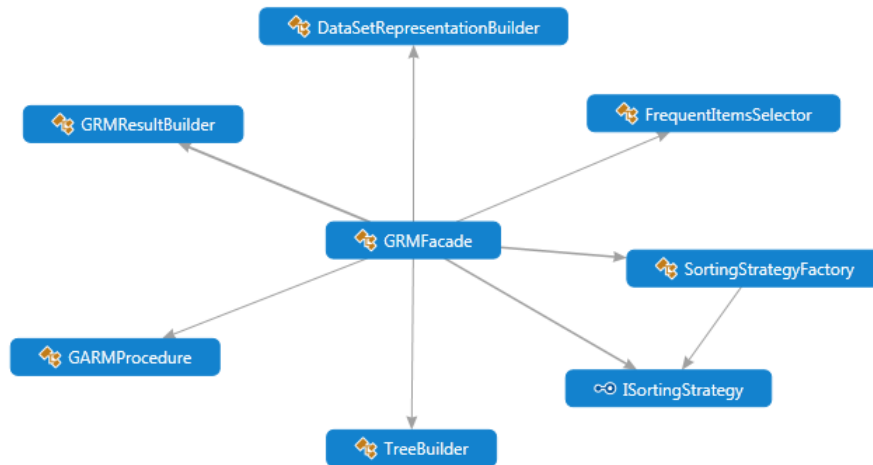
Moduł **GRM.Logic** jest głównym modułem aplikacji. Zawiera implementację zmodyfikowanego algorytmu GRM. Jego diagram klas został przedstawiony na Rysunku 1.



Rysunek 1: Uproszczony diagram klas modułu **GRM.Logic**.

Punktem wejścia modułu jest klasa **GRMFacade**. Zleca ona wykonanie kolejnych kroków algorytmu. Diagram jej zależności przedstawia Rysunek 2.

Pozzczególne kroki algorytmu zaimplementowane są w dwóch komponentach:



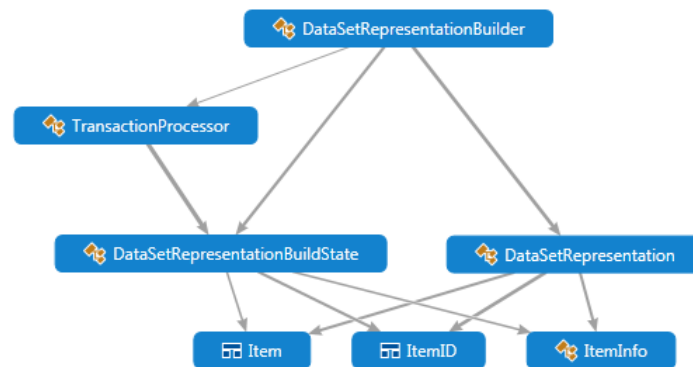
Rysunek 2: Uproszczony diagram zależności fasady modułu GRM.Logic.

- GRM.Logic.DataSetProcessing,
- GRM.Logic.GRMAlgorithm.

5.1.1 Komponent GRM.Logic.DataSetProcessing

Zadaniem komponentu GRM.Logic.DataSetProcessing jest odczytanie pliku wejściowego i zbudowanie reprezentacji zbioru danych przeznaczonej do dalszego przetwarzania.

Diagram zależności tego komponentu został przedstawiony na Rysunku 3.



Rysunek 3: Uproszczony diagram zależności komponentu GRM.Logic.DataSetProcessing.

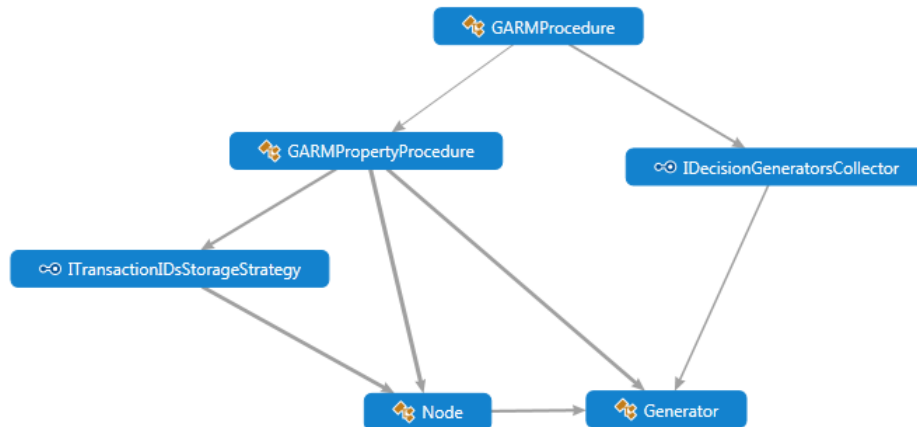
5.1.2 Komponent GRM.Logic.GRMAlgorithm

Komponent GRM.Logic.GRMAlgorithm implementuje zmodyfikowany algorytm GRM. Jego diagram zależności przedstawia Rysunek 4.

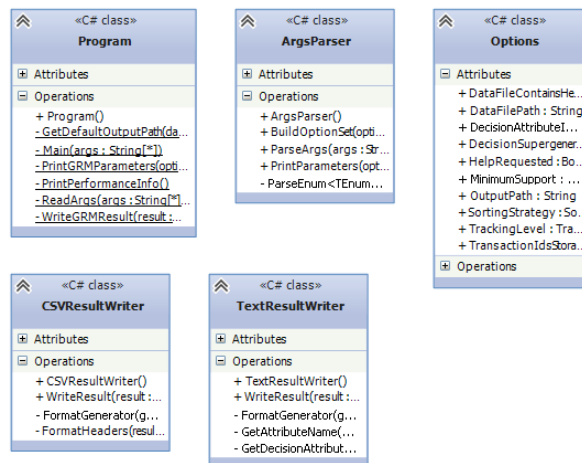
5.2 Moduł GRM.Presentation

Moduł GRM.Presentation odpowiada za komunikację z użytkownikiem aplikacji i zlecenie wykonania algorytmu GRM. Jego diagram klas przedstawia Rysunek 1.

Klasa Program zawiera główną metodę programu - Main. Jej działanie przebiega następująco:



Rysunek 4: Uproszczony diagram zależności komponentu GRM.Logic.GRMAlgorithm.



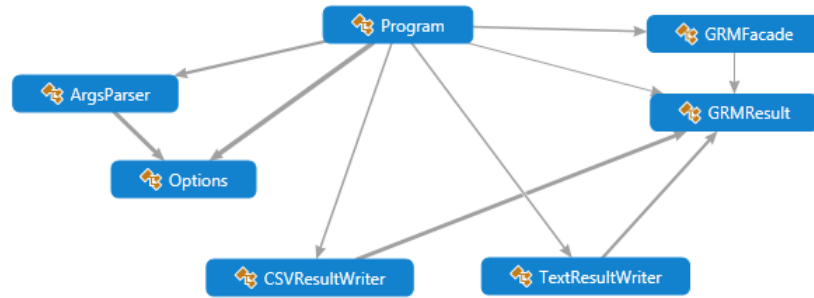
Rysunek 5: Diagram klas modułu GRM.Presentation.

1. Odczytanie parametrów wiersza poleceń (klasa **ArgsParser**). Błędne parametry skutkują wypisaniem komunikatu o błędzie i wyjściem z aplikacji.
2. Przekazanie sterowania modułowi GRM.Logic (klasie **GRMFacade**).
3. Odebranie wyniku działania algorytmu GRM (**GRMResult**) od klasy **GRMFacade**.
4. Wypisanie zebranych danych o czasach trwania kolejnych kroków algorytmu (patrz sekcja 6.5).
5. Zapisanie wyniku działania algorytmu do pliku wyjściowego w formacie czytelnym dla człowieka (klasa **TextResultWriter**).
6. Zapisanie wyniku działania algorytmu do pliku wyjściowego w formacie przystosowanym do dalszej obróbki (klasa **CSVResultWriter**).

Opisane zachowanie obrazuje diagram zależności pokazany na Rysunku 6.

6 Implementacja

Najważniejsze aspekty implementacyjne opisywanej aplikacji:



Rysunek 6: Diagram zależności modułu GRM.Presentation. Klasy GRMFacade i GRMResult należą do modułu GRM.Logic.

6.1 Kroki algorytmu

Działanie modułu GRM.Logic przebiega w następujących krokach:

[W nawiasach kwadratowych podano nazwę klasy wykonującej dany krok.]

1. [DataSetRepresentationBuilder] Odczytanie pliku wejściowego i budowa reprezentacji zbioru danych:
 - (a) określenie liczby atrybutów i odczytanie ich nagłówków (jeśli dostępne);
 - (b) dla każdego z wierszy reprezentujących transakcje:
 - i. [TransactionProcessor] nadanie identyfikatora transakcji,
 - ii. [TransactionProcessor] dla każdego elementu transakcji:
 - A. nadanie identyfikatora elementu (jeśli element o tej wartości jeszcze nie wystąpił),
 - B. aktualizacja danych elementu (zbioru transakcji, w których występuje i decyzyjności);
 - (c) zwrócenie gotowej reprezentacji danych (DataSetRepresentation).
2. [FrequentItemsSelector] Wybór elementów częstych.
3. [ISortingStrategy] Posortowanie elementów częstych.
4. [TreeBuilder] Budowa drzewa zbiorów (drzewa DZ [1]):
 - (a) utworzenie korzenia drzewa:
 - i. [ITransactionIDsStorageStrategy] wyznaczenie zbioru identyfikatorów transakcji korzenia;
 - (b) utworzenie pierwszego poziomu węzłów drzewa - dla każdego z elementów częstych:
 - i. ustawienie elementu jako generatora węzła,
 - ii. [ITransactionIDsStorageStrategy] wyznaczenie zbioru identyfikatorów transakcji węzła oraz jego wsparcia i decyzyjności;
 - (c) zwrócenie korzenia drzewa zbiorów (Node).
5. [GARMPcedure] Wywołanie procedury GARM [1].
6. [GRMResultBuilder] Budowa i zwrócenie wyniku (GRMResult).

6.2 Strategie sortowania elementów

Zaimplementowano następujące strategie sortowania elementów w drzewie zbiorów:

- AscendingSupport (klasa AscendingSupportSortingStrategy) - sortowanie rosnące według wsparć elementów,

- **DescendingSupport** (klasa **DescendingSupportSortingStrategy**) - sortowanie malejące według wsparć elementów,
- **Lexicographical** (klasa **LexicographicalSortingStrategy**) - sortowanie leksykograficzne. tzn. według numerów atrybutów elementów (dla dwóch wartości tego samego atrybutu, pierwsza będzie ta, która wcześniej wystąpiła w zbiorze danych wejściowych).

Strategie sortowania dostarcza fabryka **SortingStrategyFactory**, tworząca strategie odpowiadające zadanym wartościom wyliczenia **SortingStrategyType**.

6.3 Strategie przechowywania identyfikatorów transakcji

Zaimplementowano następujące strategie przechowywania identyfikatorów transakcji:

- **TIDSets** [2] (klasa **TIDSetsStorageStrategy**),
- **DiffSets** [1] (klasa **DiffSetsStorageStrategy**).

Strategie przechowywania dostarcza fabryka **TransactionIDsStorageStrategyFactory**, tworząca strategie odpowiadające zadanym wartościom wyliczenia **TransactionIDsStorageStrategyType**.

6.4 Strategie przechowywania generatorów decyzji

Zaimplementowano następujące strategie przechowywania generatorów decyzji, a także wykrywania i usuwania ich nadgeneratorów:

- **InvertedLists** (klasa **InvertedListsDecisionGeneratorsCollector**) - wykorzystuje koncepcję indeksu odwróconego³, gdzie kluczem indeksu jest element, a wartością - lista generatorów, w których ten element występuje,
- **BruteForce** (klasa **BruteForceDecisionGeneratorsCollector**) - przechowuje jedynie listę generatorów, nadgeneratory wykrywa porównując każdy element danego generatora z każdym elementem jego potencjalnego nadgeneratora.

Strategie dostarcza fabryka **DecisionGeneratorsCollectorFactory**, tworząca strategie odpowiadające zadanym wartościom wyliczenia **DecisionSupergeneratorsHandlingStrategyType**.

6.5 Monitorowanie wydajności programu

Aplikacja udostępnia funkcjonalność mierzenia czasu wykonania poszczególnych kroków algorytmu. Zaimplementowano następujące rodzaje pomiaru:

- **NoTracking** (klasa **EmptyProgressTracker**) - brak pomiaru wydajności,
- **Task** (klasa **TaskProgressTracker**) - pomiar czasu trwania całego algorytmu,
- **Steps** (klasa **StepProgressTracker**) - pomiar czasu trwania głównych kroków algorytmu (odczyt danych wejściowych, wykonanie procedury *GARM* itd.),
- **Substeps** (klasa **SubstepProgressTracker**) - pomiar czasu trwania mniejszych kroków algorytmu (budowanie słownika identyfikatorów decyzji, wykonanie procedury *GARM-Property* itd.). Uwaga: może powodować znaczący spadek ogólnej wydajności programu.

Obiekty monitorujące dostarcza fabryka **ProgressTrackerFactory**, tworząca obiekty odpowiadające zadanym wartościom wyliczenia **TrackingLevel**.

³Patrz http://en.wikipedia.org/wiki/Inverted_index.

6.6 Zabiegi optymalizacyjne

- Aby zminimalizować czas odczytu danych wejściowych, plik wejściowy jest odczytywany jednokrotnie, wiersz po wierszu.
- Aby zminimalizować czas trwania porównań, wszystkie wartości atrybutów otrzymują identyfikatory liczbowe.
- Aby zminimalizować czas operacji (np. przecięcia, różnicy) na zbiorach identyfikatorów transakcji, identyfikatory te są przechowywane w ustalonym (rosnącym) porządku.
- Aby możliwie wcześnie “odciąć” gałęzie drzewa, które nie prowadzą do znalezienia generatorów decyzji:
 - jeśli dwa węzły zawierają różne wartości tego samego atrybutu, to nie są “parowane” (tzn. poddawane procedurze *GARM-Property*) - przecięcie zbiorów identyfikatorów transakcji, w których występują jest puste, a zatem ich potencjalne dziecko nie może generować żadnej decyzji;
 - jeśli dany węzeł jest “decyzyjny” (tzn. generuje decyzję), to nie jest dalej rozwijany - generatory jego potencjalnych dzieci byłyby nadgeneratorami jego generatora.
- Jako że generator dowolnej decyzji D_1 nie może być nadgeneratorem żadnego generatora decyzji D_2 ($D_1 \neq D_2$), strategię przechowywania generatorów decyzji (patrz sekcja 6.4) przechowują generatory w słownikach, których kluczami są identyfikatory decyzji, a wartościami - zbiory generatorów tych decyzji. Pozwala to zminimalizować rozmiary porównywanych zbiorów generatorów.
- Granica GBd [1] nie jest tworzona i aktualizowana, jako że nie jest wykorzystywana w poszukiwaniu reguł decyzyjnych.

6.7 Praktyki programistyczne

- Test-Driven Development, TDD⁴ - najważniejsze funkcjonalności aplikacji powstały zgodnie z zachowaniem kolejności: testy jednostkowe \rightarrow implementacja \rightarrow refaktoryzacja.
- Dependency Injection⁵ - obiekty klas tworzonych i wykorzystywanych przez klasę *GRMFacade* nie tworzą swoich zależności samodzielnie, a otrzymują je z zewnątrz.
- Wzorce projektowe:
 - Fasada⁶ (klasa *GRMFacade*),
 - Budowniczy⁷ (para klas: *DataSetRepresentationBuilder* i *TransactionProcessor*),
 - Strategia⁸ (strategie opisane wyżej),
 - Fabryka⁹ (fabryki strategii opisane wyżej).

6.8 Wykorzystane biblioteki zewnętrzne

1. *NDesk.Options*¹⁰ - ułatwia przetwarzanie parametrów wiersza poleceń,
2. *xUnit.net*¹¹ - umożliwia tworzenie automatycznych testów jednostkowych,
3. *moq*¹² - wspiera tworzenie automatycznych testów jednostkowych.

⁴Patrz http://pl.wikipedia.org/wiki/Test-driven_development.

⁵Patrz http://pl.wikipedia.org/wiki/Wstrzykiwanie_zależności.

⁶Patrz [http://pl.wikipedia.org/wiki/Fasada_\(wzorzec_projektowy\)](http://pl.wikipedia.org/wiki/Fasada_(wzorzec_projektowy)).

⁷Patrz [http://pl.wikipedia.org/wiki/Budowniczy_\(wzorzec_projektowy\)](http://pl.wikipedia.org/wiki/Budowniczy_(wzorzec_projektowy)).

⁸Patrz [http://pl.wikipedia.org/wiki/Strategia_\(wzorzec_projektowy\)](http://pl.wikipedia.org/wiki/Strategia_(wzorzec_projektowy)).

⁹Patrz [http://pl.wikipedia.org/wiki/Fabryka_abstrakcyjna_\(wzorzec_projektowy\)](http://pl.wikipedia.org/wiki/Fabryka_abstrakcyjna_(wzorzec_projektowy)).

¹⁰Patrz <http://www.ndesk.org/Options>.

¹¹Patrz <http://xunit.codeplex.com/>.

¹²Patrz <https://github.com/Moq/moq4>.

7 Podręcznik użytkownika

podręcznik potencjalnego użytkownika wytworzonego oprogramowania (zamierzam korzystać z niego podczas sprawdzania Państwa rozwiązań) - wszystkie opcje programu - przykładowa komenda i wynik na konsoli

8 Analiza poprawności

wszystkie wyniki wytwarzane przez program otrzymane dla małego, przykładowego zbioru danych (w celu weryfikacji poprawności działania programu) - przykład z konsultacji

9 Analiza wydajności

wyniki jakościowe i ilościowe na (np. czas działania; liczba wzorców) uzyskane dla większych (wielkich) zbiorów danych (np. z <http://archive.ics.uci.edu/ml/> or <http://fimi.cs.helsinki.fi/data/> lub uzgodnionych już wcześniej ze mną podczas konsultacji projektowych) - uzasadnić supergenerators Inverted Lists - ze sortowanie AscendingSupport - ze storage TIDSets - wykresy, wykresy - ze dla dużej liczby atrybutów mało wydajny

10 Wnioski

wnioski z realizacji projektu - że trzeba by poprawić wykrywanie supergeneratorów - że sortowanie ma duży wpływ - że ogólnie działa spoczko (nurse)

Literatura

- [1] *Odkrywanie reprezentacji generatorowej wzorców częstych z wykorzystaniem struktur listowych*, Kryszkiewicz M., Pielasa P., Instytut Informatyki, Politechnika Warszawska.
- [2] *CHARM: An Efficient Algorithm for Closed Itemset Mining* [online], Zaki M., Hsiao C. <http://epubs.siam.org/doi/pdf/10.1137/1.9781611972726.27> [dostęp: styczeń 2013].