# Анотація

Магістерська кваліфікаційна робота присвячена дослідженню та реалізації алгоритму Apriori, та введенню можливих модифікацій для кращої роботи алгоритму.

Проект включає програмну реалізацію алгоритму Apriori та проведення дослідження для цього алгоритму.

Проведено дослідження класичного алгоритму без модифікацій, дослідження модифікованого алгоритму за першим способом, дослідження модифікованого алгоритму за другим способом, дослідження роботи алгоритму для різних значень коефіцієнтів та порівняльний аналіз вище перелічених досліджень.

Основна мета роботи полягає в порівняльному аналізі роботи класичного алгоритму Apriori та його модифікацій. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил для бази даних із дорожньо-транспортними пригодами. Виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

Об’єкт дослідження – алгоритм apriori та його модифікації.

Предмет дослідження – час і точність пошуку асоціативних правил та пошук можливих модифікацій.

Загальний обсяг роботи 79 сторінок.

# ABSTRACT

The master's qualification work is devoted to the research and development of Apriori algorithm and provide some modifications for better performance.

The project includes software implementation of the Apriori algorithm and conducting experiments of the developed tool.

The Research was conducted for basic algorithm without modifications, the study of the modified algorithm by the first method, the research of the modified algorithm by the second method, research of the algorithm's performance for different values ​​of the coefficients and a comparative analysis of the above-mentioned studies.

The main purpose of master's qualification work is to compare the work of the basic Apriori algorithm and its modifications on large volumes of data. It is necessary to process the data with the specified algorithm and estimate the speed of the algorithm, the load on the resources, the number of found associative rules for the database with road accidentsts. To work out on the basis of computational experiments a proposal for modification of the algorithm.

The object of the research is the apriori algorithm and its modifications.

Subject of research is the time and accuracy of the search for associative rules and the search for possible modifications.

The total workload is 79 pages.

# ВСТУП

Зараз ІТ є однією з динамічніших сфер, де активно розробляються нові технології, створюються сотні програмних продуктів та накопичуються приблизно декілька сотень гігабайт інформації за годину. Ця інформація надходить з різних джерел, та містить багато даних.

Big Data відноситься до величезних масивів даних, які важко аналізуються традиційними методами. Найбільш відомим та широко застосовуваним зараз є алгоритм Apriori.

Алгоритм Apriori призначений для знаходження всіх частих наборів елементів множини транзакцій, що аналізується. Він працює ітераційно, використовуючи пошук в ширину і та принцип зверху вниз. Алгоритм Apriori має просту структуру, завдяки якій в неї можна легко вносити модифікації для оптимізації алгоритму під конкретні набори даних для знаходження асоціативних зв’язків.

Мета роботи полягає в порівняльному аналізі роботи класичного алгоритму Apriori та його модифікацій. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил на певних наборах даних. Виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

У першому розділі даної роботи проведено аналіз предметної області. Після аналізу існуючих систем, що займаються пошуком закономірностей у даних, огляду існуючих підходів та алгоритмів для опрацювання великих даних було виявлено потребу у дослідженні роботи алгоритму, та на основі обчислювальних експериментів надати пропозицію для модифікації алгоритму.

У другому розділі цієї роботи подано опис алгоритму та можливі модифікації. Для першої модифікації потрібно розбити множину всіх можливих транспортних пригод на дві підмножини, одна з них містить корисну інформацію для аналізу, а інша ні, отже та що не містить корисної інформації потрібно відкидати на кожному кроці виконання алгоритму. Для другої модифікації потрібно додати характеристику асоціативних правил Ліфт, та обчислювати його на кожному кроці алгоритму.

Третій розділ роботи містить специфікацію вимог до програмної реалізації алгоритму представленого у розділі 2, інформацію про використані технології, графічний інтерфейс програмного засобу та певні особливості програмної реалізації засобу для проведення аналізу даних алгоритмом Apriori.

У четвертому розділі показано дослідження розробленого засобу. Дослідження складається з таких частин: дослідження класичного алгоритму без модифікацій, дослідження модифікованого алгоритму за першим способом, дослідження модифікованого алгоритму за другим способом, дослідження роботи алгоритму при різними значеннями коефіцієнтів та порівняльний аналіз вище перелічених досліджень.

У останньому розділі даної роботи показано економічну характеристику програмного продукту, розробленого у розділі 3.

# РОЗДІЛ 1. АНАЛІЗ ЗАСТОСУВАННЯ АЛГОРИТМУ APRIORI

* 1. **Великі дані, визначення та їх особливості**

Великі дані - це набори даних, настільки об'ємні та складні, що традиційне прикладне програмне забезпечення для обробки даних є недостатнім для їх вирішення. Великі задачі даних охоплюють захоплення даних, зберігання даних, аналіз даних, пошук, обмін, передачу, візуалізацію, запити, оновлення, конфіденційність інформації та джерела даних. Існує ряд понять, пов'язаних з великими даними: спочатку було 3 поняття обсягу, різноманітності, швидкості[1]. Іншими поняттями, які потім пояснюються великими даними, є вірогідність (тобто, скільки шуму в даних) [2] та значення. Останнім часом термін "великі дані" має на увазі використання прогнозної аналітики, аналізу поведінки користувачів або деяких інших розширених методів аналізу даних, які витягують значення з даних, а рідко - до певного розміру набору даних. "Існує мало сумнівів у тому, що кількість даних, які зараз доступні, дійсно велика, але це не є найважливішою характеристикою цієї нової екосистеми даних". Аналіз наборів даних може знайти нові зв'язки з "точковими діловими тенденціями, запобігання захворюванням, боротьба зі злочинністю та ін. "[3]

Вчені, керівники підприємств, практики медицини, реклами та уряди так само регулярно зустрічаються з труднощами з великими наборами даних у таких областях, як пошук в Інтернеті, технології, міська інформатика та бізнес-інформатика. Вчені стикаються з обмеженнями в роботі електронної науки, включаючи метеорологію, складні фізичні симуляції, біологію та екологічні дослідження [4].

Цей термін використовується з 1990-х років, при цьому деякі приносять кредит Джону Маші за виготовлення або, принаймні, популярністю. [5] Великі дані зазвичай включають в себе набори даних, розміри яких не залежать від можливостей широко використовуваних програмних інструментів для захоплення, копіювання, керування та обробки даних протягом допустимого минулого часу [6]. Велика філософія даних охоплює неструктуровані, напівструктуровані та структуровані дані, однак основна увага приділяється неструктурованим даним.

Визначення 2016 р. Свідчить, що "великі дані являють собою інформаційні активи, які характеризуються такими великими обсягами, швидкістю та різноманітністю, що вимагають спеціальних технологій та аналітичних методів перетворення їх у вартість" [7].

Великі дані можна описати за такими характеристиками: [8]

**Обсяг** (англ. Volume) **–** кількість створених та збережених даних. Розмір даних визначає цінність та потенційне розуміння, і чи можна вважати великі дані чи ні.

**Різноманітність** (англ. Variety) **–** тип та характер даних. Це допомагає людям, які аналізують його, ефективно використовувати отримане розуміння. Великі дані витягуються з тексту, зображень, аудіо, відео; плюс він завершує відсутність частин через злиття даних.

**Швидкість** (англ. Velocity) **–** у цьому контексті швидкість, з якою дані створюються та обробляються, щоб задовольнити вимоги та виклики, які лежать на шляху зростання та розвитку. Великі дані часто доступні в режимі реального часу.

Дані треба обробляти за допомогою розширених інструментів (аналітики та алгоритмів) для виявлення значущої інформації. Наприклад, щоб керувати фабрикою, слід розглядати як видимі, так і невидимі проблеми з різними компонентами. Алгоритми формування інформації повинні виявляти та розв'язувати невидимі проблеми, такі як деградація машини, знос компонентів тощо на заводі. [9]

* 1. **Поняття асоціативних правил**

Афінітивний аналіз — один з найбільш відомих методів Data Mining. Назва його походить від слова affinity, що перекладається як «близькість», «подібність». Мета даного методу — взаємне дослідження зв'язку між подіями, які відбуваються спільно. Різновидом афінітивного аналізу є аналіз ринкового кошика (market basket analysis), ціль якого - виявити асоціації між різними подіями, тобто знайти правила для кількісного опису взаємного зв'язку між двома або більше подіями. Такі правила називаються асоціативними правилами.[10]

Основним поняттям в теорії асоціативних правил є транзакція - сукупність подій, що відбуваються разом. Типова транзакція - покупки клієнта в супермаркеті. У переважній більшості випадків клієнт купує більше одного продукту, а набір товарів, є ринковою корзиною. Це ставить питання: чи є покупка одного продукту в кошику як наслідок чи причина для покупки іншого товару, тобто пов'язані з цим події? Це з'єднання встановлює асоціативні правила. Наприклад, можна знайти асоціативне правило, в якому зазначається, що замовник, який купив молоко, з імовірністю на 75% купує хліб.

Наступним важливим поняттям є предметний набір. Це непустий набір елементів, які з'являються в одній транзакції.

Аналіз ринку корзин - це аналіз наборів даних для певної комбінації суміжних продуктів. Іншими словами, здійснюється пошук товарів, наявність яких у транзакції впливає на вірогідність наявності інших товарів або комбінацій товарів.

Сучасні касові апарати в супермаркетах дозволяють збирати інформацію про покупки, які можна зберігати в базі даних. Потім накопичені дані можуть бути використані для побудови пошукових систем асоціативних правил.

У табл. 1.1 наведено простий приклад ринкової корзини. Кожен рядок містить комбінацію продуктів, придбаних для однієї покупки. Хоча на практиці він має справу з мільйонами транзакцій із десятками та сотнями різних продуктів, приклад обмежується 10 операціями, що містять 13 видів продукції: це достатньо для ілюстрації методології визначення асоціативних правил.

Таблиця 1.1.

Приклад набору транзакцій

|  |  |
| --- | --- |
| **№** | **Транзакція** |
| 1 | Сливи, салат, помідори |
| 2 | Селера, цукерки |
| 3 | Цукерки |
| 4 | Яблука, морква, помідори, картопля, цукерки |
| 5 | Яблука, апельсини, салат, цукерки, помідори |
| 6 | Персики, апельсини, селера, помідори |
| 7 | Квасоля, салат, помідори |
| 8 | Апельсини, салат, морква, помідори, цукерки |
| 9 | Яблука, банани, сливи, морква, помідори, цибуля, цукерки |
| 10 | Яблука, картопля |

Візуальний аналіз цього прикладу показує, що всі чотири транзакції, що включають салат, також включають помідори, а в чотири з семи помідорів містять також салат. Салат і помідори в більшості випадків купуються разом. Асоціативні правила дозволяють виявити та кількісно визначити такі збіги.

Асоціативне правило складається з двох наборів об'єктів, що називаються попередніми та наступними, і записані у формі. Таким чином, асоціативне правило формулюється у формі: "Якщо стан, то наслідок".

Умова може бути обмежена лише одним предметом. Правила зазвичай відображаються за допомогою стрілок, спрямованих від умов до ефекту, наприклад, помідори → салат.

Візуальний аналіз цього прикладу показує, що всі чотири транзакції, що включають цибулю, також включають картоплю, і чотири з семи картоплі, що містять картоплю, також містять цибулю. У більшості випадків цибулю та картоплю купуються разом. Асоціативні правила дозволяють виявити та кількісно визначити такі збіги. Також можна комбінувати правила (Цибуля, Картопля) => {Бургер}

Умова може бути обмежена лише одним предметом. Правила зазвичай відображаються за допомогою стрілок, спрямованих від умови до ефекту, наприклад, цибуля → картопля.

Основними характеристиками, які описують асоціативне правило, є підтримка та надійність. [11-12].

Позначимо базу даних транзакцій через , а число транзакцій у цій базі , то кожна транзакція  являє собою певний набір предметів. Позначимо підтримку правила через , а вірогідність – через .

Підтримка асоціативного правила — це число транзакцій, які містять як умову, так і наслідок. Наприклад, для асоціації  можна записати[30]

 (1.1)

Імовірність асоціативного правила  є мірою точності правил і визначається як відношення кількості транзакцій, що містять стан і наслідок до кількості транзакцій, що містять лише умову:

 (1.2)

Якщо підтримка та ймовірність досить високі, ви, швидше за все, можете стверджувати, що будь-яка майбутня транзакція, включаючи умови, також матиме наслідок.

Наприклад розглянемо правило салат → помідори, яке випливає з попередніх спостережень. Для нього

;

.

Це правило, яке відбувається в 40% транзакцій, полягає в тому, що вихідні дані є абсолютно правильними - у всіх випадках, коли клієнт купує салат, він також купує помідори. Це легко пояснити логічно - обидва продукти використовуються для приготування овочевих страв і часто купуються разом.

Тепер давайте розглянемо асоціацію цукерки → помідори, яка містить продукти, які погано сумісні з гастрономією.

Підтримка цієї асоціації S = 4/10 = 0,4 - така ж, як у попередньому правилі, а ймовірність C = 4/6 = 0,67. Отже, відносно низька ймовірність цієї асоціації викликає сумніви в тому, що це правило.

Число 0.67, здається, не настільки мале. Чому ми говоримо про "незначну ймовірність" цієї асоціації? Справа в тому, що помідори знаходяться в 7 перевірках з 10 (Р(В)=0.7). Прийнято вважати, що всі правила з ймовірністю, меншими за просту ймовірність наслідків, не повинні розглядатися, оскільки вони фактично є випадковими. Щоб прийняти асоціативне правило, його ймовірність повинна бути не меншою, ніж ймовірність наслідку.

Останнє зауваження в досить великому діапазоні продуктів призводить до необхідності подвоїти кількість розрахунків. На практиці аналітики можуть віддати перевагу правилам, які мають високий рівень підтримки (вище певного рівня, наприклад 0,3) або високої ймовірності (щонайменше 0,8-0,85). Висування вимог щодо одночасної підтримки та правдоподібності може значно пом'якшити критерії (підтримка до 0.1-0.15, ймовірність - до 0.67-0.75). Правила, для яких підтримка та значення надійності перевищують певні пороги, встановлені користувачем, називаються сильними правилами. Всі вищезгадані числові значення є емпіричними. Наприклад, в задачі виявлення шахрайських операцій вартість підтримки може зменшитися до 1%, оскільки кількість операцій, пов'язаних із шахрайством, є порівняно невеликою.

Окрім об'єктивних оцінок (підтримки та ймовірності) кожного порожнього правила, для різних джерел рекомендується використовувати деякі суб'єктивні оцінки. Всі вони, так чи інакше, базуються на об'єктивних.

Ліфт (від interest lift – підвищення інтересу) підраховується наступним чином

 (1.3)

Ліфт - це співвідношення частоти виникнення умови у транзакціях, які містять як умову, так і наслідок до частоти виникнення ефекту в цілому. Чим більше значення ліфта, тим частіше наслідок визначається умовою у порівнянні з випадками, коли відсутня умова. Якщо ліфт дорівнює 1, зв'язок відсутній, значення, близькі до нуля, вказують на сильні взаємні відносини [12].

Для нашого прикладу в таблиці 1.1. з однаковою вірогідністю приймають два правила: 4/10 = 0,4. ,4/7 = 0,57.

*салат) =*

*C(помідори*



 6/10 = 0,6. 4/7 = 0,57.

*цукерки) =*

*C(помідори*



Здавалося б, правила однаково достовірні. Після розрахунку ліфта все стає на місця:

0,57/0,4 = 1,425;-

*салат) =*

*L(помідори*



0,57/0,6 = 0,95.

*цукерки) =*

*L(помідори*



Не вважайте ліфт універсальною мірою адекватності. Справа в тому, що правило з меншою підтримкою та збільшенням ліфта може бути менш важливим, ніж альтернативне правило з більшою підтримкою та меншим підйомом. Це пов'язано з тим, що останній застосовується для більшої кількості покупців.

Левередж (leverage – важіль, плече) – це різниця між частотою спостереження, з якої складаються умови та наслідки, а також результат періодичності появи умов і ефектів окремо

 (1.4)

Левередж дозволяє впоратися із ситуаціями, коли й підтримка, й ліфт у правил ідентичні, але їх важливість явно відрізняється. Наприклад, у нашому овочевому магазині в правила салат → помідори

;  0,7;  1/0,7 = 1,43.

Морква, як показує таблиця 1.1, також продається тільки з помідорами, і також зустрічається чотири рази, тому й у правила морква → помідори

;  0,7;  1/0,7 = 1,43.

А от левередж у цих правил відрізняється на 30%:

 0,3 – 0,3 \* 0,7 = 0,09.

 0,4 – 0,4 \* 0,7 = 0,12.

Таким чином, важливість другої асоціації більша, ніж першої.

Альтернативою для левередж є поліпшення.

Поліпшення (improvement) – це відношення частоти спостережуваних виконань правила до добутку частот появи умови й наслідку окремо.

 (1.5)

фактично, поліпшення показує, у скільки разів розглянуте правило забезпечує правильний прогноз краще, ніж випадкове вгадування. Всі правила  не є важливими.

Такі міри, як ліфт, левередж і поліпшення, можуть використовуватися для обмеження набору розглянутих асоціацій шляхом встановлення граничних значень важливості, нижче яких асоціації відкидаються.

* 1. **Аналіз існуючих досліджень**

У [13] проаналізовано алгоритм Apriori як метод пошуку асоціативних правил у структурованих та неструктурованих даних з погляду кількості знайдених правил, швидкодії та потреб в обчислювальних ресурсах. Неструктуровані дані тісно пов’язані з терміном Big Data. Актуальним завданням інженерії даних є виявлення ефективних засобів опрацювання неструктурованої інформації. Для проведення обчислювальних експериментів розроблено програмну систему, що опрацьовує дані алгоритмом Apriori, предметною областю якої вибрано торгівлю. Така система може бути прототипом реальної рекомендаційної системи.

Отже, було досліджено що,

1. Втрати у швидкодії для неструктурованої інформації порівняно зі структурованою інформацією незначні зі зростанням обсягів інформації, і зокрема, для випадку обсягу даних 2,5 Гб зростання часу роботи становить близько 12 %. А на порівняно невеликих обсягах даних для структурованих даних алгоритм працює утричі швидше порівняно із неструктурованими.

2. Кількість знайдених правил для неструктурованої інформації на близько 25 % менша, ніж для структурованої інформації.

3. Для коефіцієнтів підтримки та достовірності, вищих від середнього, для неструктурованих даних не знайдено жодного правила, а для структурованих результат – від 2 до 5 правил.

У [14] виконувався аналіз надзвичайних ситуацій у залізничному транспорті для пошуку асоціативних правил із використанням алгоритму Apriori. У цьому дослідженні запропоновано модифікацію алгоритму: розбиття множини всіх можливих характеристик на дві підмножини:

1. A – характеристики, сукупність яких могла стати причиною виникнення надзвичайних ситуацій.
2. B – характеристики, скупність яких описує надзвичайну ситуацію що виникла.

В результаті застосування цієї модифікації збільшується швидкість роботи

алгоритму.

З отриманих результатів дослідження видно що алгоритм є доволі дослідженим та схильним до модифікацій проте, його можна досліджувати для навіструктурованих даних або ввести нові модифікації для більш точної обробки інформації.

* 1. **Мета, об’єкт та предмет дослідження**

Мета роботи полягає в порівняльному аналізі роботи алгоритму Apriori. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил для бази даних із дорожньо-транспортними пригодами. Виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

Об’єкт дослідження – алгоритм apriori та його модифікації.

Предмет дослідження – час і точність пошуку асоціативних правил та пошук можливих модифікацій.

* 1. **Висновок до розділу 1**

Накопичення інформації є актуальною темою сьогодення. Для аналізу цієї інформації використовують різноманітні технології та алгоритму. Після огляду декількох із них було вирішено провести дослідження для алгоритму Apriori.

Отже, у магістерській кваліфікаційній роботі необхідно розглянути алгоритм Apriori з точки зору пошуку асоціативних правил у великій кількості даних та проаналізувати затрачені обчислювальні ресурси. Мета роботи полягає в порівняльному аналізі роботи класичного алгоритму Apriori та його модифікацій. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил на певних наборах даних.

Після аналізу існуючих досліджень, що займаються пошуком закономірностей у даних, огляду існуючих підходів та алгоритмів для опрацювання великих даних було виявлено потребу у дослідженні роботи алгоритму, та на основі обчислювальних експериментів надати пропозицію для модифікації алгоритму.

# РОЗДІЛ 2. Опис алгоритму apriori

1. 1. **Алгоритм apriori**

На даний момент існує декілька основних алгоритмів пошуку асоціативних правил в загальному вигляді, що не зав’язані на конкретизовану предметну область. Розглянемо найбільш поширені з них.

Алгоритм Apriori [15] призначений для знаходження всіх частих наборів елементів множини транзакцій, що аналізується. Він працює ітеративно, використовуючи пошук в ширину і принцип зверху вниз. Алгоритм Apriori має просту структуру, завдяки якій в неї можна легко вносити модифікації для оптимізації алгоритму під конкретні набори даних для вирішення задачі пошуку асоціативних правил.

Алгоритм Eclat [16] побудований на основі пошуку в глибину використовуючи перетин наборів елементів для знаходження частих наборів елементів. Це дуже простий алгоритм пошуку частого набору елементів. Цей алгоритм використовує вертикальну базу даних. Він не може використовувати горизонтальну базу даних. Якщо є горизонтальна база даних, то нам потрібно перетворити її у вертикальну базу даних. Не потрібно сканувати базу даних знову і знову. Алгоритм Eclat сканує базу даних лише один раз. Підтримка враховується в цьому алгоритмі. В цьому алгоритмі не враховується достовірність.

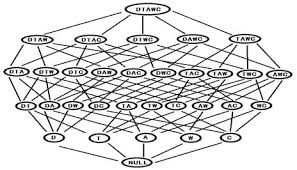


Рис. 2.1. Приклад роботи алгоритму Eclat

В загальному випадку алгоритм працює повільніше ніж алгоритм Apriori, однак в ситуації коли кількість різних елементів транзакцій є невеликою, він працює швидше.

Алгоритм FP-growth [17] (frequent pattern growth, зростання частих патернів) використовує розширене префіксне дерево для збереження бази даних у стислому вигляді. Цей алгоритм застосовує метод «розділяй і володарюй» для декомпозиції і видобутку знань із бази даних. В процесі роботи алгоритм використовує метод зростаючих патернів для уникнення ресурсоємкого процесу генерації частих кандидатів і їх тестування, що використовується в алгоритмі Apriori. Однак через це ускладнюється процес оптимізації алгоритму для використання з специфічними наборами даних, де покращення швидкості роботи можна досягти іншими методами.

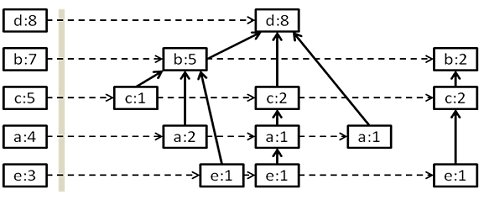


Рис. 2.2. Приклад роботи алгоритму FP-growth

Отже, враховуючи переваги та недоліки розглянутих алгоритмів пошуку асоціативних правил, було обрано алгоритм Apriori так як він є простим у розумінні та можна легко вносити модифікації для оптимізації алгоритму під конкретні набори даних.[23-29]

Таблиця 2.1.

Приклад набору транзакцій

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Transaction ID** | **Цибуля** | **Картопля** | **Бургер** | **Молоко** | **Пиво** |
|  | * 1 | 1 | 1 | 0 | 0 |
| t_2 | 0 | 1 | 1 | 1 | 0 |
| t_3 | 0 | 0 | 0 | 1 | 1 |
| t_4 | 1 | 1 | 0 | 1 | 0 |
| t_5 | 1 | 1 | 1 | 0 | 1 |
| t_6 | 1 | 1 | 1 | 1 | 1 |

Ключовим поняттям в алгоритмі Апріорі є антимонотонність міри підтримки. Це передбачає що:

* Всі підмножини частої Набір повинні бути частими
* Подібним чином, для будь-якого незвичного набору, всі його суперсеті також повинні бути нечастими

Давайте розглянемо інтуїтивно зрозуміле пояснення алгоритму за допомогою наведеного вище прикладу(табл. 2). Перед початком процесу, встановіть порогової підтримки до 50%, тобто лише ті елементи значні, для яких підтримка більше 50%.

**Крок 1:** Створити таблицю частот для всіх елементів, що виникають у всіх транзакціях. Для нашого випадку:

Таблиця 2.2.

Таблиця частот

|  |  |
| --- | --- |
| **Одиниця** | **Частота** |
| Цибуля(O) | 4 |
| Картопля(P) | 5 |
| Бургер(B) | 4 |
| Молоко(M) | 4 |
| Пиво(Be) | 2 |

**Крок 2:** Ми знаємо, що важливі лише ті елементи, для яких підтримка більше або дорівнює пороговій підтримці. Тут границя підтримки становить 50%, отже, лише ті елементи значущі, що трапляються у більш ніж трьох транзакціях, а такими є - Цибуля (O), Картопля (P), Бургер (B) і Молоко (M). Тому ми отримуємо наступну таблицю:

Таблиця 2.3.

Таблиця частот

|  |  |
| --- | --- |
| **Одиниця** | **Частота** |
| Цибуля(O) | 4 |
| Картопля(P) | 5 |
| Бургер(B) | 4 |
| Молоко(M) | 4 |

**Крок 3:** Наступним кроком є зробити всі можливі пари значущих елементів, маючи на увазі, що порядок не має значення, тобто АB такий самий, як BА. Щоб зробити це, візьміть перший предмет і з'єднайте його з усіма іншими, такими як OP, OB, OM. Аналогічним чином розглянемо другий елемент і об'єднає його з попередніми пунктами, тобто PB, PM. Ми розглядаємо лише попередні пункти, оскільки PO (як OP) вже існує. Отже, всі пари в нашому прикладі - OP, OB, OM, PB, PM, BM.

**Крок 4:** Тепер ми розглянемо випадки кожної пари у всіх транзакціях.

Таблиця 2.4.

Таблиця частот

|  |  |
| --- | --- |
| **Набір** | **Частота** |
| OP | 4 |
| OB | 3 |
| OM | 2 |
| PB | 4 |
| PM | 3 |
| BM | 2 |

**Крок 5.** Знову тільки ті набори наборів є значними, які перетинають порогову підтримку, а такі - OP, OB, PB та PM.

**Крок 6.** Тепер скажімо, ми хотіли б шукати набір з трьох елементів, які купуються разом. Ми будемо використовувати пункти, знайдені на кроці 5, і створимо набір з 3 елементів.

Щоб створити набір з 3 елементів, потрібно інше правило, яке називається самоз'єднання. Він говорить, що з пар OP, OB, PB та PM ми шукаємо дві пари з однаковою першою буквою, і об’єднуємо їх:

* OP та OB, це дає OPB
* PB і PM, це дає PBM

Далі ми знайдемо частоту для цих двох наборів.

Таблиця 2.5.

Таблиця частот

|  |  |
| --- | --- |
| **Набір** | **Частота** |
| OPB | 4 |
| PBM | 3 |

Знову застосувавши правило порогу, ми виявили, що OPB є єдиним значним набором елементів.

Отже, набір з 3 предметів, який був придбаний найчастіше, - OPB.

Приклад, який ми розглянули, був досить простим, і видобуток частих предметів у товаристві зупинено на 3 пунктах, але на практиці існує кілька десятків предметів, і цей процес може тривати багато предметів. Припустимо, що ми отримали значні набори з трьома елементами як OPQ, OPR, OQR, OQS та PQR, і тепер ми хочемо створити набір з 4 елементів. Для цього ми розглянемо набори, які мають перші два алфавіти, тобто загалом

OPQ та OPR дають OPQR

OQR та OQS дають OQRS

Взагалі, ми повинні шукати набори, які тільки відрізняються в їхній останній букві / предметі.

Тепер, коли ми розглянули приклад функціональності алгоритму Апріорі, давайте сформулювати загальний процес.

Весь алгоритм можна розділити на два етапи:

**Крок 1:** застосуйте мінімальну підтримку, щоб знайти всі часто використовувані набори з елементами k у базі даних.

**Крок 2.** Використовуйте правило самопідключення, щоб знайти часті множини з елементами k + 1 за допомогою часто використовуваних k-елементів. Повторіть цей процес з k = 1 до моменту, коли нам не вдається застосувати правило самопідключення.

Такий підхід до розповсюдження частих елементів, що належить до одного, називається підходом "знизу вверх".

Cкладність цього алгоритму залежить від мінімального значення підтримки. Припустимо, що кількість вхідних транзакцій N, а кількість унікальних елементів R. Складність для генерації набору розміру i – О(Ri), де і – час для обчислення підтримки для кожного набору може бути O(N), у разі використання HashMap. Блок-схема алгоритму подано у Додатку Б.

* 1. **Видобування асоціативних правил**

До цих пір ми розглянули алгоритм Апріорі по відношенню до генеруванні частих наборів. Існує ще одне завдання, за допомогою якого ми можемо використовувати цей алгоритм, тобто ефективно знаходити асоціативні правила. [26-27]

Щоб знайти асоціативні правила, нам потрібно знайти всі правила, що мають підтримку, більшу за порогову підтримку та довіру, більшу за порогову довіру.

Але як ми знаходимо це? Одним із можливих шляхів – це перебір всіх можливих асоціативних правил та обчислення підтримки та впевненості для кожного правила. Потім просканувати правила, які не підтримують порогову підтримку та довіру. Але це обчислювально дуже складний і неефективний процес, оскільки кількість всіх можливих правил асоціації зростає експоненціально з кількістю предметів.

Враховуючи, що в наборі I є n пунктів, загальна кількість можливих правил асоціації становить 3 ^ n - 2 ^ (n +1) + 1.

Ми також можемо використовувати інший спосіб, який називається двоетапним підходом, для пошуку ефективних правил асоціації.

Двоступеневий підхід:

Крок 1: генерація частотних наборів елементів: знайдіть всі набори предметів, для яких підтримка перевищує порогову підтримку відповідно до процесу, який ми вже бачили раніше в цій статті.

Крок 2: генерація правил: створюйте правила з кожної частинки, використовуючи бінарний розділ часто використовуваних предметів і шукайте тих, що мають високу впевненість. Ці правила називаються правилами кандидатів.

Давайте розглянемо наш попередній приклад, щоб отримати ефективне правило об'єднання. Ми виявили, що OPB є частим комплектом. Тому для цієї проблеми, крок 1 вже зроблено. Отже, давайте розглянемо крок 2. Усі можливі правила, що використовують OPB, є:

OP\longrightarrowB, OB\longrightarrowP, PB\longrightarrowO, B\longrightarrow OP, P\longrightarrowOB, O\longrightarrowPB

Якщо X являє собою частий набір елементів з k-елементами, то існують правила асоціації кандидатів 2 ^ k-2.

Плюси алгоритму:

Легкий у реалізації та розумінні.

Може бути використаний для величезних даних.

Мінуси алгоритму:

Іноді, потрібно знайти величезну кількість правил кандидатів, що є обчислювально затратно.

Підрахунок підтримки також затратний по ресурсах, адже потрібно пройтись по всій базі.

* 1. **Опис відомих модифікацій**

Проаналізувавши суміжні дослідження було виявлено що вже існує ряд модифікацій для алгоритму apriori, які покращують його роботу.

PredictiveApriori - це алгоритм, який використовує більшу підтримку та обмінюється з більшою впевненістю, а також обчислює очікувану точність в байєсівських рамках. Результат цього алгоритму максимізує очікувану точність майбутніх даних правил асоціації. Цей алгоритм генерує правила асоціації як очікуване число правил користувача. Схеффер [13] визначає цей алгоритм за допомогою:

Нехай D - це база даних, індивідуальні записи r яких генеруються статичним процесом P, нехай X → Y є правилом асоціації. Точність прогнозування c (X → Y) = Pr (r, що задовольняє Y | r, задовольняє X) є умовною ймовірністю Y ⊆ r, якщо X ⊆ r, коли розподіл r регулюється P.

Правило додається, якщо: очікувана прогностична точність цього правила є одним з 'N' найкращих і не поглинена правилом, принаймні тієї ж очікуваної точності прогнозу. Якщо включені асоціативні правила видобуваються замість (загальних) асоціативних правил.

Якщо встановлено значення -1, останній атрибут буде взятий в якості атрибута класу та існує кількість правил для знаходження. Можливі пропущені значення, бінарні атрибути, порожні номінальні атрибути та номінальний клас.

Алгоритм AprioriTid є різновидом алгоритму Apriori. Цікава особливість полягає в тому, що база даних D не використовується для підрахунку підтримка після першого проходу. Кожен член комплекту | Ck | має форму <TID; {Xk}>, де кожний Xk потенційно великі k-елементи присутні в транзакції за ідентифікатором TID. Для k = 1, C1 відповідає базі даних D, хоча концептуально кожен елемент замінюється itemset {i}. При k> 1, | Ck | згенерований алгоритмом. Член Ck, що відповідає транзакції t, є <t.TID, {c∈ Ck | c міститься в t}>. Якщо транзакція не містить жодного кандидата k-елементів, а потім | Ck | не матиме запису для цього транзакція Таким чином, кількість записів у | Ck | може бути менше числа операції в базі даних, особливо для великих значень k. Також, для великих значення k, кожен запис може бути меншим, ніж відповідна транзакція, тому що, дуже мало кандидатів можуть бути включені в транзакцію. Однак для невеликих цінностей для k кожен запис може бути більшим, ніж відповідна транзакція, тому що запис у Ck включає всі k-множини кандидатів, що містяться в транзакції.

Кожен член безлічі Ck є парою виду , де кожен Fk є потенційно частим k - елементним набором, представленим в транзакції з ідентифікатором TID. Безліч C1 = D відповідає безлічі транзакцій, хоча кожен об'єкт в транзакції відповідає однооб'єктному набору в безлічі C1, що містить цей об'єкт. Для k> 1 безліч Ck генерується відповідно до алгоритму, описаним нижче.

Член безлічі Ck, відповідний транзакції Т, є парою такого вигляду:

 (2.1)

Підмножина наборів в Ck з однаковими TID (містяться в одній і тій же транзакції) називається записом. Якщо транзакція не містить ні одного k - елементного кандидата, то Ck не матиме записи для цієї транзакції Тобто кількість записів в Ck може бути менше, ніж в D, особливо для великих значень k. Крім того, для великих значень k кожна запис може бути менше, ніж відповідна їй транзакція, т. К. В транзакції буде міститися мало кандидатів. Однак для малих знань k кожен запис може бути більше, ніж відповідна транзакція, тобто Ck включає всіх кандидатів k-елементних наборів, що містяться в транзакції [13].

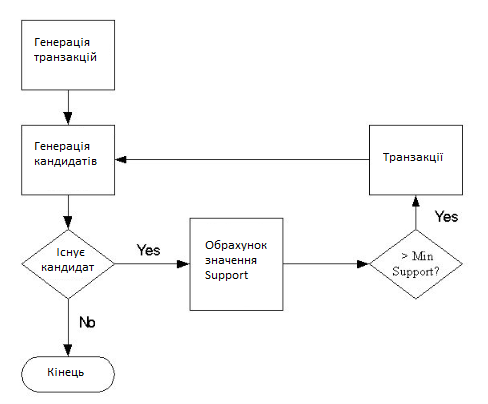


Рис. 2.1. Блок-схема роботи алгоритму AprioriTID

Інший вид алгоритму Apriori - алгоритм MSAP (Mining Sequential Alarm Patterns), спеціально розроблений для проведення послідовного аналізу мережних телекомунікаційних збоїв. Він підтримує такі функції: для будь-якої послідовності Lk його підтримка буде меншою, ніж підтримуючі послідовності з набору Lk1. Алгоритм MSAP використовує концепцію "Urgent Window" для пошуку подій, які слідують один за одним. Це дозволяє виявити не тільки однакові послідовності подій, але й наступні послідовності подій. В іншому випадку цей алгоритм працює на тому ж принципі, що й Апріорі.

* 1. **Опис можливих модифікацій**

Отже, переглянувши декілька відомих модифікацій, проаналізувавши предметну область та зрозумівши принцип роботи алгоритму та пошуку асоціативних правил було запропоновано модифікації для алгоритму.

Відомо, що алгоритм на кожному кроці визначає:

* підтримку (англ. support) - скільки разів у всьому масиві використано елементи даних, що складаються з X та Y
* достовірність (англ. confidence) - який відсоток від всіх одиниць, що містять X, містить також і Y

Обраховуючи Ліфт на кожному кроці алгоритму можна здійснити більше відсіканання абсурдних або випадкових груп правил, оттже швидкість алгоритму повинна зрости адже не потрібно здійснюватинаступні кроки по вже відкинутих правилах.

Цей підхід можна спробувати для вище перелічених методів, а також спробувати даний алгоритм при різних значеннях коефіцієнтів підтримки та достовірності.

Ще одним із можливих варіантів модифікації розбиття множини всіх можливих транспортних пригод I = {Is, Id} на дві підмножини:

* Is – множина характеристик, дані якої для нашого випадку є абсурдними або загальновідомими фактами (стан алкогольного сп’яніння, технічний стан автомобіля, вік водія, вік авто тощо).
* Id – множина характеристик, дані якої для нашого випадку будуть цікавими (місце проживання – село/місто, освіта, колір машини, кількість пасажирів, порядок власника авто тощо).

Асоціативні правила, що містять лише підмножину факторів із множини Si , не містять\* корисної інформації для виконання подальшого аналізу, завдяки чому можна прискорити алгоритм пошуку асоціативних правил. Для цього необхідно змінити процес генерації наборів кандидатів Ck , в якому відсікати набори, що не містять факторів із множини Di. В результаті значно зменшиться кількість наборів кандидатів Ck , що генеруватиметься на кожній ітерації алгоритму, а також це впливатиме на генерацію частих наборів Lk. Час обробки множини наборів Ck та генерації Lk лінійно залежить від потужності даних множин і загальної потужності множини І, яка є сталою в процесі роботи алгоритму. Тому завдяки зменшенню кількості елементів множини Ck зменшиться загальний час роботи алгоритму.

* 1. **Висновок до розділу 2**

Здійснено обгрунтування та порівняльний аналіз алгоритму, визначено що алгоритм є простий у розумінні та схильний до модифікацій.

Запропоновано покращення базового алгоритму, за допомогою застосування характиристики Ліфту на кожному кроці алгоритму. Даний підхід зменшить витрати часу при знаходженні асоціативних правил. Також є можливим впровадження даного підходу що уже відомих модифікацій, а також спробувати даний алгоритм при різних значеннях коефіцієнтів підтримки та достовірності.

Також запропоновано ще один варіант у якому вхідна множина розділяється на дві підмножини, перша множина це множина із факторами які мали місце до виникнення асоціацій і друга – множина із факторами що стали результатом виникнення асоціацій, на кожному кроці відсікається перша множина, адже вона не містить корисної інформаці.

# Розділ 3. ПРОГРАМНА РЕАЛІЗАЦІя АЛГОРИТМУ apriori



## Загальна постановка задачі

Реалізувати програмний засіб, за допомогою якого можна здійснювати інтелектуальний аналіз даних, а саме пошук асоціативних правил. Даний програмний засіб повинен надавати можливість для проведення аналізу вхідних даних та виведення результатів аналізу (знайдені асоціативні правила, їх кількість та швидкодію). Після виконання програма повинна формувати звітність на основі результатів аналізу представлених даних. Результати представлятимуться безпосередньо на екрані із можливістю збереження їх у файл.

Провести дослідження ефективності алгоритму. На основі отриманих даних із експериментів визначити можливість видозміни алгоритму для збільшення ефективності його роботи. При позитивних результатах виконати модифікацію алгоритму.

## Специфікація вимог

**3.2.1. Вступ**

Призначення даного продукту – автоматичний аналіз великих наборів даних та пошук в них асоціативних правил, формування звіту.

Мета – реалізувати класичний алгоритм apriori та модифіковані його версії та порівняти результати роботи у вигляді програмного продуку.

**3.2.2. Загальний опис**

**3.2.2.1. Характеристики продукту**

Функції, що будуть реалізовані у програмі:

* Зчитування даних з файлу;
* Аналіз даних;
* Виведення результатів у вигляді таблиць та діаграм;
* Формування звіту про результати аналізу;

**3.2.2.2. Класи користувачів та їх характеристики**

В даній програмі передбачено лише один клас користувачів, тому функціонал не буде мати обмеження.

Програмний засіб призначений для бізнес аналітиків, щоб ефективно провести інтелектуальний аналіз даних та сформувати звіт замовнику.

**3.2.2.3. Середовище функціонування**

Апаратні вимоги:

* Частота процесора 1 GHz;
* Оперативна пам’ять 512 Mb;
* 10 Mb вільного простору на диску.

Системні вимоги:

* Операційна система Microsoft Windows XP або вище;
* .Net Framework версії 4.0 і вище;

**3.2.3. Характеристики системи**

**3.2.3.1. Пошук асоціативних правил для введених даних**

3.2.3.1.1. Опис і пріоритет

Аналіз введених даних, пошук залежностей. Виведення результатів тестування у зручному для користувача вигляді, у вигляді таблиць та діаграм, із можливістю збереження на локальний диск.

Пріоритет – високий.

3.2.3.1.2. Послідовності дія/відгук

Користувач вибирає текстовий документ із даними, аналіз яких потрібно провести. Далі завантажує список характеристик написнувши на кнопку «Завантажити I». Потім задає коефіцієнти підтримки, достовірності та ліфта, при потребі завантажує характеристики що описують ДТП. Натискає на кнопку «Розпочати» і у низу вікна з’являється анімація завантаження.

3.2.3.1.3. Функціональні вимоги

REQ-1.1: Зчитування даних із .txt формата;

REQ-1.2: Проведення аналізу;

REQ-1.3: Пошук асоціативних правил;

REQ-1.4: Виведення повідомлення про закінчення проведення аналізу;

REQ-1.5: Виведення результатів;

**3.2.3.2. Формування звіту результату**

3.2.3.2.1. Опис і пріоритет

Створення звіту, який буде відображати інформацію про результат пошуку залежностей. Звіт можна зберегти на диск у форматі txt.

3.2.3.2.2. Послідовності дія/відгук

Для збереження звіту потрібно натиснути кнопку «Зберегти звіт», після чого появиться діалогове вікно для вибору місця збереження файлу.

3.2.3.2.3. Функціональні вимоги

REQ-2.1: Формування звіту;

REQ-2.2: Збереження на диску у форматі txt;

**3.2.3.3. Перегляд результатів виконання**

3.2.3.3.1. Опис і пріоритет

Перегляд результатів виконання програми у вигляді діаграм та таблиці

3.2.3.3.2. Послідовності дія/відгук

Для перегляду результатів потрібно натиснути кнопку «Результати», після чого появиться нове вікно із результатами виконання.

3.2.3.3.3. Функціональні вимоги

REQ-3.1: Формування звіту;

REQ-3.2: Збереження на диску у форматі txt;

**3.2.4. Вимоги зовнішніх інтерфейсів**

**3.2.4.1. Користувацькі інтерфейси**

Інтерфейс програми повинен бути простим для розуміння, забезпечувати швидкодію відповідно до вимог та дозволяти зручно виконувати необхідні операції.

**3.2.4.2. Програмні інтерфейси**

* Бібліотека Microsoft.Office.Interop.Excel;
* Бібліотека Xml.Serializer;

**3.2.5. Інші нефункційні вимоги**

**3.2.5.1. Вимоги продуктивності**

Швидкість проведення аналізу буде прямо залежати від кількості завантажених даних, коефіцієнтів роботи алгоритму та від потужності процесора.

**3.2.5.2. Вимоги безпеки**

Немає вимог безпеки.

**3.2.5.3. Атрибути якості програмного продукту**

Програма реалізована із мінімальним основним набором функції, саме тому вона зрозуміла та зручна для використання користувачам, які з нею ніколи не працювали;

Ніяким чином не впливає на роботу інших програм.

**3.2.5.4. Інші вимоги**

Необхідна наявність .Net Framework версії 4.0 і вище.

## Використані технології

Оскільки основним завданням проекту буде аналіз роботи алгоритму то було обрано технологію .Net, середовище розробки Microsoft Visual Studio, мова C# та WPF[18].

Для зберігання поточного стану розробленої системи використано систему контролю версій Git - це система керування версіями для відстеження змін у комп'ютерних файлах та координаціїіроботи над цими файлами серед декількох людей. Він використовується, перш за все, для управління вихідним кодом у розробці програмного забезпечення [18], але йогоіможнаівикористовувати для відстеження змініу будь-якомуінаборі файлів. В якості розподіленої системи перегляду-контролю вонаіспрямованаіна швидкість, цілісністьіданих та підтримку розподілених, нелінійних робочих процесів.

.NET Framework є основою програмного забезпечення, розроблена корпорацією≥Майкрософт, яка працює≥в основному на Microsoft Windows. Вона містить велику бібліотеку класів під назвою Framework Class Library (FCL) і забезпечує мовну сумісність (кожна мова може використовувати код, написаний на інших мовах) на кілька≥мов програмування. Програми, написані для .NET Framework виконуються в програмному≥середовищі (на відміну від апаратної середовища) під≥назвою Common≥Language Runtime (CLR), віртуальна≥машина додаток, яке надає≥такі послуги, як безпека, управління≥пам'яттю і обробки винятків.

Microsoft Visual Studio є інтегрованим середовищем розробки Microsoft (IDE). Він використовується для розробки комп'ютерних програм, а також веб-сайтів, веб-програм, веб-сервісів та мобільних додатків. Visual Studio використовує платформи для розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store і Microsoft Silverlight. Він може створювати як власний код, так і керований код.[18]

Visual Studio включає редактор коду, що підтримує IntelliSense, а також рефакторинг коду. Вбудований відладчик працює як відладчик на рівні вихідного рівня та налагоджувача на рівні машини. Інші вбудовані інструменти включають програму для кодування, дизайнеру форми для побудови графічних інтерфейсів, веб-дизайнера, дизайнера класу та дизайнера схеми баз даних. Він приймає плагіни, які покращують функціональність практично на всіх рівнях, включаючи підтримку систем управління ресурсами (наприклад, Subversion та Gіt) та додавання нових наборів інструментів, таких як редактори та візуальні розробники для мов або наборів інструментів для інших аспектів розробки програмного забезпечення життєвий цикл.[18-19]

C # — є об’єктно-орієнтованою мовою≥програмування≥яка охоплює строгу типізацію, імперативність, декларативність, функціональні, загальні, об'єктно-орієнтовані (на основі класів), компонент-орієнтоване≥програмування дисциплін. C # є одним з мов програмування, призначених для Common Language Infrastructure.

Мова призначена як проста, сучасна, загальноприйнята, об'єктно-орієнтована мова програмування.

Мова та її реалізація повинні забезпечувати підтримку принципів розробки програмного забезпечення, таких як перевірка сильного типу, перевірка кордону, виявлення спроб використання неініціалізованих змінних та автоматичного збирання сміття. Важливою є надійність програмного забезпечення, надійність і продуктивність програміста.[19]

Мова призначений для розробки програмних компонентів, які підходять для розгортання в розподілених середовищах.

Переносимість дуже важлива для вихідного коду та програмістів, особливо тих, хто вже знайомий з C і C ++.

Інтернаціоналізаційна підтримка дуже важлива.

C # призначений для написання програм для хостингу та вбудованих систем, починаючи від дуже великих, використовуючи складні операційні системи, до дуже малих спеціальних функцій.

Хоча додатки C # є економічними з точки зору вимог до пам'яті та керування енергією, мова не мав на меті прямо конкурувати з продуктивністю чи розміром з C або асемблером.

На відміну від C ++, C # не підтримує множинне успадкування, хоча клас може реалізовувати будь-яку кількість інтерфейсів. Це було рішення дизайнера провідного архітектора мови, щоб уникнути ускладнень та спростити архітектурні вимоги в рамках CLI. При виконанні декількох інтерфейсів, що містять метод з тим самим підписом, C # дозволяє реалізувати кожен метод залежно від того, який інтерфейс цей метод називається, або, наприклад, Java, дозволяє реалізувати цей метод один раз, і це єдиний виклик до дзвінка через будь-який інтерфейс класу.

C #, покажчики адрес пам'яті можуть використовуватися лише в блоках, спеціально позначених як небезпечні, а програми з небезпечним кодом потребують відповідних дозволів для запуску. Більшість доступ до об'єкта здійснюється через безпечні посилання на об'єкти, які завжди вказують на "живий" об'єкт або мають чітко визначене нульове значення; неможливо отримати посилання на "мертвий" об'єкт (той, який був зібраний сміттям) або випадковий блок пам'яті. Небезпечний покажчик може вказувати на екземпляр типу-значення, масиву, рядка або блоку пам'яті, виділеного стеку. Код, який не позначено як «небезпечний», може зберігати і керувати покажчиками через тип System.IntPtr, але він не може їх вилучити.

Неможливо явним чином звільнити керовану пам'ять; замість цього автоматично збирають сміття. Мусорна колекція вирішує проблему витоків пам'яті, звільняючи програміста від відповідальності за випуск пам'яті, яка більше не потрібна.

Для малювання діаграм використано бібліотеку DataVisualization, яка дозволяє легко та прозоро малювати різного роду діаграми.

Windows Presentation Foundation (WPF) - це Microsoft графічна підсистема для відображення користувацьких інтерфейсів у додатках Windows. WPF, раніше відомий як "Avalon", спочатку був випущений в рамках .NET Framework 3.0 у 2006 році. WPF використовує DirectX і намагається забезпечити послідовну модель програмування для створення додатків. Він відокремлює інтерфейс користувача від бізнес-логіки і нагадує подібні XML-орієнтовані об'єктні моделі, такі як ті, що реалізовані в XUL та SVG.[19]

WPF використовує XAML, мову на базі XML, для визначення та поєднання різних елементів інтерфейсу. Програми WPF можуть бути розгорнуті як окремі настільні програми або розміщені як вбудований об'єкт на веб-сайті. WPF має на меті об'єднати ряд загальних елементів користувальницького інтерфейсу, таких як 2D / 3D рендеринг, фіксовані та адаптивні документи, типографіка, векторна графіка, анімація часу виконання та попередньо відтворені носії. Потім ці елементи можуть бути пов'язаними та маніпулювати на основі різних подій, взаємодії користувачів та прив'язування даних.

Бібліотеки бібліотеки WPF входять до всіх версій Microsoft Windows, починаючи з Windows Vista та Windows Server 2008. Користувачі Windows XP SP2 / SP3 та Windows Server 2003 можуть додатково встановлювати необхідні бібліотеки.

## Реалізація інтерфейсу користувача програмного засобу для роботи алгоритму apriori

Розроблений інтерфейс користувача складається із мінімального набору компонентів, що робить його дуже зручним та простим у користуванні, в той самий час, даний інтерфейстдаєтзмогу користувачу в повній мірітзастосувати весь функціоналтрозробленого засобу.

Інтерфейс користувача складається з основного вікна та додаткового (див. Рис. 3.1. та Рис. 3.2.). Основне вікно застосунку дозволяє користувачу завантажити список транзакцій(список із ДТП), завантажити характеристики ДТП, завантажити характеристики що описують виникнення ДТП, а також встановити коефіцієнти підтримки достовірності та ліфта. Також, на головному вікні користувач має можливість почати процес аналізу великих даних алгоритмом apriori, та побачити результати виконання алгоритму результати (кількість знайдених правил чис виконання, кількість транзакій тощо.).

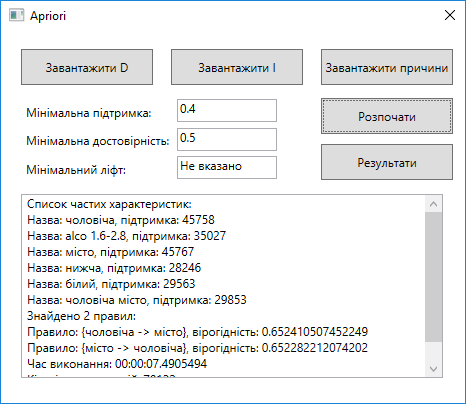


Рис. 3.1. Користувацький інтерфейс основного вікна програми

Вікно «Результати» відповідає за відображення результатів програми та порівняння їх із попередніми значеннями. Дане вікно складається дане вікно складається із трьох діаграм, таблиці та кнопки «Зберегти»(Рис. 3.2.). Таблиця представляє список із результатами виконання програми та такими колонками як: «Спроба №» - показує номер спроби, «Час виконання» - час виконання анлізу для спроби, «Кількість транзакцій» - кількість транзакцій у початковому списку, «Кількість правил» - кількість знайдених асоціативних правил.

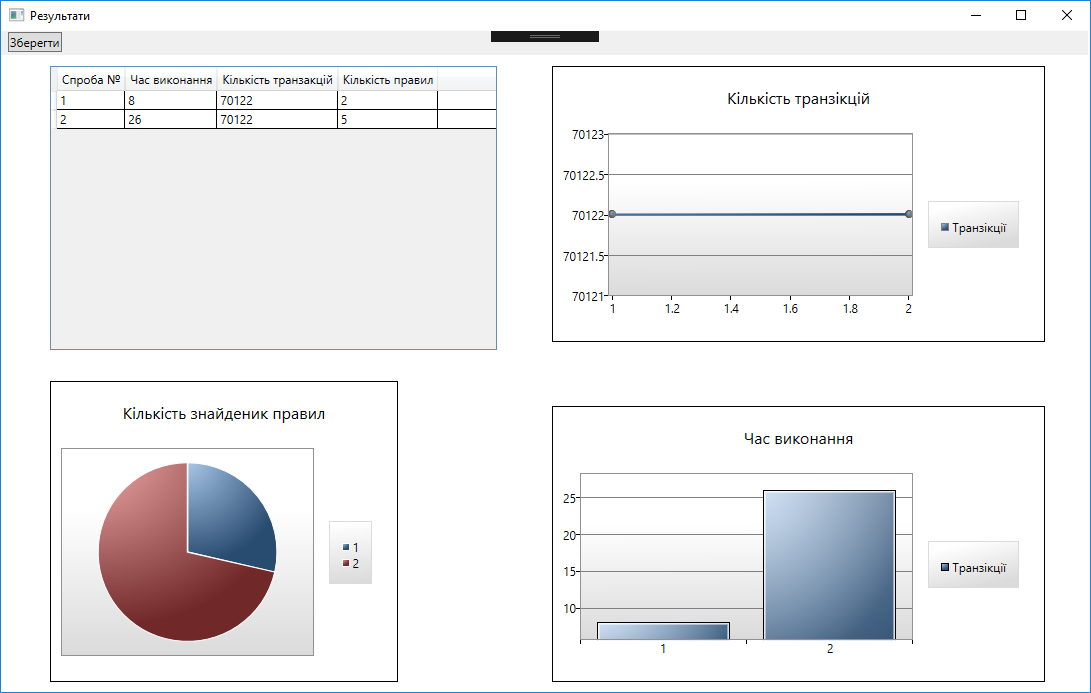


Рис. 3.2. Користувацький інтерфейс із результатами виконання програми

Таблиця «Кількість знайдених правил» показує відношення у відсотках скільки правил було знайдено для кожної спроби.

Таблиця «Кількість транзакцій» показує скільки транзакцій було в початковій базі даних.

Таблиця «Час виконання» показує час виконання для кожної спроби.

Якщо користувачцвводить неправильне значення змінної (наприклад, для змінної типу int – стрічку), програмацнецдасть змоги зберегтиезміни і попередить проецеекористувача(Рис. 3.3.).

Отже розглянувши всі ці результати можна легко зробити висновки про результати роботи алгоритму як для мофікованих режимів так і для стандартного вигляду.

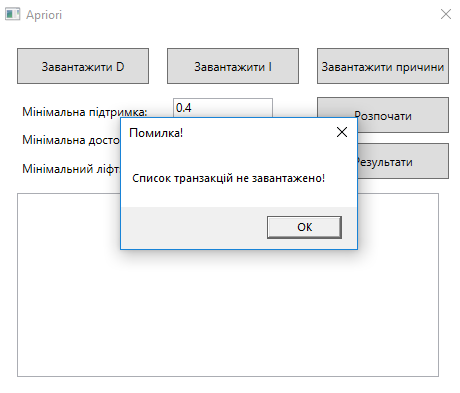


Рис. 3.3. Інтерфейс користувача, при неправильному введені даних.

## Розробка архітектури системи для роботи алгоритму apriori

Створення програми відбувається згідно з розробленою діаграмою класів програми. Розглянемо складові діаграми, щоб детальніше зрозуміти роботу програми:

* FileReader відповідає за зчитування даних із файлів та конвертання їх у масив. Після відкриття файлу з’являється діалогове вікно для вибору .txt файлу.
* Клас Item містить інформацію про кожен предмет який відповідає вравилам.
* Rule містить інформацію про правило X – лівий операнд, Y правий операнд, та Confidence – вірогідність.
* Клас Output – містить інформацію про результат роботи алгоритму.
* Основна логіка виконується а класі Apriori – клас який асимілює всі перелічені вище класи та має публічний метод ProcessTransaction, який на вхід отримує вхідні дані: масив «items», масив «transactions», масив «itemsD» мінімальне значення вірогідності, підтримки та ліфта. Та після обробки повертає результат типу Output.



Рис. 3.4. UML діаграма класів програми

* AprioriInThread – клас який забезпечує виконання алгоритму в окремому потоці щоб не блокувати користувацький інтерфейс, містить один метод «ProcessInThread» який повертає результат Task<Output>.
* Клас Result містить інформацію яка необхідна для побудови графіків та списку у вікні з результатами.

Також було розроблено UML діаграму прецедентів(Рис. 3.5) на якій зображено відношення акторів та прецедентів в системі.



Рис. 3.5. UML діаграма прецедентів

## Реалізація алгоритму apriori для аналізу ДТП

Алгоритм apriori було реалізовано згідно наведеної блок-схеми (Див. Додаток Б)

Список транзакцій завантажується із .txt файла та обробляється до потрібного вигляду, це двовимірний масив стрічок, за допомогою класу FileReader. Далі за таким самими принципом завантажується список характеристик та наслідків.

Список транзакцій разом із коефіцієнтами подається на вхід функції ProcessTransactions. Всередині цієї функції відбувається покрокова генерація кандидатів та частих наборів.

Під час генерації кандидатів обраховується їхня кількість відносно всіх транзакцій, та якщо використовується модифікована версія із завантаженими причинами то відсікаються кандидати у яких немає цих причин, завдяки такому підходу на кожній ітерації вдається відсікнути багато зайвих кандидатів і тому виконання алгоритму стає набагато швидшим.

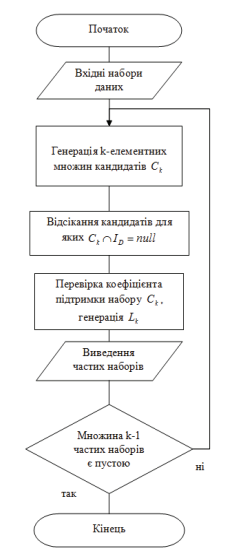


Рис. 3.6. Блок-схема модифікованого алгоритму apriori

Під час генерації частих наборів обраховується «підтримка» та порівнюється із мінімально допустимою, якщо ця умова виконується кандидат додається до частих наборів.

Усе це повторяється поки множина кандидатів не буде порожньою.

Після цього відбувається генерація асоціативних правил на основі частих наборів. Далі відбувається генерація «міцних» правил в методі GetStrongRules, де відбувається обрахунок вірогідності, та якщо використовується модифікована версія обраховується значення ліфта, такий підхід дозволяє відсікати абсурдні або випадкові правила та дозволяє збільшити точність виконання алгоритму.

Далі метод ProcessTransactions повертає список Output об’єктів з яких в подальшому можна згенерувати звіт та порівняльні діаграми.

## Висновки до розділу 3

Ключовим завданням постає розробка програмного засобу для проведення аналізу даних алгоритмом apriori. По суті програма буде працювати у певній логічній послідовності, отже варіантів дій користувача не так і багато, що робить користування просте та зрозуміле. Користувачу представляється лише вибір даних, який слід аналізувати і результати для подальшого використання.

Так як ціль проекту включає в собі інтелектуальний аналіз даних то технологія Microsoft .NET чудово підходить для написання прикладної програми. Для побудови інтерфейсу використовуватиметься технологія WPF, яка дозволить без проблем створити цікавий вигляд програми.

У цьому розділі детально описано специфікацію вимог, реалізовано інтерфейс, архітектуру системи та реалізовано алгоритм.

# Розділ 4. ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ ТА АНАЛІЗ ОТРИМАНИХ ДАНИХ

## 4.1. Опис процесу дослідження

Поданий алгоритм apriori використовується для аналізу дорожньо-транспортних пригод, а саме пошуку асоціативних зв’зків між наборами даних які стали причинами ДТП.

При застосуванні пошуку асоціативних правил для аналізу ДТП виникають проблеми:

1. Збільшення потужності множини всіх можливих ДТП.
2. Збільшення часу для аналізу даних.
3. Низька інформаційність отриманого результату.

Запропоновано наступні модифікації для розв'язання цих проблем:

1. Розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини:[3]

* *Is* – множина характеристик, дані якої для нашого випадку є абсурдними або загальновідомими фактами (стан алкогольного сп’яніння, технічний стан автомобіля, вік водія, вік авто тощо).
* *Id* – множина характеристик, дані якої для нашого випадку будуть цікавими (місце проживання – село/місто, освіта, колір машини, кількість пасажирів, порядок власника авто тощо).

Асоціативні правила, що містять лише підмножину із множини Is не містять корисної інформації, тому на кожному кроці після генерації кандидатів можна відсікати кандидатів, для яких Ck ∩ Id = null.

У результаті застосування такого підходу збільшується швидкість роботи алгоритму та ступінь інформаційності отриманих асоціативних правил.

1. Додати ще одну характеристику асоціативних правил Ліфт (англ. lift) - це відношення частоти появи умови в транзакціях, які містять й умову, і наслідок, до частоти появи наслідку в цілому.[4] Обраховуючи Ліфт на кожному кроці алгоритму можна здійснити більше відсікання абсурдних або випадкових груп правил, отже швидкість алгоритму повинна зрости адже не потрібно здійснювати наступні кроки по вже відкинутих правилах.

Отже, для якісної оцінки запропонованих модифікацій потрібно

згенерувати базу даних із причинами які спровокували виникнення дорожньо-транспортних пригод.

Перш за все, потрібно здійснити аналіз стандартного алгоритму без модифікацій.

База даних складається із таких характеристик:

* Колір (білий, чорний, червоний, зелений, жовтий);
* Кількість пасажирів в салоні на момент ДТП (пасажири 0, пасажири, пасажири 2, пасажири 3, пасажири 4, пасажири 4+);
* Порядок власності (власник 1, власник 2, власник 3, власник 4+);
* Вік автомобіля (1-5, 6-10, 11-15, 16-20);
* Освіта (вища, середня, базова);
* Місце проживання (місто, село);
* Пора року (літо, осінь, зима, весна);
* Пора доби (ранок, обід, вечір, ніч);
* Рівень алкоголю в крові (alco 0, alco 0.2-0.5, alco 0.6-0.9, alco 1.0-1.5, alco 1.6-2.8);
* Вік водія (15-17, 18-21, 22-25, 26-30, 31-35, 36-40, 41-45, 46-50);
* Стать (чоловіча, жіноча);

Другим етапом дослідження є порівняння роботи стандартного алгоритму із запропонованою першою модифікацією – розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини.

Третім етапом буде порівняння роботи стандартного алгоритму із запропонованою другою модифікацією.

На наступному етапі можна порівняти між собою три попередні варіанти.

На завершення потрібно підібрати найкращі варіанти коефіцієнтів алгоритму, які б показували найкращі результати в швидкодії та точності пошуку асоціативних правил.

## 4.2. Аналіз ароботи алгоритму без модифікацій

Після генерації даних було здійснено аналіз роботи алгоритму без модифікацій.

Результати застосування розробленого засобу для аналізу роботи алгоритму apriori без додаткових модифікацій для дорожньо-транспортних пригод наведені в Таблиці 4.1.

*Таблиця 4.1.*

Результати виконання розробленого засобу

|  |  |
| --- | --- |
| Номер досліду | Час виконання, мс |
| 1 | 40 409 |
| 2 | 40 407 |
| 3 | 42 354 |
| 4 | 45 373 |
| 5 | 41 295 |
| 6 | 45 573 |
| 7 | 43 401 |
| 8 | 42 563 |
| 9 | 45 765 |
| 10 | 43 642 |

Таким чином, середній час виконання алгоритму, виміряний розробленим засобом становить 42 878.2мс.

Також, аплікація згенерувала графік на якому видно, що дані підтверджено. Результат представлений на рис. 4.1.



Рис. 4.1. Графік із зображенням відношення часу виконання до кожної транзакції

Таким чином, дані які представлено вище можна вважати еталоном для нашого дослідження, так як це результати роботи алгоритму без будь яких додаткових втручань.

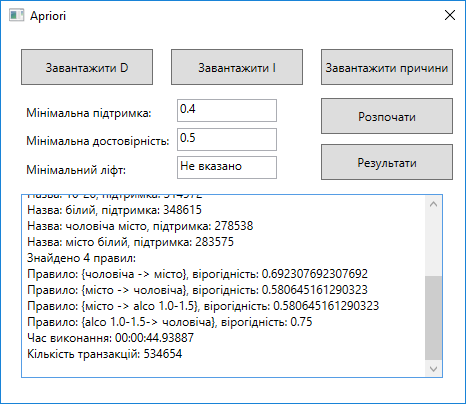


Рис. 4.2. Знімок результатів виконання програми

Отже, середнє значенням часу виконання алгоритму дорівнює 42 878.2 мс

для даних обсягом 230мб та кількістю транзакцій 1 000 000 одиниць, було знайдено 4 міцних правила таких як:

{чоловіча –> місто}, вірогідність: 0.692307692307692

{місто –> чоловіча}, вірогідність: 0.580645161290323

{місто –> alco 1.0-1.5}, вірогідність: 0.580645161290323

{alco 1.0-1.5 –> чоловіча}, вірогідність: 0.75

Також знайдено 11 частих наборів, які перевищують значення мінімального значення підтримки. Мінімальне значення підтримки дорівнює 0.4 та мінімальне значення вірогідності 0.5.

## 4.3. Аналіз роботи алгоритму із модифікаціями

Для початку дослідимо роботу алгоритму для модифікації №1 – Розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини.

Для множини Id – множина із характеристиками що описують ДТП, було обрано такі категорії:

* Колір (білий, чорний, червоний, зелений, жовтий);
* Кількість пасажирів в салоні на момент ДТП (пасажири 0, пасажири, пасажири 2, пасажири 3, пасажири 4, пасажири 4+);
* Кількість власників (власник 1, власник 2, власник 3, власник 4+);
* Вік автомобіля (1-5, 6-10, 11-15, 16-20);
* Освіта (вища, середня, базова);
* Місце проживання (місто, село);
* Стать (чоловіча, жіноча);

Для множини Is – множина характеристик, дані якої для нашого випадку є абсурдними або загальновідомими фактами

* Пора року (літо, осінь, зима, весна);
* Пора доби (ранок, обід, вечір, ніч);
* Рівень алкоголю в крові (alco 0, alco 0.2-0.5, alco 0.6-0.9, alco 1.0-1.5, alco 1.6-2.8);
* Вік водія (15-17, 18-21, 22-25, 26-30, 31-35, 36-40, 41-45, 46-50);

Таким чином, середній час виконання алгоритму, виміряний розробленим засобом становить 38 178.2мс.

Також, результати можна переглянути на графіку. Результат представлений на рис. 4.3.

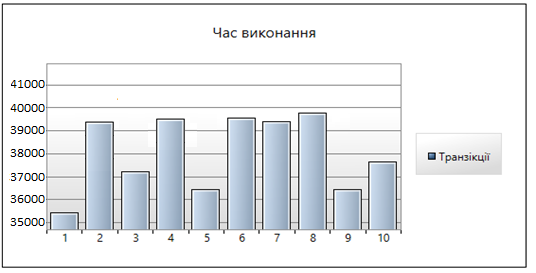


Рис. 4.3. Графік із зображенням відношення часу виконання до кожної транзакції

Середнє значенням часу виконання модифікованого алгоритму №1 дорівнює 38 178.2мс, для даних обсягом 230мб та кількістю транзакцій 1 000 000 одиниць, було знайдено 4 міцних правила та 11 частих наборів, які перевищують значення мінімального значення підтримки. Мінімальне значення підтримки дорівнює 0.4 та мінімальне значення вірогідності 0.5.

Отже, можна побачити, що час роботи алгоритму скоротився на 11% при цьому кількість знайдених частих наборів і «міцних» правил не зменшилась.

Приступимо до аналізу роботи алгоритму для модифікації №2 – додавання ще однієї характеристику асоціативних правил Ліфта.Обраховуючи Ліфт на кожному кроці алгоритму можна здійснити більше відсікання абсурдних або випадкових груп правил. Проте перш ніж здійснити аналіз потрібно уточнити що застосувати Ліфт на етапі генерації кандидатів не вийшло так як логіка алгоритму не дозволяє цього зробити, ліфт можна застосувати при генерації асоціативних правил.

Таким чином, середній час виконання алгоритму, виміряний розробленим засобом становить 43 132.2мс.

Як видно із результатів виконання алгоритму дані не сильно відрізняються від еталонних(результати які було отримано із базового алгоритму) в деяких випадках результати навіть гірші. На практиці додавання характеристики Ліфта до роботи алгоритму нічого не або навіть погіршило результати. Проте це тільки для нашого конкретного випадку і бази даних із дорожнього-транспортними пригодами можливо приінших випадках результатит були б інакшими.

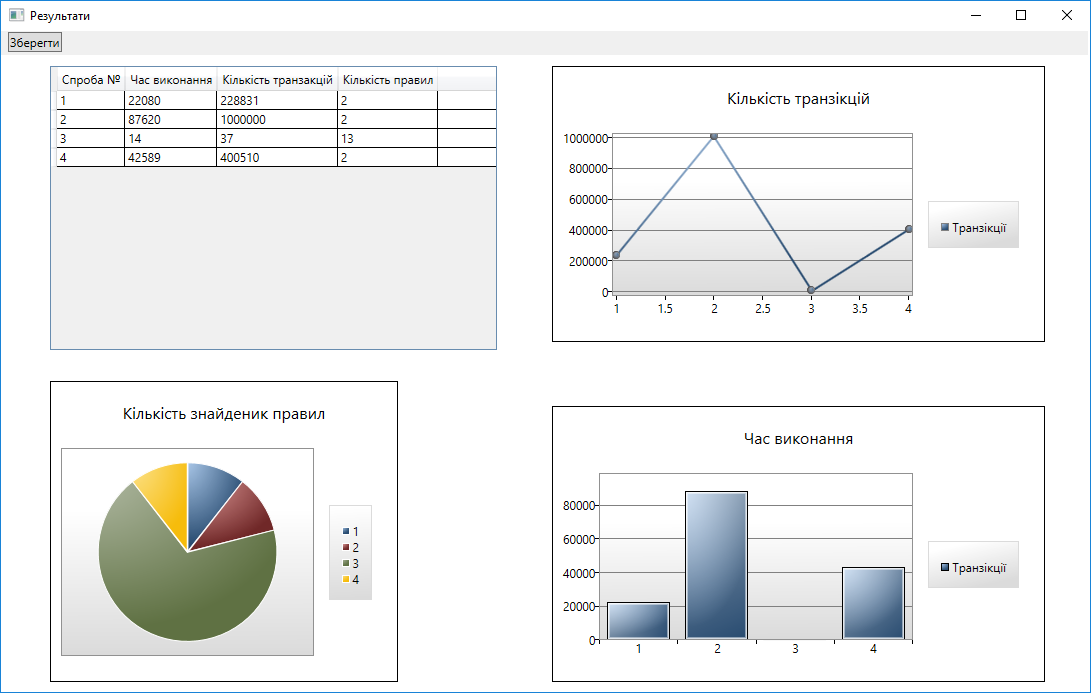


Рис. 4.5. Результати виконання програми

Отже, середнє значенням часу виконання алгоритму дорівнює 43 132.2 мс

для даних обсягом 230мб та кількістю транзакцій 1 000 000 одиниць, було знайдено 4 міцних правила та 11 частих наборів, які перевищують значення мінімального значення підтримки. Мінімальне значення підтримки дорівнює 0.4 мінімальне значення вірогідності 0.5 та мінімальне значення Ліфта дорівнює 0.9.

Після генерації іншої бази даних було проведено додакове дослідження для впливу Ліфта на результати аналізу.

Результати застосування розробленого засобу для аналізу роботи алгоритму apriori із модифікацією №2 для дорожньо-транспортних пригод наведені в Таблиці 4.2.

*Таблиця 4.2.*

Результати виконання розробленого засобу

|  |  |  |
| --- | --- | --- |
| Номер досліду | Кількість знайдених правил | Значення Ліфта |
| 1 | 75 | Не вказано |
| 2 | 64 | 0.95 |
| 3 | 56 | 1.05 |
| 4 | 45 | 1.2 |
| 5 | 35 | 1.4 |

Як видно із результатів виконання алгоритму зі збільшенням значення Ліфта деякі збільшується відсікання знайдених правил із цього виходить підтвердження теоретичних міркувань, що при додаванні значення Ліфта багато абсурдних або випадкових правил будуть відкинуті.

Так як, дослідження проводилось на невеликій базі із кількістю транзакцій 70 одиниць, час не було враховано в цьому дослідженні.

Було здійсненно порівняльний аналіз для різних значень коефіцієнтів

Результати роботи алгоритму apriori для різних значень коефіцієнтів наведені в Таблиці 4.3.

*Таблиця 4.3.*

Результати виконання розробленого засобу

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер досліду | Час виконання, мс | Кількість знайдених правил | Значення підтримки | Значення достовірності | Значення Ліфта | Причини завантажено |
| 1 | 38784 | 2 | 0,4 | 0,5 | Не вказано | Ні |
| 2 | 25751 | 2 | 0.4 | 0,5 | Не вказано | Так |
| 3 | 36118 | 15 | 0,3 | 0,5 | Не вказано | Так |
| 4 | 38118 | 18 | 0,3 | 0,5 | Не вказано | Ні |
| 5 | 26680 | 4 | 0,35 | 0,5 | Не вказано | Так |
| 6 | 26778 | 2 | 0,35 | 0,5 | 1 | Так |
| 7 | 30564 | 6 | 0,3 | 0,6 | Не вказано | Так |

Усі вище перелічені досліди було проведено для однієї і тієї ж бази розміром 530000 транзакцій. Отже, проаналізувавши результати можна зробити висновок, що найкращим результатом по швидкодії є варіанті із застосування методу модифікації №2 та із коефіцієнтами: достовірність 0,5 , підтримки 0,4. Найточніший варіант, коли було знайдено найбільше правил це варіант базового алгоритму та коефіцієнтами: достовірність 0,5 , підтримки 0,3.

## 4.4. Висновки до розділу 4

У цьому розділі було проведено дослідження алгоритму Apriori та його модифікацій. Дослідження здійснено в такій послідовності: перш за все, було проведено аналіз базового алгоритму без модифікацій, далі було проведено порівняння роботи стандартного алгоритму із запропонованою першою модифікацією – розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини, третім етапом було порівняння роботи стандартного алгоритму із запропонованою другою модифікацією. Останнім етапом було порівняння між собою трьох попередніх варіантів.

Отже, після здійснення досліджень можна сказати що запропонований варіанти модифікації №1 є найкращим у нашому випадку, за цим методом далось скоротити час виконання алгоритму на 11%, при цьому не втративши жодної важливої інформації.

Варіант модифікації №2 не вдалось реалізувати як було описано в теорії, тому що це руйнує логіку алгоритму, проте можна застосувати при генерації асоціативних зв’язків але результати для нашого випадку, аналіз ДТП, були гіршими за еталон.