PARTYCJONOWANIE GRAFÓW A OPTYMALIZACJA WYKONANIA ZBIORU ZAPYTAŃ EKSPLORACYJNYCH

Marek WOJCIECHOWSKI *, Maciej ZAKRZEWICZ **

Streszczenie. Optymalizacja wykonania zbioru zapytań eksploracyjnych polega na takim podziale zbioru zapytań na rozłączne podzbiory zapytań wykonywanych współbieżnie, aby sumaryczny koszt I/O ich realizacji był minimalny. W artykule proponujemy transformację problemu optymalizacji wykonania zbioru zapytań eksploracyjnych odkrywania zbiorów częstych do problemu partycjonowania grafu, intensywnie badanego w wielu dziedzinach nauki. Redefiniujemy klasyczny problem partycjonowania grafu oraz pokazujemy, w jaki sposób istniejące algorytmy heurystyczne mogą być zaadaptowane w celu rozwiązania problemu optymalizacji wykonania zbioru zapytań eksploracyjnych.

1. Wprowadzenie

Optymalizacja zbiorów zapytań [15] jest popularną dziedziną badań nad bazami danych, której celem jest wspólna optymalizacja wielu zapytań poprzez wyodrębnienie i jednokrotne wykonanie powtarzających się wyrażeń składowych [2][11]. Głównym zadaniem w optymalizacji zbiorów zapytań jest identyfikacja wspólnych wyrażeń oraz konstrukcja globalnego planu wykonania. W tradycyjnych systemach baz danych, wspólnymi wyrażeniami mogą być projekcje, selekcje, połączenia, półpołączenia i przesłania sieciowe. Kiedy wspólne wyrażenia zostały zidentyfikowane, następuje ich ewaluacja i materializacja wyników (statyczna lub dynamiczna) – operacja taka prowadzona jest jednokrotnie dla całego zbioru wykonywanych zapytań, zamiast być powtarzaną

* Politechnika Poznańska, Wydział Informatyki i Zarządzania, ul. Piotrowo 3a, 60-965 Poznań, e-mail: marek@cs.put.poznan.pl

** Politechnika Poznańska, Wydział Informatyki i Zarządzania, ul. Piotrowo 3a, 60-965 Poznań, e-mail: mzakrz@cs.put.poznan.pl

podczas realizacji każdego zapytania składowego. Na potrzeby tradycyjnej optymalizacji wykonania zbioru zapytań zaproponowano wiele algorytmów heurystycznych [14][16].

Szczególnym rodzajem zapytania do bazy danych jest zapytanie eksploracyjne [9], które deklaratywnie opisuje zadanie eksploracji danych. Zapytanie eksploracyjne definiuje ograniczenia, którym podlegają eksplorowane dane, a także ograniczenia dla odkrywanych wzorców. Zapytania eksploracyjne mogą być wyrażone z użyciem jednego z wielu języków zapytań eksploracyjnych [5][7][10][13]. Zapytania eksploracyjne są przekazywane do realizacji systemowi eksploracji danych [9], który jest systemem zarządzania bazą danych wzbogaconym o funkcjonalność eksploracji danych. Obecne systemy eksploracji danych wykonują zapytania w sposób szeregowy i nie poszukują żadnych powtarzających się wyrażeń lub operatorów.

Zapytania eksploracyjne są często wykonywane wsadowo w zbiorach 10-100 zapytań, przetwarzanych w okresie niskiej aktywności użytkowników systemu (np. w godzinach nocnych). Zapytania takie mogą wykazywać wiele podobieństw obejmujących ograniczenia dla eksplorowanych danych i ograniczenia dla odkrywanych wzorców. W przypadku, gdy zapytania takie są wykonywane szeregowo, może okazać się, że wiele operacji I/O jest marnotrawionych, gdyż te same bloki bazy danych są odczytywane przez wiele zapytań eksploracyjnych. Gdyby takie powtarzalne operacje I/O zostały zintegrowane i wykonane jednorazowo, moglibyśmy dokonać redukcji całkowitego kosztu I/O wykonania zbioru zapytań eksploracyjnych.

Tradycyjne metody optymalizacji wykonania zbiorów zapytań nie są stosowalne do zbiorów zapytań eksploracyjnych. Zapytania eksploracyjne dokonują operacji odczytu bardzo dużych fragmentów bazy danych, a te nie mogą i nie powinny być statycznie materializowane. Ponadto, zapytania eksploracyjne mają wysokie wymagania w zakresie alokowanej pamięci operacyjnej, co utrudnia lub uniemożliwia dynamiczną materializację wyników pośrednich. Jedną z metod zaproponowanych w przeszłości przez autorów tego artykułu jest Apriori Common Counting dla problemu odkrywania zbiorów częstych [1]. Metoda Apriori Common Counting jest oparta na klasycznym algorytmie Apriori [3] i służy integracji faz wyliczania wsparcia zbiorów-kandydatów – haszowe drzewa kandydatów dla wielu zapytań eksploracyjnych są wspólnie umieszczane w pamięci operacyjnej, a następnie wykonywana jest pojedyncza operacja odczytu bazy danych na potrzeby zliczania zbiorów-kandydatów należących do wielu zapytań eksploracyjnych. Podstawowa metoda Apriori Common Counting [17] zakłada, że haszowe drzewa kandydatów dla wszystkich zapytań eksploracyjnych mieszczą się w dostępnej pamięci operacyjnej, co nie w każdych warunkach może być spełnione. Jeżeli dostępna pamięć operacyjna może pomieścić struktury tylko części zapytań eksploracyjnych, to konieczny jest podział oryginalnego zbioru zapytań na rozłączne podzbiory, nazywane fazami [18]. Sposób, w jaki ten podział zostanie przeprowadzony, bezpośrednio wpływa na całkowity koszt realizacji zbioru zapytań eksploracyjnych. W ramach dotychczasowych badań zaproponowaliśmy kilka heurystycznych algorytmów podziału zbioru zapytań eksploracyjnych [18][19].

2. Optymalizacja zbioru zapytań eksploracyjnych

2.1. Definicja problemu

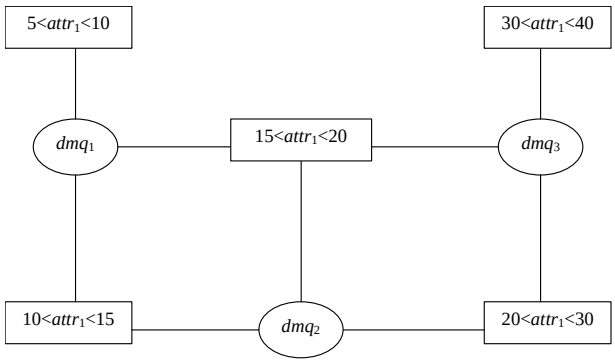
Rozważamy system zarządzania bazą danych, który jest zdolny do wykonywania algorytmów eksploracji na żądanie użytkownika. Żądania użytkownika są formułowane w postaci szczególnych zapytań, nazywanych zapytaniami eksploracyjnymi. Zapytanie eksploracyjne odkrywania zbiorów częstych definiujemy jako piątkę $DMQ = (R, a, \Sigma, \Phi, \beta)$, gdzie R to relacja bazy danych, a jest atrybutem relacji R, Σ jest warunkiem selekcji rekordów relacji R, Φ jest warunkiem selekcji odkrywanych wzorców, a β jest progiem minimalnego wsparcia odkrywanych wzorców. Wynikiem zapytania eksploracyjnego jest zbiór wzorców odkrytych w $\pi_a\sigma_\Sigma R$ spełniających Φ.

Zapytanie eksploracyjne jest realizowane przez algorytm eksploracji danych. Na koszt wykonania tego algorytmu wpływają następujące trzy kluczowe składniki: (1) liczba bloków bazy danych do odczytu, (2) rozmiar bufora zaalokowanego w pamięci operacyjnej, (3) czas CPU. Przykładowo, w algorytmie Apriori odkrywania zbiorów częstych, liczba bloków bazy danych do odczytu jest równa rozmiarowi bazy danych przemnożonemu przez liczbę iteracji algorytmu, rozmiar alokowanego w pamięci operacyjnej bufora jest równy rozmiarowi haszowych drzew kandydatów, a czas CPU jest związany z generowaniem i zliczaniem zbiorów-kandydatów. Poważnym problemem w dziedzinie przetwarzania zapytań eksploracyjnych jest ich wysoki koszt wykonania, zwłaszcza w zakresie kosztu I/O.

Zapytania eksploracyjne są często wykonywane w zbiorach, wsadowo. Zapytania należące do jednego zbioru mogą wykazywać wiele elementów wspólnych, związanych z warunkami selekcji danych i warunkami selekcji wzorców. Integracja wykonania wspólnych operacji występujących w wielu zapytaniach eksploracyjnych może umożliwić redukcję całkowitego kosztu operacji I/O. W celu integracji operacji I/O wielu zapytań eksploracyjnych, obszary buforów dla tych zapytań muszą być wspólnie zaalokowane w jednym obszarze pamięci operacyjnej. Niestety, ze względu na ograniczenie rozmiaru pamięci operacyjnej, może się zdarzyć, iż nie wszystkie zapytania eksploracyjne będą mogły być wykonane współbieżnie. Istnieje zatem potrzeba podziału zbioru zapytań na rozłączne podzbiory zapytań, które mogą i powinny być wykonane wspólnie. Dla każdego z podzbiorów, łączne zapotrzebowanie na pamięć buforową nie może przekraczać dostępnej pamięci operacyjnej. Celem podziału zbioru zapytań eksploracyjnych na podzbiory jest minimalizacja łącznego kosztu I/O, a zatem wskazane jest grupowanie w podzbiorach tych zapytań eksploracyjnych, które współdzielą wiele bloków bazy danych, a separowanie tych, które operują na rozłącznych jej fragmentach. Formalny problem, który adresujemy jest więc następujący. Dany jest zbiór zapytań eksploracyjnych $DMQ = \{dmq_1, dmq_2, ..., dmq_n\}$, gdzie $dmq_i = (R, a, \Sigma_i, \Phi_i, \beta_i)$, Σ_i ma postać "$(l^i_{1min} < a < l^i_{1max}) \vee (l^i_{2min} < a < l^i_{2max}) \vee..\vee (l^i_{kmin} < a < l^i_{kmax})$", $l^i_* \in dom(a)$ oraz istnieją co najmniej dwa zapytania eksploracyjne $dmq_i = (R, a, \Sigma_i, \Phi_i, \beta_i)$ i $dmq_j = (R, a, \Sigma_j, \Phi_j, \beta_j)$ takie, że $\sigma_{\Sigma i}R \cap \sigma_{\Sigma j}R \neq \emptyset$. Problem optymalizacji zbioru zapytań eksploracyjnych DMQ polega na takim podziale tego zbioru na rozłączne podzbiory, aby: (1) w ramach dostępnej pamięci operacyjnej zostały zaspokojone zapotrzebowania zapytań na pamięć buforową, (2) całkowity koszt I/O wykonania DMQ był minimalny.

2.2. Model współdzielenia danych przez wiele zapytań eksploracyjnych

Zbiór zapytań eksploracyjnych reprezentujemy w postaci grafu posiadającego dwa rodzaje wierzchołków: (1) wierzchołki zapytań eksploracyjnych o (2) wierzchołki rozłącznych formuł selekcji. Krawędzie grafu łączą zapytania eksploracyjne z rozłącznymi formułami selekcji, na których te zapytania operują. Graf taki będziemy nazywać grafem współdzielenia danych. Z formalnego punktu widzenia, graf współdzielenia danych jest zdefiniowany następująco. Niech $S = \{s_1 , s_2 ,..., s_k\}$ oznacza zbiór rozłącznych formuł selekcji dla *DMQ*, tzn. zbiór takich warunków selekcji opartych na atrybucie *a* relacji *R*, że dla każdego *i*,*j* mamy $\sigma_{si}R \cap \sigma_{sj}R = \emptyset$ oraz dla każdego *i* istnieją takie liczby naturalne *a*, *b*, ..., *m*, że $\sigma_{\Sigma i}R = \sigma_{sa}R \cup \sigma_{sb}R \cup ... \cup \sigma_{sm}R$. Graf *DSG* = (*V*,*E*) nazywamy grafem współdzielenia danych dla zbioru zapytań eksploracyjnych *DMQ* wtedy i tylko wtedy gdy $V = DMQ \cup S$, $E = \{(dmq_i,s_j) \mid dmq_j \in DMQ, s_j \in S, \sigma_{\Sigma i}R \cap \sigma_{sj}R \neq \emptyset\}$.



Rys. 1. Przykładowy graf współdzielenia danych dla zbioru zapytań eksploracyjnych

Rozważmy następujący przykładowy graf współdzielenia danych. Dana jest relacja R_1 = ($attr_1$, $attr_2$) oraz trzy zapytania eksploracyjne: dmq_1 = (R_1, "$attr_2$", "5 < $attr_1$ < 20", ∅, 3), dmq_2 = (R_1, "$attr_2$", "10 < $attr_1$ < 30", ∅, 5), dmq_3 = (R_1, "$attr_2$", "15 < $attr_1$ < 40", ∅, 4). Zbiór rozłącznych formuł selekcji będzie zatem następujący: $S = \{s_1$ = "5 < $attr_1$ < 10", s_2 = "10 < $attr_1$ < 15", s_3 = "15 < $attr_1$ < 20", s_4 = "20 < $attr_1$ < 30", s_5 = "30 < $attr_1$ < 40"}. Graf współdzielenia danych dla {dmq_1, dmq_2, dmq_3} został przedstawiony na rys. 1. Elipsy reprezentują zapytania eksploracyjne, a prostokąty – rozłączne formuły selekcji.

3. Optymalizacja zbioru zapytań eksploracyjnych jako problem partycjonowania grafu

3.1. Partycjonowanie grafów

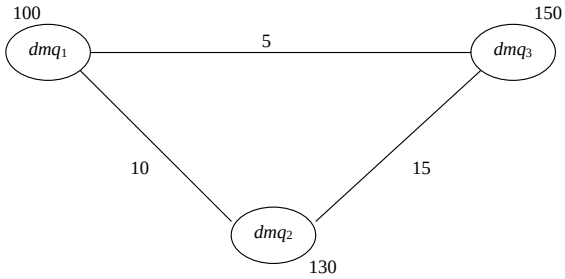
Partycjonowanie grafu jest interesującym problemem naukowym, posiadającym szereg odniesień do dziedzin: optymalizacji, częściowych równań różniczkowych, dużych systemów równań liniowych, projektowania układów VLSI, podziału zadań w przetwarzaniu równoległym, projektowania wspomaganego komputerowo, itp. Podstawowa definicja problemu partycjonowania grafu przedstawia się następująco. Dany jest graf $G=(V,E)$, gdzie V jest zbiorem wierzchołków, a E jest zbiorem krawędzi łączących wierzchołki. Zarówno wierzchołki, jak i krawędzie są opisane wagami; $|v|$ oznacza wagę wierzchołka $v \in V$, a $|e|$ oznacza wagę krawędzi $e \in E$. Klasyczny problem partycjonowania grafu polega na podziale G na k rozłącznych partycji w taki sposób, aby suma wag wierzchołków w każdej partycji była w przybliżeniu taka sama, a suma wag krawędzi łączących wierzchołki należące do różnych partycji była jak najmniejsza.

Problem partycjonowania grafu jest traktowany obliczeniowo jako problem NP-trudny [6], w związku z czym stał się inspiracją dla wielu heurystyk dostarczających rozwiązań suboptymalnych [4][8][12]. Najbardziej obiecującą i efektywną grupę heurystyk stanowią wielopoziomowe algorytmy partycjonowania grafu, składające się z czterech faz: coarsening, initial partitioning, uncoarsening i refining. Celem fazy coarsening jest redukcja rozmiaru grafu przy jednoczesnym zachowaniu tych jego własności, które są kluczowe dla dobrego partycjonowania. W fazie initial partitioning, zredukowany graf jest dzielony na przybliżone partycje. W fazie uncoarsening, zredukowane wierzchołki grafu są rozbijane na wierzchołki oryginalne. W każdym kroku fazy uncoarsening, granice partycji podlegają korekcie służącej podniesieniu jakości partycjonowania.

3.2. Graf uzysku

Na potrzeby transformacji problemu optymalizacji zbioru zapytań eksploracyjnych do problemu partycjonowania grafu wprowadzimy kolejną strukturę grafową nazywaną grafem uzysku. W grafie uzysku wierzchołki reprezentują zapytania eksploracyjne, a krawędzie – oszczędność na kosztach I/O, jaka będzie uzyskana w przypadku, gdy zapytania eksploracyjne połączone krawędzią zostaną wykonane współbieżnie (oszczędność wynikające z jednokrotnego wspólnego odczytu fragmentu bazy danych zamiast wykonania dwóch niezależnych odczytów). W przypadku, gdy wspólne wykonanie dwóch zapytań eksploracyjnych nie umożliwia redukcji kosztów I/O, krawędź pomiędzy tymi zapytaniami nie istnieje. Wierzchołki są opisane wagami, oznaczającymi rozmiar wymaganej pamięci buforowej. Graf uzysku dla przykładowych zapytań eksploracyjnych z rys. 1 został przedstawiony na rys.2 . Dla przykładu przyjęliśmy, że zapytanie dmq_1 wymaga 100 bloków pamięci, *dmq2 – 130*, a *dmq3* – 150. Ponadto, założyliśmy, że koszty I/O odczytu danych

opisanych rozłącznymi warunkami selekcji s_2, s_3 i s_4 wynoszą odpowiednio: 5 bloków, 5 bloków, 10 bloków. Na podstawie rys. 1 możemy zaobserwować, że np. współbieżne wykonanie zapytań eksploracyjnych dmq_1 i dmq_2 pozwoli zredukować łączny koszt I/O wykonania zbioru zapytań o 10 bloków dyskowych, ponieważ zapytania te współdzielą dwie rozłączne formuły selekcji, każda o koszcie wykonania 5 bloków. W związku z tym, w grafie uzysku zapytania te są połączone krawędzią o wadze 10.



Rys. 2. Przykładowy graf uzysku

3.3. Redefinicja problemu partycjonowania grafu

Korzystając z pojęcia grafu uzysku przystępujemy do transformacji problemu optymalizacji zbioru zapytań eksploracyjnych do następującego problemu partycjonowania grafu. Dany jest graf $G=(V,E)$, gdzie V to zbiór wierzchołków opisanych wagami, a E to zbiór krawędzi opisanych wagami. Każdy wierzchołek $v \in V$ reprezentuje pojedyncze zapytanie eksploracyjne, a związana z nim waga, $|v|$, to rozmiar bufora pamięci operacyjnej wymaganego do wykonania zapytania. Każda krawędź $e \in E$ reprezentuje współdzielenie danych źródłowych przez dwa zapytania eksploracyjne, związana z nią waga $|e|$, to łączna liczba bloków dyskowych współdzielonych przez zapytania eksploracyjne połączone daną krawędzią. Definiowany problem polega na podziale G na rozłączne partycje w taki sposób, aby suma wag wierzchołków w każdej partycji nie przekraczała arbitralnie ustalonego progu (rozmiar systemowej pamięci operacyjnej) a łączna waga wszystkich krawędzi łączących ze sobą wierzchołki należące do różnych partycji była jak najmniejsza.

Nasze sformułowanie problemu partycjonowania grafu różni się od oryginalnego podejścia tym, że nakładamy dodatkowe ograniczenie na łączną wagę wierzchołków w partycji oraz tym, że liczba partycji, które chcemy uzyskać jest nieznana z góry. Takie ograniczenia mogą jednać być zaimplementowane poprzez rozszerzenie wielu znanych heurystycznych algorytmów partycjonowania grafów. W kolejnym podrozdziale przedstawiamy przykładowy algorytm partycjonowania grafu uzysku oparty na znanej metodzie *heavy-edge matching* method [12].

3.4. Przykładowa adaptacja algorytmu partycjonowania grafu

Oryginalny algorytm heavy-edge matching iteracyjne redukuje (ang. coarsening) graf w celu zmniejszenia liczby jego wierzchołków. W każdej iteracji algorytm znajduje taki maksymalny zbiór krawędzi, w którym żadne dwie krawędzie nie współdzielą wierzchołka. Zbiór taki nazywany jest dopasowaniem (ang. matching). Następnie, krawędzie z dopasowania są usuwane, a łączone przez nie wierzchołki są redukowane do jednego wierzchołka, którego waga równa jest sumie wag wierzchołków składowych. Redukcji ulegają również krawędzie, a ich wagi podlegają sumowaniu. W celu wygenerowania dopasowania, oryginalny algorytm heavy-edge matching dokonuje losowego wyboru wierzchołka nieuczestniczącego dotąd w dopasowaniu, następnie wybiera wychodzącą z niego krawędź o największej wadze prowadzącą do innego wierzchołka, który także nie uczestniczy jeszcze w dopasowaniu, by ostatecznie oznaczyć oba wierzchołki jako „matched".

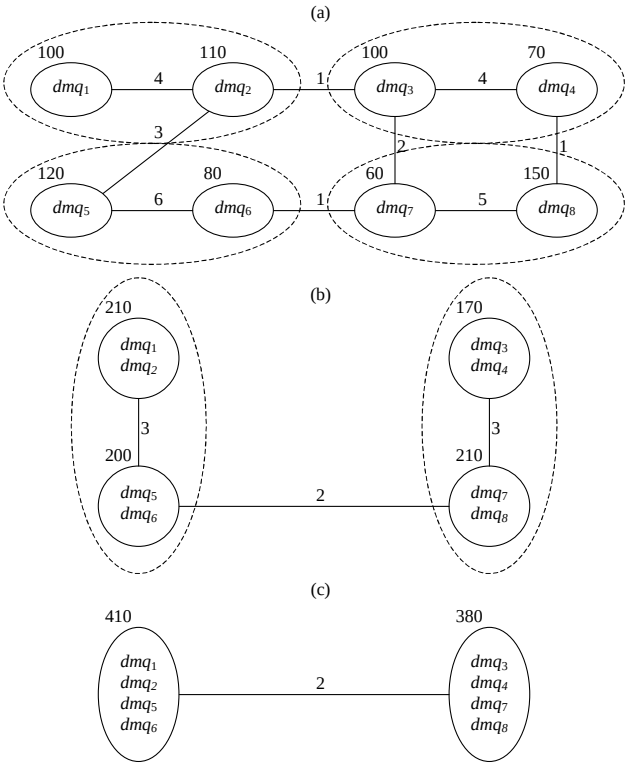
Nasza adaptacja algorytmu heavy-edge matching skupia się na modyfikacji kroku redukcji wierzchołków. Kiedy wybierany jest losowy wierzchołek nieuczestniczący dotąd w dopasowaniu, sortujemy malejąco wychodzące z niego krawędzie, prowadzące do innych wierzchołków nieuczestniczących w dopasowaniu. Następnie wybieramy krawędź o największej wadze i szacujemy oczekiwaną wagę połączonego wierzchołka. Jeżeli waga ta nie przekracza arbitralnie ustalonego limitu, to dokonujemy połączenia wierzchołków, w przeciwnym razie wybieramy kolejną krawędź z posortowanej listy i próbujemy połączyć badany wierzchołek z innym, itd. Jeżeli badany wierzchołek nie może zostać połączony z żadnym innym w ramach ustalonego limitu, to nie dokonujemy żadnego połączenia, ale badany wierzchołek oznaczamy jako „matched". Algorytm kończy pracę w chwili, gdy nie można już dokonać żadnego połączenia wierzchołków. Wynikiem pracy algorytmu jest niewielki graf, którego wierzchołki reprezentują współbieżne wykonania podzbiorów zapytań eksploracyjnych – zapytań opisanych zredukowanymi wierzchołkami.

3.5. Przykład

Rozważmy następujący przykład optymalizacji zbioru zapytań eksploracyjnych. Rys. 3a przedstawia graf uzysku dla zbioru 8 zapytań eksploracyjnych. Większość z tych zapytań współdzieli pewne fragmenty bazy danych, w związku z czym próba współbieżnego ich wykonania jest uzasadniona. Przyjmijmy, że rozmiar pamięci operacyjnej wynosi 500 jednostek. Zapotrzebowanie zapytań na pamięć operacyjną jest zapisane jako wagi wierzchołków grafu uzysku. Jak można zauważyć, wszystkie zapytania eksploracyjne nie zmieszczą się jednocześnie w pamięci operacyjnej, w związku z czym konieczne jest ich partycjonowanie.

Zaadaptowany algorytm heavy-weight matching dokonuje losowego wyboru wierzchołków. Początkowo, żaden z wierzchołków nie należy do dopasowania. Załóżmy, że pierwszym losowo wybranym wierzchołkiem jest dmq_8. Sortujemy krawędzie wychodzące z dmq_8 a prowadzące do innych wierzchołków nieuczestniczących dotąd w dopasowaniu. Najcięższą krawędzią jest krawędź łącząca dmq_8 z dmq_7. Szacujemy oczekiwana wagę połączonego wierzchołka: $60 + 150 = 210 < 500$. Oznacza to, że rozważane zapytania

zmieszczą się w pamięci operacyjnej, w związku z czym połączenie wierzchołków jest akceptowane, a wierzchołki oznaczane są jako „matched”. W podobny sposób następuje redukcja pozostałych wierzchołków, jak pokazano na rys. 3a. Wynikiem pierwszej iteracji jest nowy graf przedstawiony na rys. 3b. Wierzchołki tego grafu nie przedstawiają już pojedynczych zapytań eksploracyjnych lecz grupy zapytań, które powinny zostać wykonane współbieżnie. Zauważmy, że w wyniku połączenia wierzchołków, redukcji uległy również krawędzie grafu. Podczas drugiej iteracji algorytmu, znajdujemy dopasowanie w grafie z rys. 3b. Załóżmy, że losowy wybór wyłonił wierzchołek dmq_5/dmq_6, z którego najcięższa krawędź prowadzi do wierzchołka dmq_1/dmq_2. Połączenie tych wierzchołków jest możliwe, ponieważ waga nowego wierzchołka wyniesie 200 + 210 = 410 < 500, co oznacza, że reprezentowane zapytania nadal zmieszczą się w pamięci operacyjnej. W podobny sposób połączone zostaną wierzchołki dmq_3/dmq_4 i dmq_7/dmq_8. Wynik drugiej iteracji algorytmu przedstawia rys. 3c. W następnej iteracji usiłujemy połączyć wierzchołki grafu z rys. 3c. Niestety połączenie takie nie jest możliwe ponieważ oczekiwana waga połączonego wierzchołka przekracza rozmiar dostępnej pamięci operacyjnej (410 + 380 = 790 > 500). Algorytm kończy pracę.



Rys. 3. Metoda Heavy-Edge Matching zaadaptowana na potrzeby optymalizacji wykonania zbioru zapytań eksploracyjnych

Uzyskaliśmy następujące partycjonowanie naszego zbioru zapytań eksploracyjnych: współbieżnie zostaną wykonane zapytania dmq_1, dmq_2, dmq_5, dmq_6, a następnie zapytania dmq_3, dmq_4, dmq_7, dmq_8. Posługując się grafem uzysku szacujemy, że taki sposób zbioru wykonania zapytań eksploracyjnych pozwoli zredukować łączny koszt I/O o 4 + 3 + 6 + 4 + 2 + 1 + 5 = 25 bloków dyskowych.

4. Podsumowanie

W niniejszej pracy rozważaliśmy problem optymalizacji zbioru zapytań eksploracyjnych polegający na podziale zbioru zapytań na współbieżnie wykonywane podzbiory w taki sposób, aby: zapytania z każdego podzbioru mieściły się równocześnie w pamięci operacyjnej i łączny koszt I/O wykonania wszystkich zapytań był jak najmniejszy. Pokazaliśmy, że problem taki może być traktowany jako zmodyfikowany problem partycjonowania grafu. Ponieważ problem partycjonowania grafu jest problemem NP-trudnym, to skupiliśmy się na heurystycznych rozwiązaniach, które mogą być zaadaptowane na potrzeby rozwiązania problemu optymalizacji zbioru zapytań eksploracyjnych.

LITERATURA

[1] Agrawal R., Imielinski T., Swami A., Mining Association Rules Between Sets of Items in Large Databases, *Proc. of the 1993 ACM SIGMOD Conf. on Management of Data*, 1993.

[2] Alsabbagh J.R., Raghavan V.V., Analysis of common subexpression exploitation models in multiple-query processing, *Proc. of the 10th ICDE Conference*, 1994.

[3] Agrawal R., Srikant R., Fast Algorithms for Mining Association Rules, *Proc. of the 20th Int'l Conf. on Very Large Data Bases*, 1994.

[4] Bui, T., Jones, C., A heuristic for reducing fill in sparse matrix factorization, *Proc. of the 6th SIAM Conference on parallel Processing for Scientific Computing*, 1993.

[5] Ceri S., Meo R., Psaila G., A New SQL-like Operator for Mining Association Rules, *Proc. of the 22nd Int'l Conference on Very Large Data Bases*, 1996.

[6] Garey, M., Johnson, D., Stockmeyer, L., Some simplified NP-complete graph problems, *Theoretical Computer Science*, **1**, 1976, 237-267.

[7] Han J., Fu Y., Wang W., Chiang J., Gong W., Koperski K., Li D., Lu Y., Rajan A., Stefanovic N., Xia B., Zaiane O.R., DBMiner: A System for Mining Knowledge in Large Relational Databases, *Proc. of the 2nd KDD Conference*, 1996.

[8] Hendrickson, B., Leland, R., A multilevel algorithm for partitioning graphs, *Proc. of Supercomputing'95*, 1995.

[9] Imielinski T., Mannila H., A Database Perspective on Knowledge Discovery. *Communications of the ACM*, **39**, 11, 1996, 58-64.

[10] Imielinski T., Virmani A., Abdulghani A., Datamine: Application programming interface and query language for data mining, *Proc. of the 2nd KDD Conference*, 1996.

[11] Jarke M., Common subexpression isolation in multiple query optimisation, Kim W., Reiner D.S. (Eds.), *Query Processing in Database Systems*, Springer, 1985.
[12] Karypis, G., Kumar, V., A fast and high quality multilevel scheme for partitioning irregular graphs, Technical Report TR 95-035, Department of Computing Science, University of Minnesota, 1995.
[13] Morzy T., Wojciechowski M., Zakrzewicz M., Data Mining Support in Database Management Systems, *Proc. of the 2nd DaWaK Conference*, 2000.
[14] Roy P., Seshadri S., Sundarshan S., Bhobe S., Efficient and Extensible Algorithms for Multi Query Optimization, *Proc. ACM SIGMOD Intl. Conference on Management of Data*, 2000.
[15] Sellis T., Multiple query optimization, *ACM Transactions on Database Systems*, **13**, 1, 1988, 23-52.
[16] Sellis T., Ghosh S., On the Multi-Query Optimization Problem, *IEEE Transactions on Knowledge and Data Engineering*, **2**, 2, 1990, 262-266.
[17] Wojciechowski M., Zakrzewicz M., Evaluation of Common Counting Method for Concurrent Data Mining Queries, *Proc. of the 7th ADBIS Conference*, 2003.
[18] Wojciechowski M., Zakrzewicz M., Data Mining Query Scheduling for Apriori Common Counting, *Proc. of the 6th Int'l Baltic Conf. on Databases and Inf. Systems*, 2004.
[19] Wojciechowski, M., Zakrzewicz, M., On multiple data mining optimization, *Proc. of the 2005 PAKDD Conference*, 2005.

MULTIPLE DATA MINING QUERY OPTIMIZATION THROUGH GRAPH PARTITIONING

Multiple data mining query optimization consists in partitioning a set of data mining queries into non-overlapping subsets of concurrently executed queries in such a way that the total I/O cost of the queries is minimized. In this paper we argue that multiple data mining query optimization problem can be transformed into the graph partitioning problem, which was extensively researched in many scientific areas. We reformulate the classical graph partitioning problem and demonstrate how existing heuristic algorithms can be adapted to perform multiple data mining query optimization.