

Leszek Kotulski*, Adam Sędziwy*

Zastosowanie gramatyk grafowych typu *double pushout* w środowisku GRADIS

1. Wprowadzenie

Grafy i ich transformacje stanowią dobrze rozpoznane i często używane narzędzie opisu i specyfikacji struktury i zachowania systemów w wielu dziedzinach, na przykład w syntaktycznych metodach rozpoznawania obrazów, wizualnym modelowaniu, specyfikacji systemów rozproszonych. Istnieje wiele formalizmów opartych o transformacje grafowe (zob. *Handbook of Graph Grammars and Computing by Graph Transformation* [1, 2, 7]). Jeden z najbardziej obiecujących kierunków badań w tym zakresie, związany jest z wykorzystaniem gramatyk typu *double pushout* (DPO) z uwagi na ich moc opisową. Niestety, słabą stroną tego podejścia jest złożoność obliczeniowa.

Z punktu widzenia złożoności obliczeniowej w obszarze naszych zainteresowań znajdują się problemy z klasy problemów NP-zupełnych. Do tej klasy należą takie zagadnienia jak istnienie cyklu Hamiltona w grafie czy k -kolorowalność. Dla problemów tych znane są algorytmy o czasie wielomianowym wykonywane na równoległej maszynie Turinga, natomiast nie znaleziono algorytmów o tej złożoności dla maszyny sekwencyjnej. Kreowski wykazał w swojej pracy [6], że zastosowanie równoległych transformacji grafowych wykonywanych na multizbiorach grafów redukuje złożoność obliczeniową dla problemu Hamiltona do czasu wielomianowego. Formalnie oznacza to, że problem klasy NP jest równy złożonościowo problemowi P_{GPAR} . Wniosek ten stanowi zachętę do opracowywania automatycznych metod zrównoleglenia algorytmów transformacji grafowych.

Środowisko GRADIS (akronim angielskiej nazwy *GRAph DIStribution toolkit*) [3] pozwala na rozproszenie kontrolowanego grafu bez zmiany reguł transformacyjnych gramatyki grafowej, używanej do opisu problemu w wersji scentralizowanej. GRADIS dostarcza zbioru tzw. lokalnych środowisk transformacji grafowych (LGTE) rozproszonych w sieci, przeznaczonych do realizacji transformacji grafowych na lokalnych grafach i kooperujących ze sobą w celu osiągnięcia wspólnego celu. Przyjmujemy, że działania rozproszonego systemu transformacyjnego jest poprawne, jeżeli w dowolnym momencie czasu po konsoli-

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

dacji lokalnych grafów otrzymujemy graf, który da się wywieść z grafu początkowego za pomocą sekwencyjnych transformacji grafowych stosowanych grafie środowisku scentralizowanym.

Zysk z wprowadzenia transformacji równoległych pojawi się jedynie wtedy, gdy udowodnimy, że narzut spowodowany współpracą lokalnych środowisk transformacji grafowych nie będzie istotny. Narzut ten powinien być analizowany zarówno z punktu widzenia komunikacji (mierzony w ilości wymienianych komunikatów) jak i złożoności obliczeniowej stosowanych algorytmów. Niestety, narzut ten jest ściśle powiązany z określonym typem gramatyk grafowych. W niniejszym artykule dla gramatyk DPO udowodnimy (w rozdz. 4) wielomianową złożoność algorytmu koordynacji działań między grafami lokalnymi zarządzanymi przez agentów, realizowanym w rozproszonym równoległym środowisku agentowym oraz poprawność tego rozwiązania w kontekście błędów uwarunkowanych czasowo i zakleszczeń. Wcześniej jednak w rozdziale 2 wprowadzona zostanie definicja DPO oraz podany przykład jej zastosowania, a w rozdziale 3 naszkicowana zostanie koncepcja komplementarnej formy grafów lokalnych i ich reprezentacji w systemie GRADIS.

2. Gramatyki typu *double pushout*

W niniejszym artykule rozważamy grafy etykietowane (atrybutowane). Niech Σ^v i Σ^e oznaczają odpowiednio zbiory etykiet wierzchołkowych i krawędziowych. Struktura grafu zdefiniowana jest następująco.

Definicja 1. (Σ^v, Σ^e) -graf jest trójką postaci (V, E, φ) , gdzie V jest zbiorem niepustym, E jest podzbiorem $V \times \Sigma^e \times V$, natomiast $\varphi: V \rightarrow \Sigma^v$. Rodzinę (Σ^v, Σ^e) -grafów będziemy oznaczać symbolem \mathbf{G} .

Dla każdego $(V, E, \varphi) \in \mathbf{G}$, V jest zbiorem wierzchołków, E jest zbiorem krawędzi, natomiast φ funkcją etykietującą wierzchołki.

Obiektem naszego zainteresowania są gramatyki DPO. Produkcja P tej gramatyki ma następującą postać: $L \leftarrow K \rightarrow R$, $L \supseteq K \subseteq R$, gdzie graf K jest tzw. interfejsem, natomiast L , R są odpowiednio lewą i prawą stroną produkcji; dodatkowo zakładamy, że $L, K, R \in \mathbf{G}$.

Zastosowanie produkcji P do grafu G może być opisane w sposób nieformalny następująco. Najpierw należy znaleźć wystąpienie grafu L w G , a dokładniej morfizmu $m(L)$ w G . Morfizm m musi spełniać tzw. warunek sklejania (*gluing condition*) zbudowany z dwóch poniższych własności:

- 1) $\forall x, y \in V(L) : m(x) = m(y) \Leftrightarrow x = y \vee x, y \in L \cap R$ (warunek identyfikacji – *identification condition*),
- 2) $\forall x \in V(G - m(L)) \quad \forall y \in m(L) - m(K) \forall \mu \in \Sigma^e \Rightarrow (x, \mu, y) \notin E(G) \wedge (y, \mu, x) \notin E(G)$ (warunek zachowania krawędzi – *dangling condition*).

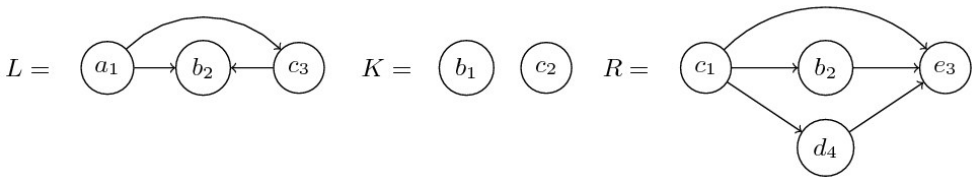
Po znalezieniu wystąpienia $m(L)$ w grafie G usuwamy z G wszystkie węzły należące do $m(L) - m(K)$ razem z incydentnymi do nich krawędziami. Na koniec dodajemy do G

węzły $m(R) - m(K)$ wraz z odpowiednimi krawędziami. W powyższych krokach używamy lokalnej indeksacji wierzchołków w grafach L, K, R .

Należy zauważyć, iż warunek zachowania krawędzi powoduje, jeżeli kasujemy wierzchołek to musimy też usunąć wszystkie krawędzie z nim incydentne. Fakt ten ma wpływ na strategię inkorporacji wierzchołków, realizowana przez agentów w środowisku rozproszonym. Formalna definicja gramatyki DPO znajduje się w [7].

2.1. Przykład

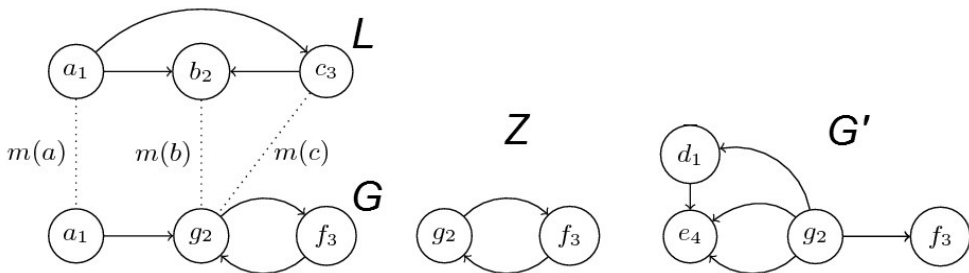
Poniższy przykład ilustruje zastosowanie produkcji gramatyki DPO. Produkcja P zdefiniowana jest poprzez grafy L, K, R z rysunku 1 i zastosowana jest do grafu G z rysunku 2 przy morfizmie m określonym następująco $m(a_1) = a_1, m(b_2) = m(c_3) = g_2$.



Rys. 1. Przykładowa postać produkcji DPO: $L \leftarrow K \rightarrow R$

Operacja ta przebiega w trzech krokach:

1. Znajdowane jest wystąpienie L w G , wg morfizmu m . Na rysunku 2 (po lewej) pokazano morfizm m odwzorowujący węzły L w węzły G .
2. Węzeł $a_1 = m(L) - m(K)$ zostaje usunięty z G , w wyniku czego powstaje tymczasowy graf Z (rys. 2, środek). Należy zauważyć, że zostaje zachowany warunek sklejania.
3. Węzły i odpowiednie krawędzie $m(R) - m(K)$ zostają dodane do Z . Otrzymany w wyniku operacji graf G' pokazano na rysunku 2 (po prawej).



Rys. 2. Zastosowanie produkcji DPO (zob. rys.1) do grafu G . Kropkowane linie oznaczają morfizmy węzłów L w G . Od lewej: graf G i morfizm $m(L)$ w G , tymczasowy graf Z , graf wynikowy G'

3. Grafy częściowe

Kluczowym krokiem prowadzącym do rozproszonych transformacji grafowych DPO jest dekompozycja grafu G opisującego dany system, na zbiór tzw. *grafów częściowych*, G_i , i zarządzanych przez agentów. W następnym kroku grafy częściowe zostają rozproszone do różnych lokalizacji, gdzie są przetwarzane. Transformacje wykonywane na każdym z grafów G_i są nadzorowane przez lokalnych agentów LGTA_{*i*} (*Local Graph Transformations Agent*). W rozdziale 4 opisano schemat współpracy między agentami w środowisku realizującym rozproszone transformacje DPO.

W celu zachowania spójności między grafem scentralizowanym G a jego rozproszoną formą, grafami częściowymi, wprowadza się pojęcie tzw. węzłów brzegowych. Są to zwykłe węzły, które są wspólne dla dwóch lub więcej grafów częściowych; zostają one zreplikowane i umieszczone w odpowiednich G_i . Graficznie oznaczone są one podwójną linią.

Wprowadza się następującą notację: $\text{Border}(G_i)$ oznacza zbiór wszystkich węzłów brzegowych grafu G_i ; $\text{PathS}(G, v, w)$ jest funkcją zwracającą zbiór wszystkich węzłów leżących na dowolnej acyklicznej ścieżce łączącej wierzchołki v i w w grafie G . Przykładowo, dla grafu G przedstawionego na rysunku 4 mamy: $\text{PathS}(G, a, g) = \{a, b, d, e, f, g\}$.

W trakcie dekompozycji grafu G należy zapewnić, że żadna krawędź łącząca dwa wierzchołki nie przecina granicy pomiędzy dwoma grafami częściowymi.

Definicja 2. Zbiór grafów częściowych $G_i = (V_i, E_i, \phi_i)$, dla $i = 1, 2, \dots, k$, jest *komplementarną formą* grafu G , jeśli istnieje zbiór iniektywnych homomorfizmów $s_i: G_i \rightarrow G$ takich, że:

- 1) $\bigcup_{i=1, \dots, k} s_i(G_i) = G$
- 2) $\forall 1 \leq i, j \leq k : s_i(V_i) \cap s_j(V_j) = s_i(\text{Border}(G_i)) \cap s_j(\text{Border}(G_j))$
- 3) $\forall w \in V_i, v \in V_{j \neq i} : \exists p = \text{PathS}(G, w, v) \Rightarrow \exists b \in \text{Border}(G_i) : s_i(b) \in p$
- 4) $\forall 1 \leq i \leq k : [v \in \text{Border}(G_i) \Leftrightarrow (\exists w \in V(G_i) : w \text{ jest sasiadem } v) \vee V(G_i) = \{v\}]$

Grafy częściowe określa się też mianem komplementarnych. Powyższa formalna definicja może być trudna do praktycznego zastosowania w trakcie konstruowania zbioru grafów częściowych. Efektywny algorytm dekompozycji grafu G na dwa grafy komplementarne znajduje się w pracy [3]. Dekompozycja G na dowolną ilość grafów komplementarnych odbywa się poprzez rekurencyjne wywołanie podstawowego algorytmu. Przykładowy zbiór grafów częściowych otrzymanych dla zadanego G pokazano na rysunku 4, przy zachowaniu następujących założeń:

- 1) Wprowadza się następującą konwencję dotyczącą indeksacji wierzchołków w zbiorze grafów częściowych. Indeks wierzchołka składa się z pary liczb.
 - W przypadku węzłów brzegowych pierwsza z nich równa jest -1 , natomiast druga stanowi globalny identyfikator węzła brzegowego. Wszystkie repliki danego węzła brzegowego mają ten sam indeks globalny postaci $(-1, j)$.

- Dla pozostałych wierzchołków (tzw. wewnętrznych) pierwsza liczba jest unikalnym numerem grafu częściowego, druga zaś unikalnym numerem wierzchołka w tym grafie.
- 2) Dla lepszej czytelności zakłada się, że w dalszej części morfizm m jest odwzorowaniem identycznościowym ($\forall v \in V(H) m(v) = v$).

Dla dowolnego wężła brzegowego v należącego do grafu częściowego G_i możliwe jest takie przesunięcie granic między G_i a sąsiednimi grafami częściowymi, że węzeł v staje się węzłem wewnętrznym w G_i (pozostałe repliki v usuwamy), natomiast jego węzły sąsiednie, należące do innych grafów częściowych, stają się węzłami brzegowymi, jeśli już wcześniej takimi nie były, a ich repliki zostają przyłączone do G_i . Operację tę realizuje procedura $\text{Incorporate}(v, i)$. Formalna definicja procedury Incorporate przedstawiona jest w [3]. Na rysunku 5(A) pokazano zbiór grafów częściowych po wywołaniu procedur $\text{Incorporate}((-1, 1), 2)$, $\text{Incorporate}((-1, 2), 2)$ wykonanych przez agenta zarządzającego grafem częściowym numer 2 z rysunku 4.

4. Operacje na grafie rozproszonym

Głównym założeniem środowiska agendowego GRADIS jest stworzenie możliwości zastosowania produkcji zdefiniowanych dla wspomagania modyfikacji grafu scentralizowanego również w stosunku do grafów częściowych. Bez modyfikacji tych produkcji jest to jednak możliwe jedynie wtedy, gdy graf lewej strony produkcji L zawiera się całkowicie w pewnym grafie PG_i . W przypadku, gdy L jedynie częściowo zawiera się w grafie częściowym PG_i , agent zarządzający PG_i negocjuje z innymi agentami, badając, czy nie uda się tak przesunąć granic pomiędzy grafami częściowymi, by nowy graf PG_i' już zawierał graf L . Po ewentualnym zlokalizowaniu brakujących fragmentów grafu L w sąsiednich grafach częściowych, agent stosuje procedurę Incorporate , aby włączyć do G_i wszystkie węzły należące do L . Po zainkorporowaniu wszystkich węzłów należących do L , produkcja P może być wykonana lokalnie.

Szczegółowa dyskusja dotycząca problemów odnajdywania wystąpienia grafu L dla produkcji $P: L \leftarrow K \rightarrow R$ oraz mechanizmu negocjacji między agentami przedstawiona jest w dalszej części.

W niniejszej pracy zakładamy, iż każdy graf częściowy zarządzany jest przez swojego agenta. W szczególności, jeden z tych grafów, oznaczony jako PG_0 jest zarządzany przez agenta A_0 , który ma za zadanie zastosować produkcję gramatyki DPO, $P: L \leftarrow K \rightarrow R$ na PG_0 . W celu realizacji tego zadania agent musi wykonać następujące trzy kroki:

1. znaleźć wystąpienie całości lub fragmentu L w PG_0 ,
2. upewnić się, że jego środowisko lokalne może uzyskać dostęp do L (po ewentualnej inkorporacji odpowiednich węzłów),
3. wykonać produkcję we współpracy z innymi agentami.

Pierwszym krokiem jest znalezienie wystąpienia L w PG_0 lub w PG_0 i innych grafach częściowych. Agent A_0 wykonuje tą operację startując z określonego wężła v należącego do $V(L)$, i przeszukując sąsiedztwo v . Wiadomo, że $L \subseteq B = k - neighborhood(v)$, gdzie k jest średnicą niekierowanego grafu L^* pochodnego do L ; $k \leq |V(L)|$.

Ponieważ wierzchołki zbioru B mogą znajdować się w innych grafach częściowych, zbiór B jest wyznaczany we współpracy z innymi agentami (ozn. A_1, A_2, \dots, A_n). A_0 wysyła żądanie do A_i ($i = 1, 2, \dots, n$) i otrzymuje wymagane dane (w postaci zbiorów B_i , $i = 1, 2, \dots, n$). Należy podkreślić, że te fragmentaryczne dane są ulotne, tzn. już po ich dostarczeniu do A_0 informacja, że w grafie PG_i znajdują się węzły podane w B_i , może być nieaktualna. Zbiór B rekonstruowany jest następująco: $B = \bigcup_i B_i$, gdzie wskaźnik i przebiega po danych zwracanych przez poszczególnych agentów. Podzbiorem B istotnym dla wykonania produkcji P jest $B' = B \cap L$. Kiedy wystąpienie L w B jest znalezione, wówczas agent A_0 może przejść do kolejnego kroku. Ze względu na warunek zachowania krawędzi (*dangling condition*) wierzchołki graniczne należące do B' nie będą usuwane (bo musielibyśmy usunąć też krawędzie, które nie należą do L), a więc nie musimy ich inkorporować.

W drugim kroku agent A_0 musi upewnić się, że graf L jest dostępny dla produkcji P . Zgodnie z semantyką protokołu zatwierdzania dwufazowego (2PC), w pierwszej fazie A_0 wysyła do wszystkich agentów A_i uczestniczących w operacji, żądanie zablokowania węzłów w zbiorach B_i' . Agent A_i blokuje węzły w B_i' (**agreement=yes**) lub odrzuca żądanie w dwóch możliwych przypadkach:

- 1) Niektóre węzły B_i' są już zablokowane (**agreement=noaccess**) lub
- 2) Niektóre węzły B_i' nie istnieją w grafie częściowym PG_i zarządzanym przez agenta A_i (**agreement=nonexist**), tj. zostały one właśnie zainkorporowane przez innego agenta.

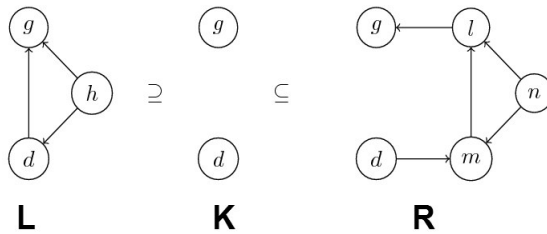
Druga faza zależy od informacji zwrotnej, jaką A_0 otrzymał od pozostałych agentów. Może przebiegać ona według jednego z trzech scenariuszy:

- 1) Jeśli wszystkie odpowiedzi są **agreement=yes**, wówczas agent A_0 wysyła do wszystkich A_i żądanie **commit**. Agenci w odpowiedzi odsyłają do A_0 swoje zbiory B_i' .
- 2) Jeśli chociaż jedną z odpowiedzi jest **agreement=nonexist**, wówczas agent A_0 wysyła żądanie **abort** do wszystkich A_i , a następnie ponawia wyszukiwanie B (B').
- 3) Jeśli chociaż jedną z odpowiedzi jest **agreement=noaccess**, wówczas agent A_0 wysyła żądanie **abort** do wszystkich A_i , a następnie ponawia pierwszą fazę po losowym opóźnieniu.

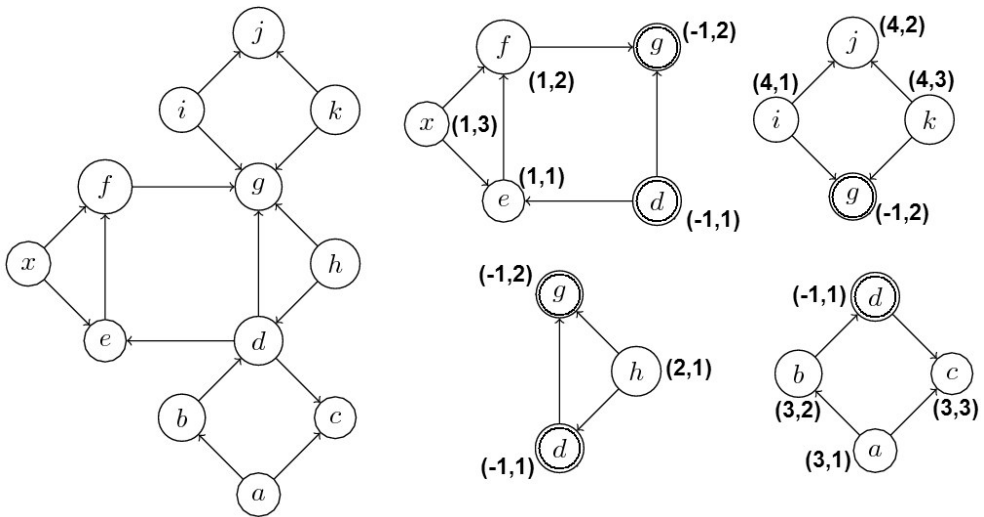
W trzecim kroku, kiedy wszystkie odpowiedzi są **agreement=yes**, agent A_0 prze-suwa granice swojego grafu częściowego poprzez inkorporację poszczególnych B_i' . Należy zauważyć, że wszystkie węzły B_i' stają się węzłami wewnętrznymi w PG_0 . Po tej operacji możliwe już jest lokalne zastosowanie produkcji P w grafie PG_0 .

Przykład

Rozważmy zastosowanie produkcji przedstawionej na rysunku 3, na rozproszonym grafie G, którego komplementarną postać przedstawia rysunku 4. Dla prostoty przyjmujemy, że morfizm m , o którym mowa w definicji gramatyk DPO, jest w tym przypadku odwzorowaniem identycznościowym, a zatem dla dowolnego wężła v mamy: $m(v) = v$.



Rys. 3. Produkcja $P: L \leftarrow K \rightarrow R$, gramatyki typu DPO



Rys. 4. Graf G (po lewej) i jego postać komplementarna. Węzły brzegowe oznaczono podwójną linią. Obok wierzchołków podano indeksację zgodną z konwencją przyjętą dla grafów częściowych

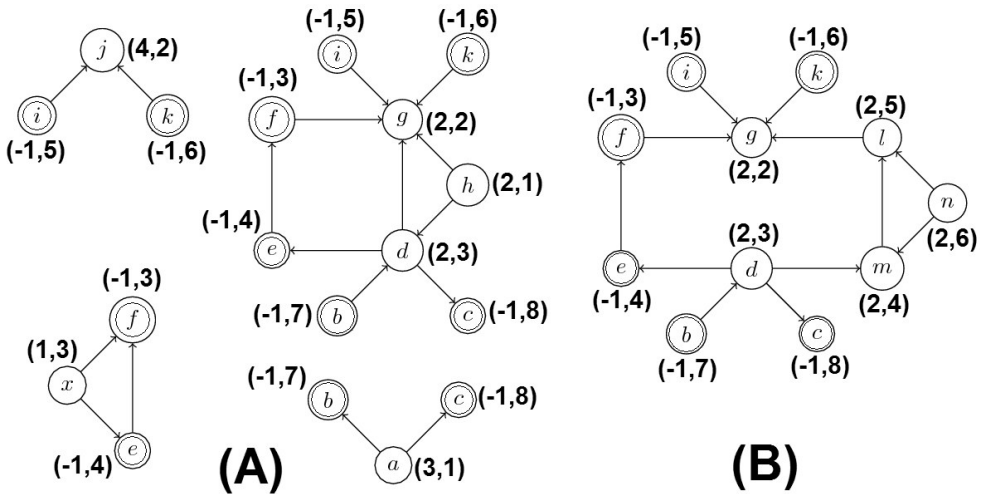
Zakładamy, że agentem inicjującym operację jest agent A_2 , zarządzający grafem częściowym nr 2. Początkowym węzłem poszukiwań grafu L dla agenta A_2 jest wierzchołek (2,1) o etykiecie h. Ponieważ średnica grafu L^* wynosi 2, agent A_2 musi otrzymać do wyłącznej dyspozycji wszystkie węzły zbioru $B = 2 - neighborhood((2,1))$. W tym celu, do odpowiednich agentów sąsiadujących z A_2 wysyłane są żądania udostępnienia brakujących fragmentów B. Tabela 1 przedstawia wykaz węzłów, które dostarczyć muszą do A_2 poszczególni agenci. Zauważyć trzeba, że po zidentyfikowaniu wystąpienia L w B, wystarczy,

że A_2 zainkorporuje jedynie węzły zbiorów $B_i' = L \cap B_i$. Tabela 1 ilustrująca przykład pokazuje, iż daje to znaczącą oszczędność w ilości wywołań operacji Incorporate. Ostatecznie, A_2 musi inkorporować jedynie dwa wierzchołki: $(-1, 1)$, $(-1, 2)$. Są to węzły brzegowe, których repliki znajdują się w grafach częściowych 1, 3, 4. Komplementarną postać G , po tym kroku pokazano na rysunku 5(A). Węzły $(-1, 1)$, $(-1, 2)$ jako węzły wewnętrzne w grafie 2 zmieniają indeksację na odpowiednio $(2, 2)$ i $(2, 3)$. Węzły, które dotychczas były wewnętrznymi w grafach PG_1 , PG_3 , PG_4 , zostają zreplikowane, a ich indeksacja ulega zmianie, zgodnie z przyjętą konwencją. Posiadając wszystkie węzły L (jako węzły wewnętrzne), agent A_2 może przystąpić do lokalnego wykonania produkcji P . Wynikowy graf częściowy 2 przedstawia rysunku 5(B).

Uwaga 1. Dla dostatecznie dużego k , może istnieć graf częściowy PG_i , będący sąsiadem PG_0 , który w całości pokryty jest zbiorem $B = k - neighborhood(v_0)$, czyli $PG_i \subseteq B$. W takim przypadku, w celu obsłużenia żądania wysłanego przez agenta A_0 , A_i zarządzający grafem PG_i musi przesłać dalej (do swoich sąsiadów) żądanie wygenerowane przez A_0 .

Tabela 1
Listy węzłów, inkorporowanych przez A_2

Agent-dawca	B_i	B_i'
A_1	$(-1,1), (-1,2), (1,1), (1,2)$	$(-1,1), (-1,2)$
A_3	$(-1,1), (3,2), (3,3)$	$(-1,1)$
A_4	$(-1,2), (4,1), (4,3)$	$(-1,2)$



Rys. 5. (A) Zbiór grafów częściowych po zainkorporowaniu przez agenta A_2 węzłów potrzebnych do wykonania lokalnego produkcji P (rys. 3). (B) Graf 2 po zastosowaniu P

4.1. Efektywność

Efektywność formalizmu dla transformacji grafowych jest ograniczona w praktycznych zastosowaniach z uwagi na wysoką złożoność czasową dla problemów parsingu i przynależności. Efektywność ta może być poprawiona poprzez przejście do paradygmatu przetwarzania równoległego. Jedną z wydajniejszych metod obejścia problemu wydajności jest zatem automatyczne rozproszenie scentralizowanego grafu opisującego dane zagadnienie i równoległe wykonanie określonych algorytmów na lokalnych podgrafach. Takie podejście dogodne jest dla projektantów systemów, którzy nie muszą interesować się problemami równoległości i niejawnej synchronizacji.

Środowisko GRADIS wspiera takie podejście poprzez wprowadzenie pojęcia grafów komplementarnych i środowiska agentowego realizującego współpracę w trakcie transformacji grafów komplementarnych. Rozważając równoległe transformacje grafowe wiadomo, że możemy za ich pomocą rozwiązywać w czasie wielomianowym problemy NP-zupełne (np. istnienie cyklu Hamiltona w grafie [6]). Z drugiej jednak strony do złożoności należy doliczyć narzut wynikający z komunikacji i synchronizacji w środowisku rozproszonym.

Rozważmy złożoność operacji wyznaczania k -sąsiedztwa, czyli zbioru k -neighborhood(v_0). W pierwszym kroku musimy odwiedzić wszystkich bezpośrednich sąsiadów wężła v_0 , ich zbiór oznaczamy określamy jako *warstwa 1*. Dla każdego wężła należącego do *warstwy j* ($j < k$) odwiedzamy wszystkich jego bezpośrednich sąsiadów nie należących do niższej lub tej samej warstwy. Algorytm zatrzymujemy w sytuacji wyznaczenia *warstwy k* lub odwiedzenia wszystkich wężłów rozważanego grafu. Zauważmy, że algorytm ten eliminuje cykle i nawroty w trakcie odwiedzania wierzchołków, a jego pesymistyczna złożoność wynosi $O(n)$, gdzie n jest liczbą wężłów w grafie scentralizowanym.

Oszacowanie ilości komunikatów potrzebnych do zestawienia informacji z danych rozproszonych jest nieco trudniejsza. W najgorszym przypadku, agent A_0 wysyła komunikaty do wszystkich pozostałych agentów (zakładamy, że informacja dotycząca kilku wężłów brzegowych należących do jednego grafu częściowego może być spakowana w jednej wiadomości); zauważmy, iż ze względu na uwagę 1 może zajść sytuacja, kiedy pewien agent, ozn. A_k , zarządzający grafem zawierającym wężel u , przy czym u jest bezpośrednim sąsiadem v_0 , wyśle informację do swoich sąsiadów będącą duplikatem żądania wysłanego uprzednio przez A_0 . Biorąc pod uwagę powyższe rozumowanie, można określić górne ograniczenie na ilość komunikatów wysłanych przez p agentów: $p(p-1)\dots(p-k+1) \in O(p^k)$. W sytuacji, kiedy wężel v otrzymuje kilka żądań mających na celu przyłączenie go do zbioru B , tylko pierwsze z nich jest realizowane, pozostałe zostają natomiast zignorowane. v może otrzymać nie więcej niż $p-2$ kolejnych żądań od innych agentów (zakładając, że agent pamięta swoją lokalną aktywność i nie wysyła do zewnętrznego wężła dwukrotnie tego samego żądania), lecz nie wywołują one żadnych dodatkowych działań agenta, przez co jego ograniczenie na ilość wysłanych żądań wynosi $n-p$. Rozważamy oba powyższe ograniczenia, gdyż w praktyce pozwalają one na ograniczenie wysłanych komunikatów. Otrzymane grafy B_i są zwracane bezpośrednio do agenta A_0 , co powoduje, że akcje **prepare** i **commit/abort** są realizowane z wysłaniem $O(p)$ komunikatów.

5. Wnioski

Środowisko GRADIS wprowadza reguły dla równoległych transformacji grafowych typu double pushout (przygotowanych dla rozwiązań scentralizowanych) w środowisku rozproszonym, kontrolowanym przez system wieloagentowy. Wprowadzenie równoległości obliczeń zwiększa efektywność systemu pod warunkiem, że narzut związany z komunikacją wynikającą z koordynacji agentów, nie jest zbyt wysoki. Z formalnego punktu widzenia wystarczy pokazać, że złożoność obliczeniowa algorytmu koordynacji jest wielomianowa (choć potęga wielomianu nie powinna być zbyt duża). W praktyce użyteczność rozważanych metod musi być zweryfikowana przez stworzenie aplikacji pracujących w danych obszarach zastosowań. Środowisko GRADIS było badane pod kątem efektywnego wyszukiwania wzorców w problemie rozpoznawania obrazu [4] i projektowania adaptacyjnego [5].

Literatura

- [1] Ehrig H., Engels G., Kreowski H.-J., Rozenberg G., *Handbook of Graph Grammars and Computing by Graph Transformation: Volume II, Application, Languages and Tools*. Ed. World Scientific Publishing Co., River Edge, NJ, 1999.
- [2] Ehrig H., Kreowski H.-J., Montanari U., Rozenberg G., *Handbook of Graph Grammars and Computing by Graph Transformation: Volume III, Concurrency, Parallelism and Distribution*. Ed. World Scientific Publishing Co., River Edge, NJ, 1999.
- [3] Kotulski L., *Distributed Graphs Transformed by Multiagent System*. Artificial Intelligence and Soft Computing ICAISC 2008, LNAI 5097, 1234–1242.
- [4] Kotulski L., *On Efficient Finding Subpatterns with Help ETPL(k) Graph Grammar* – in preparation, 2009.
- [5] Kotulski L., Strug B., *Parallel Graph Transformation in Adaptive Design*. Second International Workshop Graph Computation Models, Leicester 2008, 43–50.
- [6] Kreowski H.J., Kluske S., *Graph Multiset Transformation as a Framework for Massie Parallel Computation*. 4-th International Conference ICGT 2008, LNCS 5214, 351–365.
- [7] Rozenberg G., *Handbook of Graph Grammars and Computing by Graph Transformation: Volume I, Foundations*. Ed. World Scientific Publishing Co., River Edge, NJ, 1997.