Akademia Górniczo-Hutnicza
Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Rozprawa doktorska

Kontrola alokacji zasobów podczas drążenia danych dla potrzeb dynamicznych analiz finansowych

mgr Marcin Tusiewicz

promotor:
dr hab. Leszek Kotulski, prof. AGH

Kraków 2011

W miejscu tym chciałbym podziękować Panu Profesorowi Leszkowi Kotulskiemu

za opiekę oraz nieocenioną pomoc w realizacji niniejszej pracy

Marcin Tusiewicz

Spis treści

Wykaz stosowanych symboli, oznaczeń i skrótów

$\mathbb{N}$	Zbiór liczb naturalnych
$\mathbb{R}$	Zbiór liczb rzeczywistych
$\mathbf{A \rightarrow B}$	Krawędź skierowana o wierzchołku początkowym A i końcowym B
$\mathbf{A \rightarrow B \rightarrow C}$	$A \rightarrow B \wedge B \rightarrow C$
$\mathbf{A \rightarrow B \leftarrow C}$	$A \rightarrow B \wedge C \rightarrow B$
V(G)	Zbiór wierzchołków grafu G
E(G)	Zbiór krawędzi grafu G
#V(G)	Moc zbioru wierzchołków grafu G
#E(G)	Moc zbioru krawędzi grafu G
$\mathbf{PC(T_A)}$	wartość atrybutu PC wierzchołka T_A
$\mathbf{T_A.PC}$	wartość atrybutu PC wierzchołka T_A
$\mathbf{p: L \rightarrow R}$	produkcja Single Push-Out (L / R – lewa / prawa strona produkcji)
DFA	Dynamic Financial Analysis
SPO	Single Push-Out

Znaczenie etykiet wierzchołków grafu DFA-Static / DFA-Snapshot

IS	stan początkowy	(ang. Initial State)
FS	stan końcowy	(ang. Final State)
I	dane wejściowe	(ang. Input Data)
O	dane wyjściowe	(ang. Output Data)
T	zadanie	(ang. Task)
S	selekcja	(ang. Selection)
START	start obliczeń	
STOP	zakończenie obliczeń	
P	proces pasywny	(ang. Passive Process)
A	proces aktywny	(ang. Active Process)
F	proces zakończony	(ang. Finalized Process)
D	dane fizyczne	(ang. Data)
DP	zalążek danych	(ang. Data Placeholder)
AD	dane archiwalne	(ang. Archived Data)
N	węzeł obliczeniowy	(ang. Processing Node)

Podstawowe atrybuty wierzchołków grafu DFA-Static / DFA-Snapshot oraz jednostki miary

PC	Processor Consumption	operacja
MC	Memory Consumption	B
ME	Memory Efficiency	-
DS	Data Size	B
TTC	Time to Consume	operacja
BTR	Blocked Transformation Rule	-
CP	Computing Power	operacja/s
AM	Available Memory	B
NB	Network Bandwidth	B/s

1. Wstęp

1.1. Tematyka pracy

Szacowanie ryzyka w świecie instrumentów pochodnych [1] jest zadaniem niezmiernie skomplikowanym, o czym świadczyć może nadchodząca kolejna fala światowego kryzysu finansowego, przez szereg ekonomistów określanego nie jako „kryzys", lecz jako „skutki". Znany inwestor – Warren Buffet – określił derywaty jako „broń masowego rażenia", a o ich skali udziału w rynku może świadczyć szacowana sumaryczna wartość [2] ponad dziesięciokrotnie przewyższająca światowe bogactwo [3] i podlegająca stałemu wzrostowi.

Jednym z segmentów rynku finansów jest sektor ubezpieczeń. Przez długi okres, sięgający lat dziewięćdziesiątych, cechował się produktami stosunkowo prostymi, obejmującymi jedynie bardzo wąski obszar ryzyka i nie wymagającymi wyrafinowanego aparatu analiz. Ograniczenia prawne zostały jednak zniesione, co umożliwiło wprowadzenie produktów o dużo bardziej skomplikowanej strukturze, na które wpływ ma cały szereg czynników finansowych [4] [5]. W celu określenia wysokości zysku w stosunku do ryzyka, powstała nowa dyscyplina ekonomiczna nazwana Enterprise Risk Management [6].

Z powodu wzrostu skomplikowania problemów, używane wcześniej metody szacowania ryzyka stały się w praktyce nieefektywne [7]. Jednocześnie pojawiły się wydajne systemy komputerowe, oferujące coraz większe możliwości obliczeniowe. W ten sposób narodził się nowy sposób tworzenia modeli, bazujący na metodach symulacyjnych, nazwany Dynamiczne Analizy Finansowe (ang. DFA – Dynamic Financial Analysis) [8] [9] [10] [11]. Głównym celem DFA nie jest podanie rozwiązania optymalnego, lecz badanie relacji zysku do ryzyka, przy założonych parametrach oraz strategiach biznesowych.

Istnieje wiele praktyk związanych z implementacją procesów DFA, jednakże wspólną cechą jest generowanie dużej liczby scenariuszy przedstawiających różne warianty rzeczywistości, opartych na rzeczywistych czynnikach ekonomicznych oraz poczynionych założeniach. Każdy ze scenariuszy służy następnie jako parametr wejściowy przyjętego modelu przedsiębiorstwa, pozwalając na testowanie parametrów mających wpływ na jego funkcjonowanie [12] [13] [14].

Szeroko zakrojone badania autora nad omawianą tematyką, połączone z pracą zawodową w branży aktuarialnej, pozwoliły na systematyzację problemów pojawiających się podczas implementacji modeli Dynamicznych Analiz Finansowych. Struktura nawet stosunkowo prostych modeli DFA jest bardzo skomplikowana, a spotykane implementacje obejmują zarówno wykorzystanie powszechnie używanych arkuszy kalkulacyjnych, jak również dedykowanego oprogramowania oraz sprzętu. Sekwencyjny czas wykonania procesów DFA jest bardzo zróżnicowany i wynosi od kilku minut do kilkunastu miesięcy, niemniej jednak w każdym z przypadków istnieje silna presja środowiska biznesowego na jego obniżenie. Procesy charakteryzują się zarówno wysokimi wymaganiami pamięciowymi z powodu dużej ilości przetwarzanych danych, jak również obliczeniowymi ze względu na stopień złożoności algorytmów. Niespotykanymi dotąd czynnikami, odróżniającym Dynamiczne Analizy Finansowe od klasycznego drążenia danych [15], są nieprzewidywalność sekwencji uruchomienia podzadań oraz „eksplozja danych" – obydwa

podyktowane stochastycznym charakterem obliczeń, w tym generowaniem pokaźnych wielkości zbiorów parametrów wejściowych.

Jednocześnie zadania DFA nie posiadają wspólnej, ścisłej notacji, pozwalającej ująć całość problemu w sposób jednolity i spójny. Poszczególne zagadnienia opisywane są w sposób fragmentaryczny w języku naturalnym, bardzo często z udziałem różnego typu diagramów UML [16]. Z tego też względu nie istnieje formalizm pozwalający na koherentny opis przebiegu ewaluacji w czasie oraz brak jest mechanizmów kontroli sterujących wykonaniem procesów. Niemożliwość automatyzacji całości procesu obliczeniowego pociąga za sobą daleko idące konsekwencje, powodujące wydłużenie czasu zakończenia obliczeń. Nie istnieje także możliwość użycia technik globalnej optymalizacji szeregowania zadań w celu minimalizacji czasu zakończenia obliczeń.

Jako potencjalną bazę do zamodelowania wyżej opisanych problemów Dynamicznych Analiz Finansowych obrano formalizm grafów oraz transformacji grafowych, co motywowane było wymienionymi poniżej przesłankami.

W pracy [17] wykazano, że grafy atrybutowane są intuicyjnym mechanizmem pozwalającym na opis alokacji procesów systemu rozproszonego, natomiast w [18] [19] [20] przedstawiono transformacje grafowe jako formalizm pozwalający na charakterystykę dynamiki zmian takiego systemu. Możliwa jest także równoległa, zdecentralizowana aplikacja produkcji grafowych, zachowująca spójność grafu [21] [22]. W pracach [22] [23] [24] wprowadzono natomiast definicję grafowych struktur hierarchicznych, pozwalających na uzyskanie efektu enkapsulacji.

1.2. Cel badań

Celem podjętych **badań** było opracowanie metody opisu problemów Dynamicznych Analiz Finansowych, umożliwiającej automatyzację alokacji zadań w środowisku rozproszonym oraz wprowadzenie metod zwiększających wydajność obliczeń.

Do najważniejszych **celów pracy** należało:

- wprowadzenie formalizmu umożliwiającego w ścisły oraz jednolity sposób zdefiniowanie zadań DFA, bazującego na definicji grafów atrybutowanych,
- opisanie za pomocą wprowadzonego formalizmu wraz ze szczegółową charakterystyką problemów pojawiających się w Dynamicznych Analizach Finansowych, zaobserwowanych przez autora na etapie prowadzenia prac badawczych,
- wprowadzenie formalizmu umożliwiającego opis zachowania zadań obliczeniowych DFA w środowisku rozproszonym oraz zautomatyzowaną kontrolę procesów przetwarzania opartych na modelu statycznym za pomocą transformacji grafowych,
- użycie wprowadzonego formalizmu do charakterystyki oraz dowodu poprawności metod optymalizacji alokacji zadań dedykowanych dla procesów DFA, opracowanych przez autora.

W rozprawie sformułowano następującą tezę:

Procesy Dynamicznych Analiz Finansowych można formalnie zaspecyfikować za pomocą transformacji grafowych w sposób, który pozwoli na optymalizację alokacji zadań w środowisku rozproszonym.

1.3. Organizacja rozprawy

W Rozdziale 1 dokonano wprowadzenia do tematyki niniejszej pracy.

Rozdział 2 zawiera charakterystykę Dynamicznych Analiz Finansowych, ze szczegółową analizą metod symulacyjnych, uwzględniającą niespotykane dotychczas schematy zachowań procesów obliczeniowych. Do wyjaśnienia zachodzących zjawisk i najczęściej pojawiających się problemów użyto diagramów języka UML oraz opisu w języku naturalnym. Wskazano wymagania, jakie powinien spełniać mechanizm opisu i kontroli zadań DFA, umożliwiający automatyzację procesu obliczeniowego. Stwierdzono, że nie został dotychczas ściśle zdefiniowany taki formalizm.

W Rozdziale 3 wprowadzono model statyczny DFA-Static, bazujący na grafach atrybutowanych. Wykazano, że umożliwia on automatyzację wykonania procesów, w spójny oraz jednolity sposób reprezentując klasę problemu DFA.

W Rozdziale 4, za pomocą zdefiniowanego wcześniej modelu DFA-Static, przedstawione zostały zagadnienia zasygnalizowane w Rozdziale 2. Problemy opisywane wcześniej za pomocą języka naturalnego oraz diagramów UML przedstawione zostały za pomocą aparatu matematycznego. Przedyskutowane zostały także parametry charakteryzujące właściwości modelu, mające wpływ na wymagania środowiska wykonania oraz związaną z nimi szybkość przebiegu obliczeń.

Rozdział 5 definiuje Dynamiczny Model DFA będący rozszerzeniem modelu statycznego, umożliwiający rozproszoną kontrolę alokacji zasobów przy pomocy transformacji grafowych, do których dodany został czynnik czasowy. Omówione zostało sterowanie procesem obliczeniowym oraz zawarty został przykład ilustrujący opracowane rozwiązanie.

Rozdział 6 zawiera autorskie rozwiązania poprawiające wydajność obliczeń, dedykowane dla Dynamicznych Analiz Finansowych, będące rezultatem badań autora nad systemami tego typu.

W Rozdziale 7 zawarte zostały wyniki badań eksperymentalnych, dotyczące zagadnień teoretycznych pojawiających się w pracy.

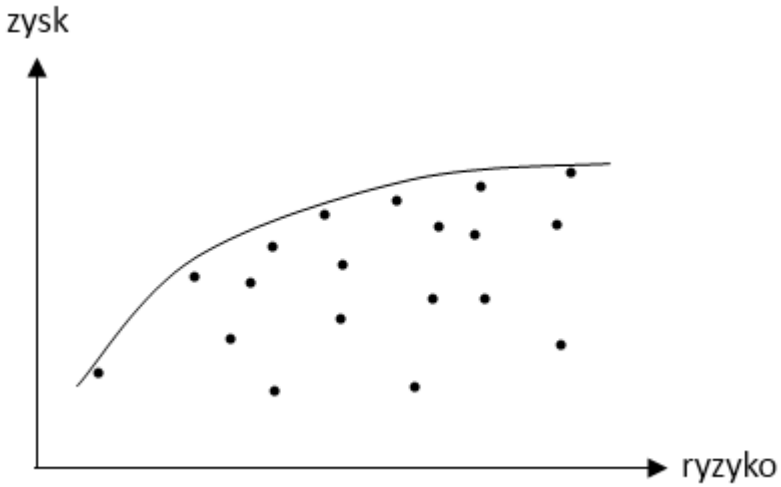
W Rozdziale 8 sformułowano wnioski końcowe oraz podano kierunki dalszych badań.

2. Dynamiczne Analizy Finansowe w ekonomii

2.1. Geneza oraz zarys metod symulacyjnych

Przez dość długi okres – aż do końca lat dziewięćdziesiątych ubiegłego wieku – rynek ubezpieczeń charakteryzował się niewielką elastycznością strategii oraz innowacyjnością. Przepisy prawne w dużym stopniu ograniczały rodzaj ubezpieczeń, co sprawiało, że oferowane produkty cechowały się prostotą i pokrywały tylko wybrany, wąski rodzaj ryzyka. Taki stan rzeczy nie wymagał wyrafinowanego aparatu analiz, który obejmował tylko wyizolowane przypadki, nie uwzględniając globalnych zależności.

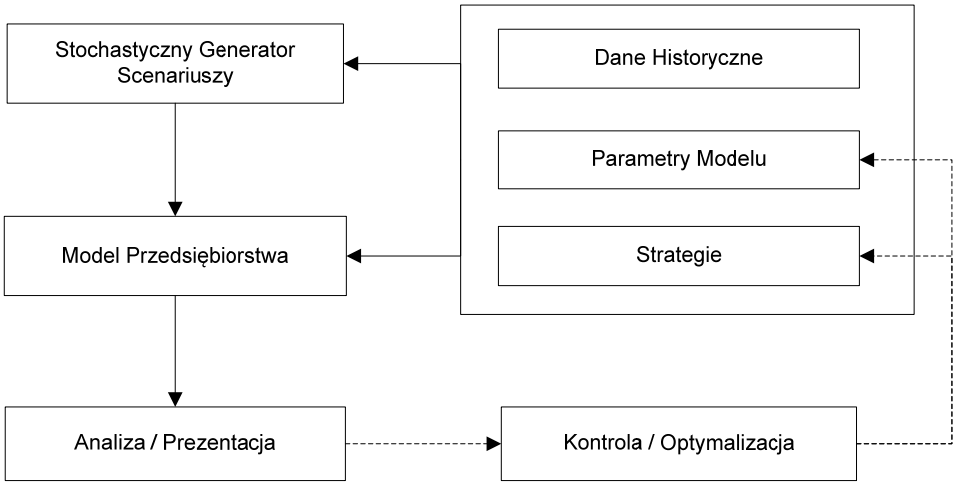
Ograniczenia prawne zostały jednak zniesione, otwierając rynek na nowe, o wiele bardziej skomplikowane produkty. Ujawniły się także nieznane dotąd czynniki cechujące ryzyko, odzwierciedlające zmiany społeczne, demograficzne, a nawet polityczne. Jednocześnie wzmogła się konkurencja oraz presja akcjonariuszy na zwiększenie zysków [25]. W rezultacie tych zmian ubezpieczyciele ustawowo zmuszeni zostali do rozważenia szerokiej gamy czynników mających wpływ na wypłacalność korporacji [4] [5]. W ten sposób zrodziła się nowa dyscyplina ekonomiczna, nazwana Enterprise Risk Management [6].



Rysunek 1. Granica Efektywności.

Tradycyjne analityczne metody szacowania ryzyka okazały się nieskuteczne z powodu dużego skomplikowania problemów, które musiały uwzględnić ogromną ilość możliwych scenariuszy [6] [7] [26] [27] [8] [28]. Wraz z pojawieniem się wydajnych komputerów osobistych oferujących ogromne możliwości obliczeniowe narodził się nowy sposób tworzenia modeli: Dynamiczne Analizy Finansowe (ang. Dynamic Financial Analysis – DFA). DFA, zastosowane do modelowania przepływu środków pieniężnych pomiędzy procesami ekonomicznymi i opisujące zachowanie korporacji ubezpieczeniowych, wykorzystują metody symulacyjne. Dwa główne (a zarazem przeciwstawne) cele DFA to maksymalizacja zysku udziałowców firmy oraz zapewnienie klientom usług o jak najwyższym standardzie (co oczywiście związane jest z kosztami). Operując w obrębie tych dwóch wytycznych,

DFA pomagają uzasadnić kluczowe decyzje dotyczące alokacji kapitału i aktywów, miary wydajności, ustalania cen, cech produktu oraz innych czynników poprzez oszacowanie ryzyka dla zadanej strategii [7] [9] [10] [11] [29]. DFA nie jest jednak tylko metodą, która zwraca optymalną strategię, lecz głównie narzędziem pozwalającym na porównanie różnych strategii w kontekście zysku i ryzyka (ang. return and risk) [30] [31] [32]. Jedną z najpopularniejszych oraz szeroko wykorzystywanych koncepcji jest Granica Efektywności (ang. Efficient Frontier), przedstawiona na Rysunku 1, będąca zbiorem portfeli efektywnych, charakteryzujących się maksymalną oczekiwaną stopą zwrotu wśród portfeli o tym samym ryzyku oraz minimalnym poziomem ryzyka wśród portfeli o takiej samej oczekiwanej stopie zwrotu [33] [34].



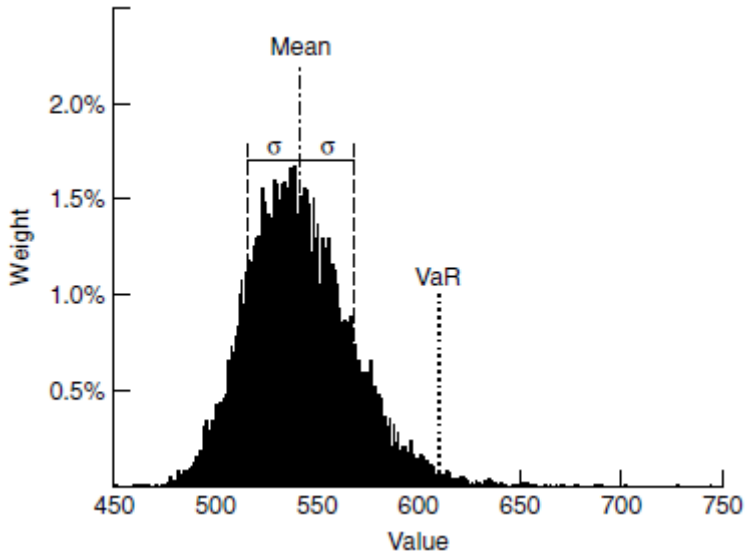
Rysunek 2. Ogólny schemat modelu DFA.

Pomimo faktu istnienia wielu praktyk związanych z implementacją procesów DFA, ogólny schemat modelu został przedstawiony na Rysunku 2, a jego części składowe oraz najczęściej występujące problemy opisane zostaną poniżej.

Stochastyczny Generator Scenariuszy (ang. Stochastic Scenario Generator) składa się z modeli reprezentujących wszystkie czynniki związane z ryzykiem i mające wpływ na przedsiębiorstwo (ang. Company). Należą do nich czynniki ekonomiczne (np. inflacja, kursy wymiany walut), zobowiązania (np. wynikające z zawartych umów ubezpieczeń), aktywa (np. stopy zwrotu akcji) oraz inne, z których wiele związanych jest bezpośrednio z rozpatrywanym typem przedsiębiorstwa (np. katastrofy naturalne w przypadku ubezpieczeń lokali użytkowych). Stochastyczny Generator Scenariuszy tworzy w sposób losowy dużą ilość scenariuszy reprezentujących potencjalne stany rozpatrywanego systemu, opierając się zarówno na danych historycznych, jak i na modelach oraz zależnościach ustalanych przez specjalistów [12] [35] [9] [36] [14].

Każdy wygenerowany scenariusz jest parametrem wejściowym modelu Przedsiębiorstwa, który reprezentuje zachowanie tego Przedsiębiorstwa w postaci scenariusza pochodnego. Badanym wynikiem nie jest jednak pojedyncza odpowiedź dotycząca jednego elementu wejściowego, lecz zbiór (a właściwie multizbiór) scenariuszy pochodnych. Dodatkowo model Przedsiębiorstwa modyfikowany jest poprzez Strategie i Parametry, służące zarówno do kalibracji na etapie

jego tworzenia oraz testowania, jak i badania wpływu decyzji kierowniczych w fazie produkcyjnej [7] [32].



Rysunek 3. Przykładowa gęstość prawdopodobieństwa badanej zmiennej (źródło: [8]).

Etap Analiz / Prezentacji rozpoczyna się kwalifikowaniem otrzymanych wyników według kryteriów, które podlegają badaniom. Następnie, najczęściej stosowanym podejściem jest użycie technik analizy statystycznej, w celu porównania zadanych strategii oraz uzyskania odpowiedzi na postawione pytania. Dla zobrazowania załóżmy, że badaną zmienną Y jest wartość akcji Przedsiębiorstwa. Wtedy symulacja DFA dostarczy nam multizbiór wartości y_1, …, y_N (gdzie N jest zazwyczaj duże). W celu analiz często używany jest rozkład empiryczny, którego dystrybuanta ma postać:

$$\mathrm{F_Y}(y) = \frac{1}{N}\sum_{k=1}^{N} 1(\mathrm{y_k} \leq \mathrm{y}), \tag{2.1}$$

a przykładowy wykres gęstości prawdopodobieństwa przedstawiony został na Rysunku 3. Aby porównać otrzymane rezultaty i podjąć strategiczne decyzje pomocne są pewne kluczowe wskaźniki, takie jak wartość średnia oraz miara ryzyka, które można uzyskać badając odpowiednio wartość oczekiwaną oraz odchylenie standardowe (możliwy jest oczywiście wybór innych kryteriów). Zarząd wyższego szczebla może być zainteresowany także innymi miarami ryzyka, jak na przykład Wartość Zagrożona (VaR – Value-at-Risk), charakteryzująca maksymalne straty i pozwalająca oszacować wymagane rezerwy kapitałowe do ich pokrycia w celu uniknięcia bankructwa [13] [8] [37] [38].

Struktura systemów informatycznych opartych nawet na stosunkowo prostym modelu DFA jest bardzo skomplikowana. W praktyce spotkać można wiele implementacji: najprostsze (chociaż często nie najprymitywniejsze) mają postać arkusza kalkulacyjnego (na przykład programu Microsoft Excel) i czas wykonania mierzony w sekundach lub minutach; te najbardziej zaawansowane wymagają dedykowanego oprogramowania oraz sprzętu, a sekwencyjny czas obliczeń estymowany

jest w miesiącach [39]. W każdym z przypadków istnieje silna oraz uzasadniona presja użytkowników na ograniczenie tego czasu. Zauważmy, że w przypadku systemów pracujących on-line, oczekiwany czas reakcji wynosi kilka sekund, lub co najwyżej minut. Dla „dużych" zadań, oczekiwanie na rezultaty parę lub parenaście miesięcy jest nie do zaakceptowania, gdyż w momencie uzyskania wyników, założenia lub sytuacja ekonomiczna mogą ulec całkowitej zmianie. Natomiast gdy czas wykonania mieści się w akceptowalnych granicach, ogromną korzyścią byłaby możliwość zwiększenia ilości symulacji, a przez to zawężenia przedziału ufności lub polepszenia precyzji algorytmów optymalizacyjnych. W każdym z wymienionych przypadków zrównoleglenie, a w szczególności dystrybucja, są rozważane jako metody przyspieszające uzyskanie wyników. W związku z tym pojawia się szereg problemów, z których dwa najpoważniejsze to ogromna ilość przetwarzanych danych [40] oraz dodatkowo niespotykany i nieanalizowany w dotychczasowych modelach obliczeniowych czynnik: dynamika połączona z nieprzewidywalnością zmian zachowania podprocesów w czasie.

2.2. Nieprzewidywalność obliczeń opartych na modelu DFA

„Nieprzewidywalność obliczeń" jest nową, niespotykaną dotychczas cechą programu złożonego z podprocesów oraz występującego między nimi przepływu danych. Polega ona na niemożliwości dokładnego określenia sekwencji uruchomienia oraz czasu wykonania podprocesów, a także wielkości przetwarzanych i przesyłanych danych, aż do pewnego momentu w trakcie trwania obliczeń. Cecha ta została bardzo wyraźnie zaobserwowana podczas przetwarzania modeli DFA. Eliminuje ona możliwość zastosowania statycznych metod alokacji; jedynym rozwiązaniem pozostają metody dynamiczne, wymagające wyrafinowanego mechanizmu pozwalającego na opisanie oraz kontrolę rozważanego systemu.

Poniżej zostały omówione najważniejsze problemy związane z obliczeniami opartymi na modelach DFA, mające swoje podłoże głównie w stochastycznym sposobie generowania danych oraz zastosowaniu metod symulacyjnych.

2.2.1. Stochastyczny sposób generowania danych

Za występowanie „nieprzewidywalności obliczeń" odpowiedzialnych jest szereg czynników, z których podstawowym jest stochastyczny sposób generowania danych, przyjmowanych na potrzeby symulacji za rzeczywiste dane wejściowe. Bardzo często wiąże się on z nieznajomością rozkładu probabilistycznego danych, które dostarczane są na wejście systemu. Co więcej – rozkład ten nie jest nawet badany pod kątem sparametryzowania, między innymi z powodu nadmiernego skomplikowania. Jedną z powszechnie używanych metod generujących dane jest metoda bootstrap, polegająca na wielokrotnym losowaniu ze zwracaniem próbek z próbki wyjściowej (zazwyczaj będącej danymi historycznymi), a wynik losowania rozpatrywany jest jako rzeczywista wartość wejściowa sytemu [41] [42] [43] [44] [45] [46] [47] [40] [27] [48]. Dane w tym przypadku podawane są na bieżąco (on-line), co powoduje, że do pewnego momentu nie jest możliwe określenie ich charakterystyki. Formalnie, zakończenie obliczeń jest uzależnione od:

(2.2) wartości pewnej funkcji F_{EVAL} analizującej otrzymane dane (Algorytm 2.1, linia 4),

lub

(2.3) osiągnięcia granicznej ilości losowań *maxIterations* (Algorytm 2.1, linia 4).

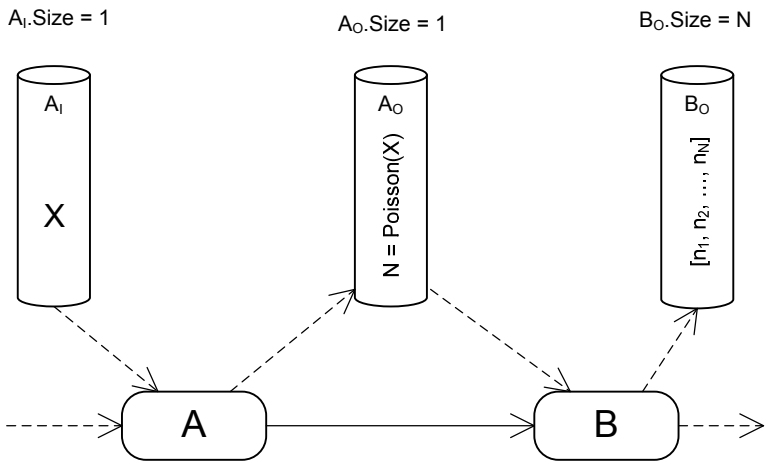
W dalszej części pracy, podczas szczegółowego omawiania problemów Dynamicznych Analizach Finansowych podane zostaną przykłady, dla których istotnym elementem jest warunek (2.2) powodujący nieprzewidywalne zachowanie procesu obliczeniowego.

Algorytm 2.1. Zakończenie obliczeń determinowane wartością danych podawanych „on-line".

```
1)  REPEAT
2)      data = Process(data, newData)
3)      i++
4)  UNTIL (F_EVAL(data) > 0) OR (i ≥ maxIterations)
```

2.2.2. Eksplozja danych

Proces przetwarzania danych podyktowany modelem DFA powoduje uruchomienie na pewnych etapach gałęzi obliczeń częściowych, które charakteryzują się stosunkowo długim czasem wykonania i powodują eksplozję danych, scharakteryzowaną poniżej.



Rysunek 4. Przykład stochastycznego generowania wartości A_O oraz eksplozji danych B_O o niedeterministycznej wielkości.

Przykładem może być fragment modelu przedstawiony na Rysunku 4: dane wejściowe A_I procesu A zawierają pojedynczą wartość *X* będącą liczbą naturalną – parametrem algorytmu generującego wartości z pewnym rozkładem (na przykład rozkładem Poissona). Dane wyjściowe A_O konsumowane następnie przez proces B zawierają pojedynczą liczbę (próbkę) wygenerowaną w stochastyczny sposób (według przyjętego rozkładu). Wartość tej liczby następnie określa rozmiar wektora będącego danymi wyjściowymi B_O procesu B, a tym samym wymagania pamięciowe oraz czas potrzebny do zakończenia obliczeń. Eksplozja danych polega na tym, że B_O.Size >> A_O.Size.

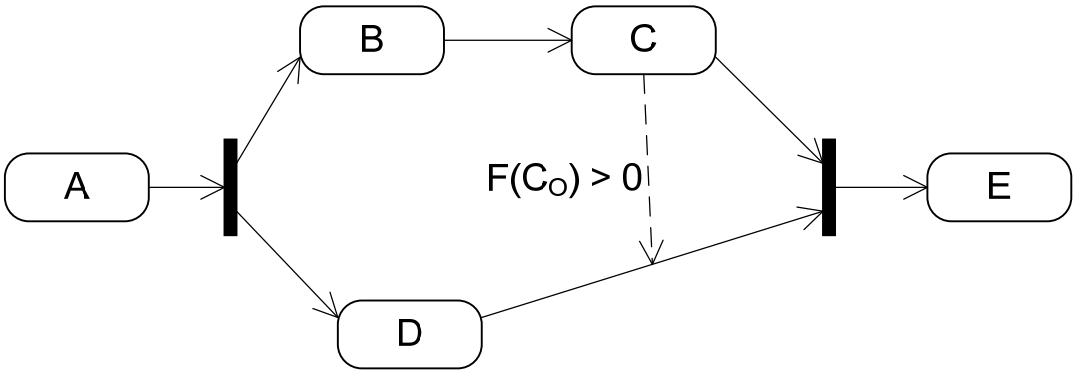
Parametry B w najprostszych przypadkach mogą być linowo zależne od wartości liczby stochastycznej, lecz często wyznaczenie ich jest o wiele bardziej skomplikowane (ze względu na strukturę algorytmów zadania B), a niekiedy wręcz niemożliwe. Warto zwrócić uwagę na fakt,

iż szybkość zakończenia obliczeń przez proces B zależy między innymi od tego, czy przetwarzane dane mieszczą się w pamięci operacyjnej, czy też użyta zostanie dużo wolniejsza pamięć pomocnicza. W drugim przypadku, pomimo niezmiennika – złożoności obliczeniowej B – wykonanie operacji jednostkowej staje się wyraźnie wolniejsze.

Uruchomienie gałęzi obliczeń powodujących eksplozję danych określane jest także bardzo często przez wartości zmiennych stochastycznych podawanych na wejście systemu. Jako przykład może posłużyć modelowanie katastrof naturalnych, które są specyficznym rodzajem rzadko występujących zdarzeń. Wystąpienie takiego zdarzenia powoduje lawinę roszczeń w różnych sektorach jednocześnie [32] [49] [50] [51] [52]. Ponadto rodzaje katastrof, jak i wielkości roszczeń wynikające z ich tytułu, są ściśle ze sobą skorelowane i także wymagają symulacji do określenia rozmiaru. Dopiero w finalnej fazie, po skończeniu losowania, możliwe staje się uruchomienie procesu estymacji rozwoju szkód, dla części których rekompensaty wypłacane są niemal natychmiastowo, a innych (zawierających takie składniki jak rehabilitacje lub renty) rozłożone są na wiele lat.

2.2.3. Warunkowa ewaluacja gałęzi obliczeń

Z punktu widzenia systemu informatycznego przetwarzającego model DFA, problemem jest wyszczególnienie niezależnych potoków obliczeń, a następnie umożliwienie kontroli alokacji zadań uwzględniającej jednocześnie wymagania czasowe oraz pamięciowe. W przypadku systemu rozproszonego, analizy wymaga nie tylko dystrybucja obliczeń na węzły, lecz także późniejsza agregacja wyników mogących być pokaźnymi zbiorami danych. Niezbędna jest również kontrola wielkości jednocześnie przetwarzanych danych przez poszczególne węzły, ze względu na zaletę ograniczenia użycia dużo wolniejszej pamięci pomocniczej: jak zaznaczono wcześniej, możliwość alokacji całego zbioru w pamięci operacyjnej gwarantującej znacznie niższy czas wykonania operacji jednostkowej ma niebagatelny wpływ na szybkość zakończenia obliczeń pomimo niezmiennika – parametrów i złożoności obliczeniowej algorytmu.



Rysunek 5. Warunkowe użycie danych wyjściowych procesu D.

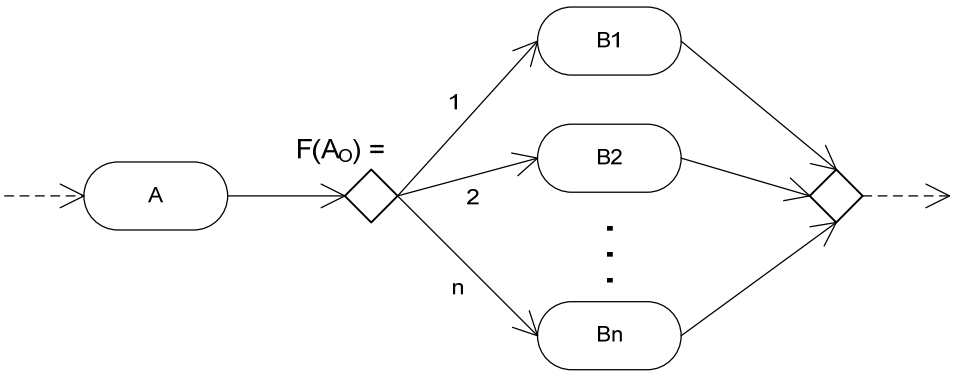
Dla zobrazowania zagadnienia, na Rysunku 5 przedstawiono przykładowy fragment modelu, którego proces E zawsze konsumuje dane wyjściowe C_O procesu C oraz warunkowo dane wyjściowe procesu D w sytuacji, gdy prawdziwa jest nierówność $F(C_O) > 0$ (dla pewnej, ustalonej funkcji F), możliwa do ewaluacji dopiero po ukończeniu działania C. Po wykonaniu procesu A możliwe jest uruchomienie dwóch czasochłonnych gałęzi obliczeń: jednej złożonej z procesów B i C, oraz drugiej – z procesu D.

W przypadku przetwarzania równoległego (również rozproszonego) istnieje potencjalna możliwość uruchomienia D zanim zakończony zostanie C oraz odrzucenia wyjścia D w przypadku negatywnej wartości warunku $F(C_O) > 0$. Efektywność takiego działania w danym momencie w czasie badana jest na podstawie ilości dostępnej pamięci operacyjnej, ilości nieobciążonych węzłów obliczeniowych, szybkości połączeń między węzłami oraz rozmiaru przesyłanych danych.

2.2.4. Dyskretna zmiana strategii

Kolejnym czynnikiem wprowadzającym niepewność są nieustanne ewolucje charakterystyk danych wejściowych. Oczywistymi są stopniowe zmiany częstości występowania poszczególnych typów roszczeń. Niemniej ważne są na bieżąco zmieniające się indeksy spółek giełdowych, kursy wymiany walut, stóp procentowych lub inflacja – niezbędne do predykcji przyszłych trendów oraz alokacji zebranych środków. Niebagatelny wpływ mają również zmiany społeczno – polityczne, a także legislacyjne [10] [32] [9] [29]. Wyżej wymienione czynniki w połączeniu z wrażliwą na zmiany siecią wewnętrznych zależności w znacznym stopniu odróżniają modele DFA od klasycznego drążenia danych.

W systemach Data Mining niewielki przyrost informacji nie powodował konieczności całkowitej rekalkulacji, a jedynie uaktualnienia pewnych zmiennych będących obszarem zainteresowania. W procesach opartych na DFA aktualizacja wyników spowodowana zmianą jednego współczynnika może spowodować skokową, nieprzewidywalną zmianę strategii i jednocześnie całego łańcucha obliczeń.



Rysunek 6. Dyskretna zmiana strategii.

Za przykład może posłużyć kryterium wyboru firm reasekuracyjnych oraz stopień cesji ryzyka i wielkości składek na nie przeniesionych, w którym możliwa konfiguracja ograniczona jest zbiorem dyskretnych wartości. Na pewnym etapie może także zmienić się dopasowanie jednego z możliwych modeli szacowania szkód do danych wejściowych, które określane jest na podstawie pewnej zależności. Inny – na daną chwilę lepszy – wybór w dużym stopniu zmienia działanie systemu. Na Rysunku 6 przedstawiony został fragment modelu, w którym następuje dyskretna zmiana strategii spośród B_1 - B_n, o potencjalnie znacząco różnych złożonościach czasowych oraz pamięciowych. Wybór strategii zależy od wartości danych wyjściowych A_O procesu A (jest określony pewną ustaloną funkcją F o wartościach z przedziału $[1, n] \cap \mathbb{N}$). Użyteczne w praktyce oszacowanie czasu zakończenia ewaluacji przedstawionego fragmentu (w celu optymalizacji alokacji zadań) możliwe jest więc dopiero po uzyskaniu wyników A_O.

2.2.5. Replikacja danych poprzez wielokrotną generację

W modelach DFA, pewne losowo wygenerowane dane przyjmowane są za parametry rzeczywiste i używane na wielu etapach ewaluacji modelu. Dane te, generowane na podstawie kilkudziesięciu parametrów mogą przybierać duże rozmiary. Przykładowo, w momencie ustalenia przyszłych trendów ekonomicznych (indeksów spółek giełdowych, stóp oprocentowania, etc) dla alokacji środków z wpływów uzyskanych dla jednej linii odszkodowań, te same dane (o pokaźnym rozmiarze) muszą posłużyć za podstawę alokacji reszty majątku. Zachowane muszą pozostać także korelacje pomiędzy poszczególnymi czynnikami w całym systemie [53] [54].

Celowym z punktu realizacji systemu informatycznego staje się modelowanie izolowanej generacji pojedynczej instancji danych oraz dostępu do współdzielonego zasobu przez kolejne (potencjalnie rozproszone) procesy. Wadą takiego rozwiązania jest konieczność transferu dużego strumienia danych na inne węzły obliczeniowe.

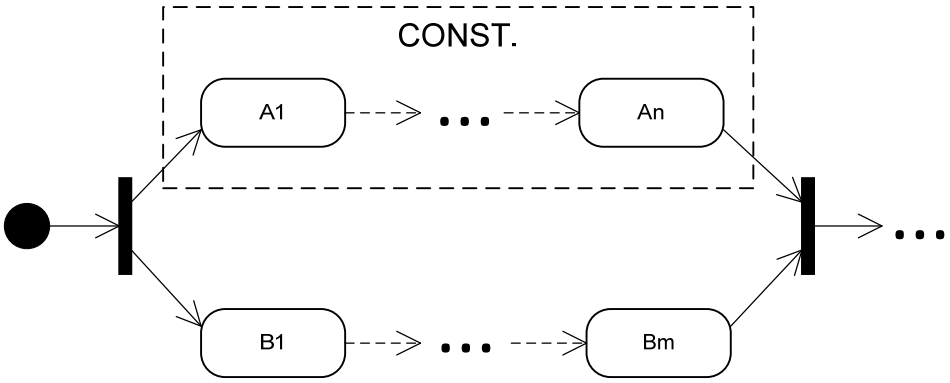
Innym – często wykorzystywanym rozwiązaniem – jest możliwość replikacji poprzez przesłanie jedynie niewielkiego zbioru parametrów oraz uruchomienie identycznych procesów generujących te same dane wyjściowe na węzłach docelowych. Gwarancją identycznych wartości danych wyjściowych jest pseudolosowy sposób generowania danych oparty na tym samym algorytmie, bazującym na wcześniej ustalonym ziarnie.

Podsumowując: w procesach DFA – w celu optymalizacji czasu wykonania – zamiast klonowania zbiorów danych stochastycznych na inne węzły obliczeniowe, możliwe jest przesłanie jedynie parametrów oraz sposobu ich generacji. Opisany schemat działania ma bardzo często zastosowanie w przypadku, gdy grupy aktuariuszy pracujących w różnych lokalizacjach zmuszone są do korzystania z identycznych danych wejściowych przedstawiających parametry ekonomiczne, w celu uzyskania spójnych wyników obliczeń globalnych.

2.2.6. Powtarzalność obliczeń w kolejnych iteracjach

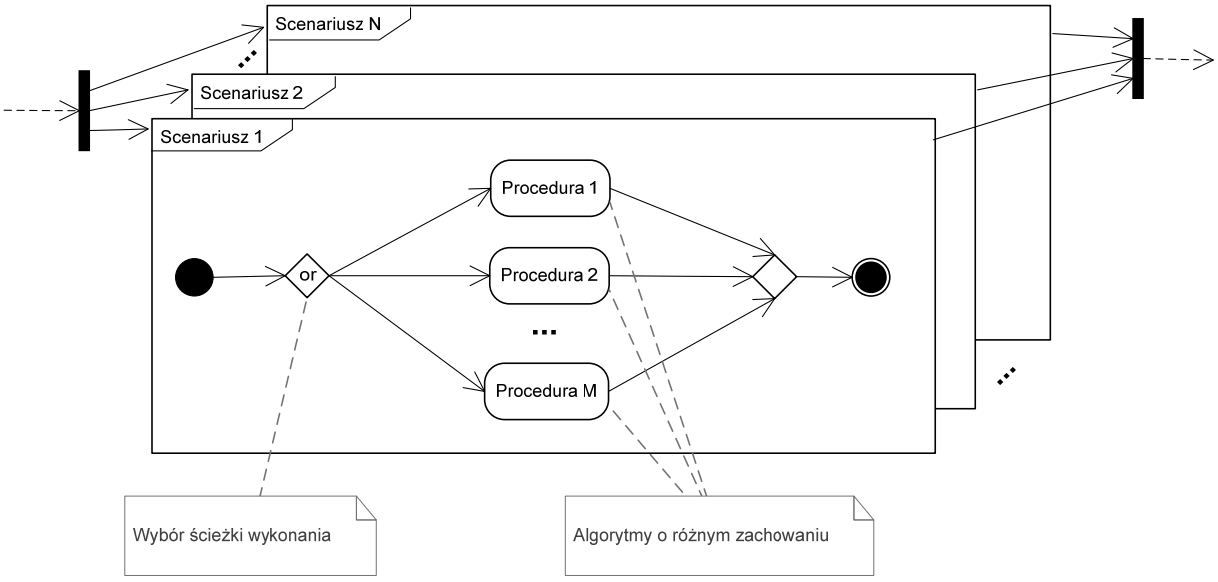
Gotowy (skalibrowany) model DFA bardzo często nie jest uruchamiany jednokrotnie, lecz służy do obserwacji korelacji ryzyka oraz zysków: wspomaga podejmowanie decyzji poprzez testowanie różnych strategii badanej korporacji [7] [9] [38]. Każda iteracja związana jest z odmiennymi parametrami modelu i może skutkować zupełnie różnymi scenariuszami przebiegu wykonania podprocesów. W użyciu są także metody iteracyjnego dopasowania najlepszego modelu do danych wejściowych, polegające na wyznaczaniu punktu stałego. W tym przypadku uruchamiane podprocesy oraz generacja wartości w następnej iteracji uzależniona jest od wyjścia poprzednich kroków [55], co może skutkować szybkim wzrostem rozmiaru danych.

Bardzo często obserwowaną cechą jest powtarzalność obliczeń dla pewnego fragmentu modelu DFA: pomimo zmian parametrów podczas kolejnych uruchomień, pewna – nierzadko czasochłonna – gałąź obliczeń pozostaje niezmieniona, generując identyczne rezultaty (jak na przykład obszar oznaczony „const.” na Rysunku 7). Fakt ten daje potencjalną możliwość ponownego użycia raz wygenerowanego zbioru danych, lub w przypadku uruchomienia modelu na innym węźle środowiska, przekazania dokładnych charakterystyk użycia pamięci oraz procesora w celu polepszenia wydajności obliczeń.



Rysunek 7. Powtarzalność obliczeń fragmentu modelu DFA w kolejnych iteracjach.

2.2.7. Przetwarzanie równoległe scenariuszy niezależnych



Rysunek 8. Różne ścieżki wykonania zależne od rozpatrywanego scenariusza.

Obliczenia oparte na modelach DFA polegają na testowaniu tysięcy, a nawet milionów scenariuszy przedstawiających różne warianty modelowanego świata. Rysunek 8 zawiera przykład: zbiór niezależnych scenariuszy o identycznej strukturze, których przebieg wykonania zależny jest od danych historycznych, obranych parametrów, strategii, a także wartości zmiennych stochastycznych, które de facto są wartościami zmiennej pseudolosowej o pewnym – czasami bardzo skomplikowanym – rozkładzie. Pomimo niezmiennej struktury modelu dla wszystkich scenariuszy,

przebieg każdej z symulacji może się zasadniczo różnić, co pociąga za sobą konsekwencje w postaci odmiennych wymagań pamięciowych, a także użycia czasu procesora. Określenie przebiegu scenariusza jest prawie zawsze bardzo trudne, a praktycznie wręcz niemożliwe przed zakończeniem jego ewaluacji.

Rozpatrywane scenariusze są niezależne, a ich synchronizacja w celu zebrania statystyk oraz podjęcia decyzji dotyczącej dalszego wykonania przeprowadzana jest na ściśle określonych etapach. W związku z tym, istnieje możliwość ich równoległego (w tym rozproszonego) przetwarzania. W przypadku rozproszenia części scenariuszy na inne węzły obliczeniowe, konieczne staje się także przeniesienie na węzły docelowe używanych przez nie danych lokalnych oraz transmisja danych wynikowych z powrotem w celu analizy rezultatów globalnych.

Definicja 2.1

Koszt ewaluacji zadania T na węźle obliczeniowym N – oznaczany $T_{EV}(T @ N)$ – to czas potrzebny na generację danych wyjściowych przez zadanie T, na podstawie danych wejściowych. Uruchomione zadanie T, dane wejściowe oraz wyjściowe zalokowane są na węźle N.

■

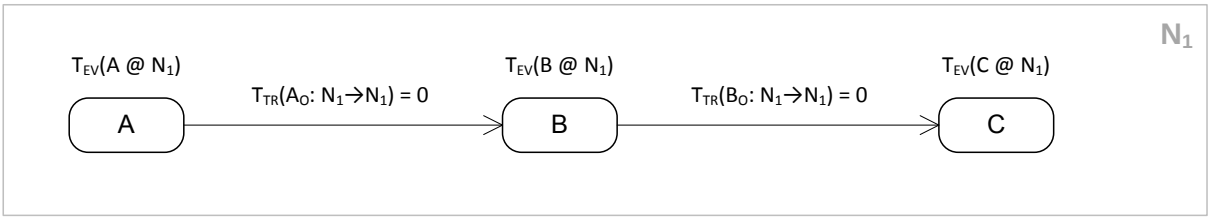
Definicja 2.2

Koszt transferu danych D z węzła obliczeniowego N_1 na węzeł N_2 – oznaczany $T_{TR}(D: N_1 \rightarrow N_2)$ – to czas potrzebny sklonowanie danych D z węzła N_1 na węzeł N_2.

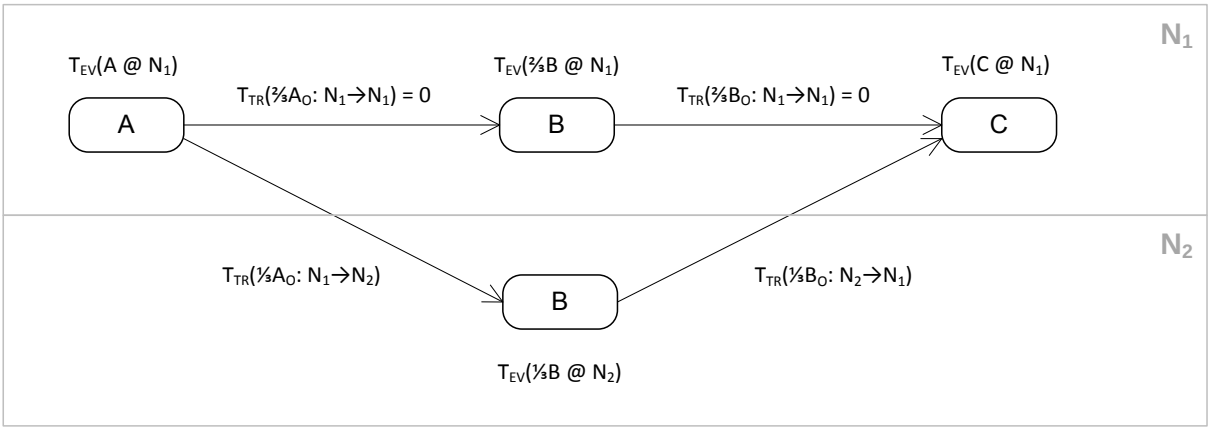
■

Opłacalność rozproszenia obliczeń – przy założeniu kryterium: minimalizacji czasu działania – została zobrazowana na przykładzie. Rysunek 9a zawiera schemat ewaluacji wszystkich scenariuszy na jednym węźle obliczeniowym, natomiast Rysunek 9b przedstawia przeniesienie części obliczeń na inny procesor. Dla uproszczenia przyjęto dużą liczbę scenariuszy oraz założono średnie wymagania pamięciowe oraz użycie procesora dla każdego z nich na takim samym poziomie.

a)



b)



Rysunek 9. Przykład dystrybucji części zadań na inny węzeł obliczeniowy.

Schemat a) przedstawia całość obliczeń przeprowadzoną na jednym węźle N_1: zadania A, B oraz C przetwarzają wszystkie scenariusze, co wymaga czasu odpowiednio $T_{EV}(A @ N_1)$, $T_{EV}(B @ N_1)$ oraz $T_{EV}(C @ N_1)$. Wymiana danych pomiędzy zadaniami A, B i C odbywa się w obrębie tego samego węzła obliczeniowego, w związku z tym przyjęto brak związanych z tym opóźnień: $T_{TR}(A_O: N_1 \rightarrow N_1) = 0$ oraz $T_{TR}(B_O: N_1 \rightarrow N_1) = 0$ odpowiednio dla danych wyjściowych zadania A i B.

Schemat b) obrazuje część (dla uproszczenia ⅓) scenariuszy uruchomioną na węźle N_2. Przyjęto czasy przetwarzania B równe $T_{EV}(⅔B @ N_1)$ na węźle N_1 oraz $T_{EV}(⅓B @ N_2)$ na N_2. Dodatkowo transmisja ⅓ danych A_O oraz B_O pomiędzy węzłami N_1 i N_2 powoduje opóźnienia: $T_{TR}(⅓A_O: N_1 \rightarrow N_2)$ oraz $T_{TR}(⅓B_O: N_2 \rightarrow N_1)$.

Ostateczna opłacalność przypadków a) i b) może zostać oszacowana poprzez porównanie czasów T_a i T_b, które wynoszą:

$$T_a = T_{EV}(A @ N_1) + T_{EV}(B @ N_1) + T_{EV}(C @ N_1) \quad (2.4)$$

$$T_b = T_{EV}(A @ N_1) + \max(T_{b1}, T_{b2}) + T_{EV}(C @ N_1) \quad (2.5)$$

gdzie

$$T_{b1} = T_{EV}(⅔B @ N_1)$$

$$T_{b2} = T_{TR}(⅓A_O: N_1 \rightarrow N_2) + T_{EV}(⅓B @ N_2) + T_{TR}(⅓B_O: N_2 \rightarrow N_1) \blacksquare$$

Wartości T_{EV} zależne są od wymagań zadania oraz mocy obliczeniowej węzła, natomiast T_{TR} od wielkości danych i szybkości połączenia między węzłami.

2.2.8. Podział modelu DFA na komponenty

W praktyce projekt DFA podzielony jest na komponenty, będące oddzielnymi fragmentami modelu. Podział na komponenty ułatwia przydzielenie pewnej, ściśle zdefiniowanej części modelu DFA grupie specjalistów, która posiada wymagane kwalifikacje uprawniające do wprowadzania zmian oraz audytu. Komponenty mogą być używane wielokrotnie oraz niezależnie uaktualniane, co znacznie podnosi wydajność tworzenia modelu, jego przejrzystość i odporność na błędy. Podział na komponenty w naturalny sposób staje się kryterium wyznaczania ziarnistości (niepodzielności zadań), na poziomie której odbywać się może szacowanie wymagań czasowych oraz pamięciowych lub dystrybucja podzadań na inne procesory.

W trakcie trwania doświadczeń zaobserwowano szereg zjawisk opisanych w niniejszym rozdziale zachodzących na poziomie abstrakcji komponentu: na przykład wyniki pewnych komponentów pozostają niezmienne dla każdej iteracji uruchomienia modelu (badanie *return and risk*); dyskretna zmiana strategii powoduje uruchomienie obliczeń definiowanych przez cały komponent.

W niniejszym rozdziale przedstawione zostało jedynie spostrzeżenie związane z pewnymi fragmentami modelu, naturalnie wydzielonymi w trakcie prac nad jego tworzeniem. Formalna definicja oraz praktyczne skutki wykorzystania komponentów omówione zostaną w dalszej części rozprawy.

2.3. Podsumowanie

Dynamiczne Analizy Finansowe są tematem szeroko dyskutowanym z ekonomicznego punktu widzenia, natomiast zagadnienia dotyczące implementacji, czyli reprezentacja modelu DFA oraz kontrola wykonania są całkowicie pomijane. Z tego powodu brak jest ścisłego modelu informatycznego, istnieją jedynie fragmentaryczne opisy działania bazujące na notacji UML [16], w szczególności na *diagramach aktywności*. Z powodu dużego skomplikowania zadania oraz braku narzędzi nie zostały dotychczas podjęte próby wyszczególnienia problemów pojawiających się podczas implementacji DFA oraz próby wskazania rozwiązań mających polepszyć wydajność.

Podsumowując: przedstawiona powyżej klasa problemów, całkowicie odmienna od dotychczas znanych i przeanalizowanych, wymaga nowego sposobu modelowania oraz kontroli. Powodem takiego stanu rzeczy jest koniunkcja szeregu czynników zaobserwowanych przez autora w trakcie przeprowadzonych badań, mających wpływ na przebieg procesu obliczeń, z których najważniejsze to:

- nieprzewidywalna sekwencja uruchomienia oraz czas trwania podprocesów, uzależnione od wartości danych wejściowych,
- nieznana a priori ilość przetwarzanych danych, uzależniona od wartości wektora wejściowego, a nie tylko jego wielkości, oraz zjawisko „eksplozji” danych,
- niewielki relatywnie przyrost informacji wymuszający całkowite powtórzenie procesu drążenia danych,

- potrzeba powtarzania eksperymentów ze zmodyfikowanymi parametrami oraz warunek stopu zależny od wyników obliczeń dla danej iteracji.

W rezultacie dekompozycja obliczeń – w celu ich przyśpieszenia – musi odbywać się na bieżąco (ang. *online*) w trakcie wykonywanych analiz, co stanowi nowe wyzwanie w stosunku do wcześniej stosowanych metod drążenia danych, gdzie procesy obliczeniowe mogły być dekomponowane statycznie, przed rozpoczęciem obliczeń. Na system bardzo często nałożone są dodatkowo ograniczenia czasowe wymuszające podanie wyniku nie później, niż zostało to ustalone. W dotychczasowych pracach nie został ściśle zdefiniowany żaden mechanizm pozwalający na spójne opisanie zadania Dynamicznych Analiz Finansowych oraz umożliwiający praktyczną kontrolę wykonania przedstawionego systemu. Z tego też powodu nie jest możliwa optymalizacja alokacji zasobów w środowisku rozproszonym, mająca na celu skrócenie czasu zakończenia obliczeń.

3. Statyczny model DFA

W poprzednim rozdziale przedstawiono specyfikę Dynamicznych Analiz Finansowych oraz wskazano cechy powodujące, że rozważane dotychczas mechanizmy opisu i kontroli wykazały niewystarczającą ścisłość oraz jednolitość. Dynamiczne Analizy Finansowe stawiają nowe wymagania w stosunku do konstrukcji modelu, między innymi ze względu na nieprzewidywalność sekwencji uruchomienia oraz czasu wykonania procesów, nieznaną a priori ilość przetwarzanych danych, a także zjawisko „eksplozji" danych.

Zaprezentowane we wcześniejszym rozdziale problemy występujące w Dynamicznych Analizach Finansowych, zilustrowane zostały przykładami opisanymi głównie *diagramami czynności* (ang. *activity diagrams*) języka UML [16], aczkolwiek wymagały one długiego opisu słownego w języku naturalnym w celu sprecyzowania szczegółów. Z tego powodu *diagramy czynności* uniemożliwiają wprowadzenie automatyzacji, zarówno na etapie projektowania (definiowania) modelu, jak również w celu przeprowadzenia obliczeń.

W praktyce niewielkie części statycznego modelu DFA przedstawiane są jako *diagram przepływu sterowania* (ang. *CFD - Control Flow Diagram*), który jest zawężeniem *diagramu aktywności*. Jest to formalizm niewystarczający do szerszego opisu problemu (posiada wady *diagramu aktywności*), a dodatkowo nie umożliwia reprezentacji pętli typu *while*. Notacja ta, pomimo wspomnianych ograniczeń, jest często wykorzystywana przez ekonomistów do przedstawienia zarysu podproblemów, a stosunkowo dużą popularność zdobyła dzięki swojej prostocie.

Pomimo istnienia formalizmów opartych o diagramy aktywności notacji UML fragmentarycznie ujmujących przedstawione zagadnienia, nie został dotychczas zdefiniowany spójny model:

(3.1) opisujący wszystkie właściwości zadania Dynamicznych Analiz Finansowych,

(3.2) stanowiący bazę do opisania zachowania procesów w czasie,

(3.3) umożliwiający kontrolę oraz sterowanie wykonaniem procesów,

(3.4) określający kryteria polepszające alokację zadań, w celu zmniejszenia czasu potrzebnego do zakończenia obliczeń.

W pracy [17] (Kotulski, 2000) wykazano, że grafy atrybutowane stanowią intuicyjny mechanizm o wystarczającej sile, aby opisać stan systemu rozproszonego w pewnym momencie w czasie. Natomiast transformacje grafowe wydają się być najodpowiedniejszym formalnym narzędziem pozwalającym na opisanie dynamiki zmian tegoż systemu [56] [57] [17] [58] [59] [21].

W niniejszym rozdziale zdefiniowany zostanie Statyczny Model DFA, będący odpowiedzią na postawiony powyżej problem (3.1), a także stanowiący podstawę do realizacji postulatów (3.2) - (3.4), co zostanie omówione w kolejnych rozdziałach. Model statyczny – budowany przez aktuariuszy – jest klasą problemu i przedstawia w zunifikowany sposób operacje oraz parametry istotne z punktu widzenia późniejszej implementacji.

Statyczny Model DFA definiuje:

- kolejność operacji oraz zależności pomiędzy operacjami,
- złożoność pamięciową oraz obliczeniową każdej z operacji,
- wielkości danych „konsumowanych” oraz „produkowanych” przez operacje.

Definicja 3.1

Graf EDG (ang. directed node- and edge-labelled graph) [60], to piątka $H = (V, D, \Sigma, \Gamma, \delta)$, gdzie:

V – jest skończonym, niepustym zbiorem wierzchołków, którym przypisano unikalne indeksy wyznaczające porządek na zbiorze V,

D – jest zbiorem krawędzi postaci (v, μ, w), gdzie $v, w \in V$, $\mu \in \Gamma$,

Σ – jest zbiorem etykiet wierzchołków,

Γ – jest zbiorem etykiet krawędzi,

$\delta: V \rightarrow \Sigma$ – jest funkcją etykietującą wierzchołki. ■

Definicja 3.2

Graf aEDG (ang. attributed directed node- and edge-labelled graph) [61], to graf EDG $H = (V, D, \Sigma, \Gamma, \delta)$, dla którego Σ i Γ zdefiniowano w następujący sposób:

$$\Sigma = NL \times (NA \rightarrow_n NV)$$

gdzie:

NL – jest zbiorem etykiet głównych wierzchołków,

NA – jest zbiorem nazw atrybutów wierzchołków,

NV – jest zbiorem wartości atrybutów wierzchołków,

$\rightarrow_n$ – jest funkcją częściową z NA do NV,

$$\Gamma = EL \times (EA \rightarrow_e EV)$$

gdzie:

EL – jest zbiorem etykiet głównych krawędzi,

EA – jest zbiorem nazw atrybutów krawędzi,

EV – jest zbiorem wartości atrybutów krawędzi,

$\rightarrow_e$ – jest funkcją częściową z EA do EV. ■

3.1. Formalna definicja modelu DFA-Static

Model statyczny będzie podstawą do zaprezentowanego w dalszej części pracy mechanizmu pozwalającego na opisanie, kontrolę oraz optymalizację zachowania procesów w czasie obliczeń. Zawarta w nim informacja będzie niezbędna do określenia kolejności uruchomienia zadań, a także pomocna do polepszenia wydajności i tym samym przyspieszenia momentu zakończenia obliczeń.

Poniżej zdefiniowany zostanie statyczny model DFA, omówione zostaną jego cechy oraz celowości ich wprowadzenia.

Definicja 3.3

Statyczny Model DFA (DFA-Static) to graf aEDG H = (V, D, Σ, Γ, δ), gdzie Σ = NL $\times$ (NA $\rightarrow_n$ NV) oraz Γ = EL $\times$ (EA $\rightarrow_e$ EV), dla którego ustalone zostały:

NL = { IS, FS, I, O, T, S },

NA = { PC, MC, ME, OP, DS, SF, OP, Name, History },

oraz wprowadzone zostały ograniczenia sąsiedztwa wierzchołków, w zależności od przypisanych etykiet ze zbioru NL, zawarte w Tabeli 3.

■

Statyczny model DFA jest sposobem zapisu algorytmu. DFA-Static jest grafem, którego wierzchołki połączone krawędziami skierowanymi, związane są z pewnymi czynnościami. Krawędź wychodząca z wierzchołka A do wierzchołka B oznacza, iż po wykonaniu czynności związanej z wierzchołkiem A, możliwe jest wykonanie czynności związanej z wierzchołkiem B.

Znaczenie etykiet wierzchołków grafu DFA-Static przedstawiono poniżej:

- IS – stan początkowy (ang. Initial State)
- FS – stan końcowy (ang. Final State)
- I – dane wejściowe (ang. Input Data)
- O – dane wyjściowe (ang. Output Data)
- T – zadanie (ang. Task)
- S – selekcja (and. Selection)

W Tabeli 1 zawarte zostały atrybuty wierzchołków związane z poszczególnymi etykietami, opisujące parametry modelu DFA. Każdy z wierzchołków posiada dodatkowo dwa atrybuty:

- Name – o unikalnej (w obrębie grafu) wartości, pozwalający na jednoznaczną i przejrzystą identyfikację, szczególnie przydatną na etapie tworzenia oraz omawiania modelu,

- History – pozwala na zapamiętanie wartości związanych z wcześniejszymi ewaluacjami modelu, które mogą zostać wykorzystane w celu przyspieszenia kolejnych iteracji obliczeń (na przykład w przypadku, gdy dane wejściowe fragmentu modelu nie ulegają zmianie, możliwe jest powtórne użycie raz uzyskanych rezultatów). Temat poruszający wykorzystanie danych historycznych zostanie szerzej przedyskutowany w dalszej części pracy.

Ustalono, że każdy z atrybutów opisanych w Tabeli 1 oraz atrybut „History" może przyjmować specjalną wartość, nazwaną „**wartością nieokreśloną**" i oznaczoną „?". Przypadki użycia wartości nieokreślonej zostaną przedyskutowane w dalszej części pracy.

Dla zwiększenia przejrzystości, wartości atrybutów oznaczane będą na dwa sposoby, w zależności od kontekstu, w którym zostaną użyte. Dla przykładu: $PC(T_A)$ oraz $T_A.PC$ oznaczają wartość atrybutu PC wierzchołka T_A.

Etykieta	Atrybut	Opis atrybutu	Jednostka
T	PC	Processor Consumption	operacja
T	MC	Memory Consumption	B
T	ME	Memory Efficiency	-
T	OP	Operation	-
I	DS	Data Size	B
O	DS	Data Size	B
S	SF	Selection Function	-
S	OP	Operation	-

Tabela 1. Atrybuty wierzchołków grafu DFA-Static.

Statyczny model DFA ustala kolejność wykonania akcji związanych z wierzchołkami. Pierwsze zadanie (inicjujące) reprezentowane jest przez wierzchołek oznaczony etykietą „IS" (ang. Initial State), posiadający jedynie krawędzie wychodzące. Natomiast ostatnim zadaniem (stanem końcowym) jest wierzchołek z etykietą „FS" (ang. Final State), posiadający jedynie krawędzie przychodzące.

Wierzchołek etykietowany „T" (ang. Task) utożsamiany jest z zadaniem obliczeniowym definiowanym przez aktuariuszy, którym może być zarówno atomowa operacja matematyczna (na przykład dodawanie dwóch liczb), jak również cały szereg operacji zamkniętych w „czarną skrzynkę". Zadanie przetwarza dane wejściowe – oznaczone etykietą „I" (ang. Input Data) oraz produkuje dane wyjściowe – etykietowane „O" (ang. Output Data). Zadanie T można traktować jako funkcję, której argumentami są dane wejściowe, a wartościami – dane wyjściowe. Działanie zadania może być definiowane przez skomplikowaną strukturę, będącą samą w sobie modelem statycznym, co zostanie szczegółowo omówione w dalszej części pracy.

Wierzchołek oznaczony etykietą „S” (ang. Selection) zawiera warunek określający, które krawędzie wychodzące prowadzą do wierzchołków będących kolejnymi zadaniami do wykonania.

Wierzchołkom o różnych etykietach nadana została odmienna reprezentacja graficzna, przedstawiona w Tabeli 2. Dla skrócenia oraz zwiększenia przejrzystości zapisu, w dalszej części pracy zwrot „wierzchołek T” utożsamiany będzie z wierzchołkiem oznaczonym etykietą „T”, natomiast krawędź skierowana *(A, B)* oznaczana będzie zamiennie A → B.

 Input Data	 Output Data	 Task
 Selection	 Initial State	 Final State

Tabela 2. Reprezentacje graficzne wierzchołków grafu statycznego modelu DFA.

Definicja 3.4

Następnik wierzchołka *v* grafu skierowanego G, to wierzchołek *w* taki, że istnieje krawędź $(v, w) \in E(G)$.

Poprzednik wierzchołka *w* grafu skierowanego G, to wierzchołek *v* taki, że istnieje krawędź $(v, w) \in E(G)$.

■

Definicja 3.5

Ścieżka grafu G o **długości** *n*, to ciąg *n*+1 wierzchołków grafu G taki, że pomiędzy każdą parą kolejnych wierzchołków istnieje krawędź. Formalnie: $(v_0, v_1, ..., v_n)$ jest ścieżką $\Leftrightarrow$ ($\forall i \in [0, ..., n]$ $v_i \in V(G) \Rightarrow \forall k: k > 0 \wedge k \leq n \; \exists (v_{k-1}, v_k) \in E(G)$).

■

W Tabeli 3 zawarto dopuszczalne etykietowanie sąsiadów wierzchołka o zadanej etykiecie.

Etykieta poprzednika	Etykieta wierzchołka	Etykieta następnika
I	T	O
IS, O, S	I	T
T	O	FS, I, S
O, S	S	I, S, FS
-	IS	I
O, S	FS	-

Tabela 3. Dopuszczalne etykietowanie sąsiadów wierzchołków w grafie DFA-Static.

Poniżej, omówione zostaną operacje oraz atrybuty związane z poszczególnymi etykietami wierzchołków grafu Statycznego Modelu DFA.

3.1.1. Stan początkowy (IS)

W grafie DFA-Static występuje tylko jeden wierzchołek etykietowany „IS" i jako jedyny nie posiada krawędzi przychodzących. Jest to punkt startowy sekwencji obliczeń reprezentowanych przez graf.

Przyjęte zostało następujące założenie:

(3.5) z wierzchołka stanu początkowego istnieje co najmniej jedna ścieżka prowadząca do każdego wierzchołka grafu DFA-Static.

3.1.2. Stan końcowy (FS)

W grafie DFA-Static także występuje tylko jeden wierzchołek etykietowany „FS" i jako jedyny nie posiada krawędzi wychodzących. Osiągnięcie stanu końcowego oznacza zakończenie procesu obliczeń.

Przyjęte zostało następujące założenie:

(3.6) z każdego wierzchołka grafu DFA-Static istnieje co najmniej jedna ścieżka prowadząca do wierzchołka stanu końcowego.

3.1.3. Dane wejściowe (I)

Wierzchołki etykietowane „I" reprezentują zbiory danych wejściowych przetwarzanych przez zadania. Atrybut „DS" (ang. Data Size), który określa rozmiar danych, może zależeć od wartości „DS" najbliższego poprzedzającego wierzchołka „O" albo być wartością (funkcją) stałą.

3.1.4. Dane wyjściowe (O)

Wierzchołki etykietowane „O" reprezentują binarne zbiory danych wyjściowych, produkowane przez zadania „T". Atrybut „DS" (ang. Data Size), który określa rozmiar danych, zależy od operacji „T" oraz wartości „DS" wierzchołków „I", będących poprzednikami „T". Formalnie:

$$DS(O) = F_{DS}(O, T, \{I: I \to T\}), \qquad \text{gdzie } T \to O \tag{3.7}$$

3.1.5. Selekcja (S)

Wierzchołek etykietowany „S” pozwala na wybór krawędzi prowadzących do następnego zadania. Wybór ten jest determinowany przez funkcję „SF” (ang. Secision Function). Funkcja ta operuje na wartościach danych wyjściowych poprzedzającego wierzchołka, a jej wynik określa dalsze ścieżki operacji do wykonania.

Przyjęte zostało następujące założenie:

(3.8) parametr OP wierzchołka „S” przyjmuje zawsze wartość nieokreśloną „?”, która oznacza niemożliwość określenia ilości operacji potrzebnych na przetworzenie wierzchołka „Selection”.

Warunek (3.8) ma związek z użyciem wierzchołka „S” do przedstawienia pętli „while”, czego następstwa omówione zostaną w dalszej części pracy poświęconej optymalizacji czasu ewaluacji modelu DFA w środowisku rozproszonym.

3.1.6. Zadanie (T)

Wierzchołek „T” reprezentuje operacje przetwarzające dane wejściowe oraz generujące dane wyjściowe. Poprzednikami wierzchołka „T” jest zawsze jeden lub więcej wierzchołków „I”, natomiast następnikami jeden lub więcej wierzchołków „O”.

W praktyce, każdy z wierzchołków przychodzących zadania „T”, będący danymi wejściowymi (wychodzących / danymi wyjściowymi) utożsamiany jest z fizycznie odrębnym zbiorem danych, na przykład plikiem, bazą danych lub tabelą bazy danych.

Zadanie „T” może być utożsamiane z funkcją F_T, o argumentach będących zbiorami danych wejściowych, która zwraca wektor wartości w postaci zbiorów danych wyjściowych:

$$(O_1, \dots, O_n) = F_T(I_1, \dots, I_m) \tag{3.9}$$

Wierzchołek T posiada następujące atrybuty, będące parametrami istotnymi dla optymalizacji szybkości działania systemu informatycznego:

- PC (ang. Processor Consumption) – określa ilość operacji procesora, potrzebną do zakończenia działania, zależną od wykonywanego zadania oraz rozmiarów danych wejściowych:

$$PC(T) = F_{PC}\left(T,\ I_{T,1}.DS, \dots, I_{T,n}.DS\right), \qquad \text{gdzie } I_{T,k} \to T \tag{3.10}$$

- MC (ang. Memory Consumption) – określa wielkość pamięci wymaganą do przeprowadzenia obliczeń, zależną także od wykonywanego zadania oraz rozmiarów danych wejściowych:

$$MC(T) = F_{MC}\left(T,\ I_{T,1}.DS, \dots, I_{T,n}.DS\right), \qquad \text{gdzie } I_{T,k} \to T \tag{3.11}$$

- ME (ang. Memory Efficiency) – określa funkcję $F_{ME}(T)$: $\mathbb{R} \to \mathbb{R}$ dla zadania T, charakteryzującą wpływ niedoboru pamięci środowiska wykonania na efektywność algorytmu obliczeniowego.

Argumentami funkcji są wartości będące stosunkiem MC / AM, gdzie MC (ang. Memory Consumption) jest wymaganą ilością pamięci do przeprowadzenia obliczeń, natomiast AM (ang. Available Memory) jest dostępną ilością pamięci operacyjnej węzła obliczeniowego (parametr AM zostanie dokładnie przedyskutowany w dalszej części rozprawy). Wartości funkcji określają szybkość wykonania operacji jednostkowej. Jak wspomniano wcześniej: pomimo niezmiennika – złożoności obliczeniowej algorytmu – czas wykonania operacji jednostkowej ulega zmianie w zależności, czy używana jest jedynie pamięć operacyjna albo zachodzi konieczność wykorzystania pamięci pomocniczej. Obecny stan badań nad funkcją Memory Efficiency zakłada osobny parametr dla każdego z wierzchołków grafu, lecz niewykluczone, iż istnieje możliwość wprowadzenia jednego parametru dla całej klasy algorytmów. Wykres przykładowej funkcji Memory Efficiency przedstawiony został na Rysunku 52 (strona 107) w Rozdziale 7.2, omawiającym doświadczenia związane z wyznaczaniem tego parametru.

- OP (ang. Operation) – określa algorytm do wykonania, w celu uzyskania danych wyjściowych na postawie danych wejściowych, w ogólnym przypadku zdefiniowany wzorem (3.9).

W modelach DFA bardzo często pojawiają się operacje wektorowe: uzyskanie wartości $Y = (y_1, \dots, y_N)$, przy zadanych $X^j = \left(x_1^j, \dots, x_N^j\right)$, $j = 1..M$ oraz funkcji $f_{OP}(T)$ zależnej od zadania T, równoważne jest wyliczeniu wszystkich y_i:

$$y_i = f_{OP}(T)\left(x_i^1, \dots, x_i^M\right) \qquad \text{dla } i = 1..N \tag{3.12}$$

gdzie $f_{OP}(T)$ może być zarówno prostą operacją dodawania, jak również skomplikowanym losowaniem próbek z pewnym rozkładem o zadanych parametrach. Wśród operacji wektorowych, często występującym przypadkiem szczególnym są operacje skalarne, jak na przykład mnożenie dwóch liczb.

Gdy parametr OP posiada wartość (także wartość nieokreśloną „?”), wtedy wierzchołek T jest *zadaniem elementarnym*, co oznacza, że określona operacja wykonywana jest wyłącznie w ramach jednego zadania T.

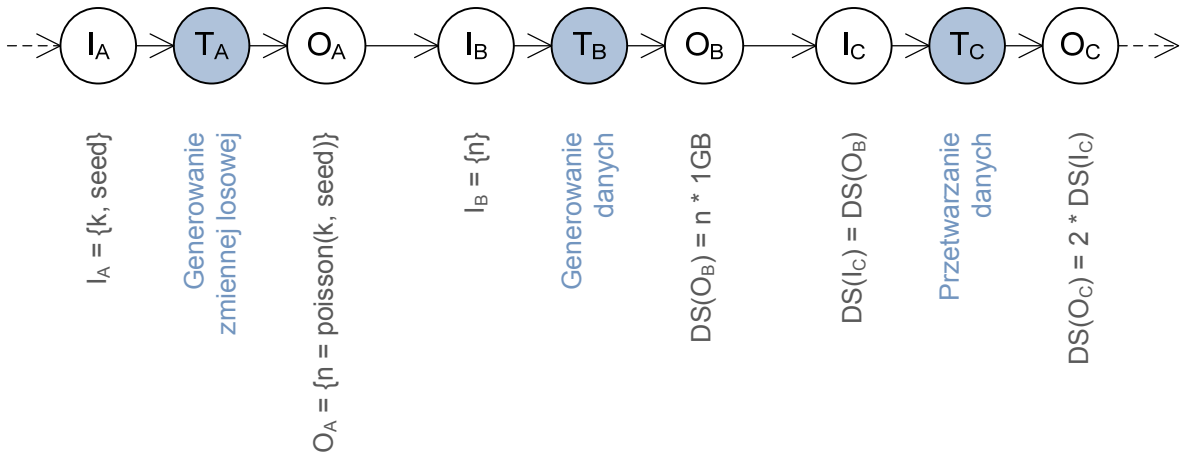
W niniejszej pracy przewidziano możliwość rozszerzenia modelu DFA-Static: w przypadku braku wartości parametru OP, wierzchołek T jest *zadaniem hierarchicznym*, a jego działanie opisane jest za pomocą podgrafu DFA-Static. Problem zadań hierarchicznych zostanie szerzej przedyskutowany w dalszej części pracy.

3.2. Wartości nieokreślone parametrów

W poprzednim rozdziale opisano parametry Statycznego Modelu DFA (klasy zadania), mające na celu taką charakterystykę problemu, aby możliwa była automatyczna alokacja procesów będących instancją modelu oraz optymalizacja tej alokacji. W tym celu potrzebna jest znajomość wartości tych parametrów.

Specyfika wielu algorytmów używanych w Dynamicznych Analizach Finansowych pozwala na ścisłe określenie wielkości parametrów jeszcze przed etapem wykonania. Przykładem może być zadanie dodające dwa wektory: Z = X + Y. Znając wielkość N danych wejściowych reprezentujących składniki operacji dodawania – $DS(I_X) = N$, $DS(I_Y) = N$ – można określić rozmiar zbioru danych reprezentującego wektor wyjściowy: $DS(O_Z) = N$. Znajomość tych wielkości pozwala na podjęcie decyzji dotyczącej alokacji procesu wykonującego opisaną operację na odpowiedni węzeł dysponujący wystarczającymi zasobami w celu przeprowadzenia obliczeń.

Proces decyzyjny opiera się na założeniu, że znana jest wielkość N, która jest wynikiem innych operacji o znanych parametrach. Gdy wielkość zbioru danych jest niemożliwa do określenia na danym etapie, oznaczana jest symbolem „?”. Jedną z przyczyn nieokreśloności parametrów jest istnienie zadania stochastycznie generującego dane lub czytającego wartości o nieznanym (na pewnym etapie) rozmiarze z bazy danych.



Rysunek 10. Fragment modelu ilustrujący problem nieokreśloności parametrów.

Rysunek 10 przedstawia fragment modelu DFA, dla którego podczas fazy projektowania możliwe jest określenie wartości parametrów tylko do pewnego miejsca, nawet pomimo znajomości a priori wartości pobieranych z bazy danych. Kolejność akcji jest następująca:

1) Dane wejściowe I_A zawierają dwie liczby naturalne *k* i *seed*. Wtedy $DS(I_A) = 2 * sizeOf(Int)$ (przykładowo dla maszyny 64-bitowej: 16 bajtów).

2) Zadanie T_A generuje liczbę naturalną *n* z rozkładem Poissona, której wartość jest niemożliwa do określenia na etapie analizy klasy problemu. Pomimo tego znana jest wielkość $DS(O_A) = sizeOf(Int)$.

3) $DS(I_B) = DS(O_A)$.

4) Zadanie T_B generuje dane wyjściowe, których wielkość nie zależy od rozmiaru danych wejściowych I_B, lecz od ich wartości. Dlatego też $DS(O_B) = ?$ (wartość nieokreślona).

5) $DS(I_C) = DS(O_B)$, czyli wartość $DS(I_C)$ nie jest również znana na etapie projektowania, lecz dopiero podczas wykonania procesu będącego instancją problemu. Po wyliczeniu $DS(O_B)$ możliwe staje się dalsze precyzyjne określenie parametrów.

6) Zadanie T_C przetwarza zbiór wejściowy, czego wynikiem jest zbiór wyjściowy O_C o rozmiarze $DS(O_C) = 2 * DS(I_C)$.

W powyższej analizie położono jedynie nacisk na parametry związane z rozmiarem danych wejściowych i wyjściowych. Dla uproszczenia pominięto bardzo ważne w praktyce parametry dotyczące zadań: PC (ang. Processor Consumption), MC (Memory Consumption) oraz ME (Memory Efficiency).

Przytoczony powyżej, stosunkowo prosty przykład, prezentuje klasyczny problem występujący w Dynamicznych Analizach Finansowych: niemożliwość określenia wartości parametrów modelu, aż do pewnego momentu na etapie wykonania. Fakt ten pociąga za sobą następujące konsekwencje:

(3.13) W ogólności niemożliwe jest szeregowanie statyczne (ang. pre-run-time scheduling) [62], będące sporządzanym z góry planem przydziału zadań do zasobów. Konieczne staje się zastosowanie szeregowania dynamicznego (ang. run-time scheduling) [63], w którym plan przydziału zadań do zasobów sporządzany jest na bieżąco w trakcie wykonania programu.

(3.14) Szeregowanie dynamiczne możliwe jest tylko w obszarze modelu, który w danej chwili posiada określone wszystkie parametry. Dla przykładu z Rysunku 10: znając rozmiar I_A jesteśmy w stanie określić parametry do wierzchołka I_B (włącznie). Znając wartość *n* generowaną przez T_B, możemy szacować dalsze parametry, ale w praktyce o parametrach wierzchołków T_B oraz O_B nie można nic powiedzieć, aż do czasu zakończenia działania T_B. W momencie, gdy znana jest wartość $DS(O_B)$ możliwe staje się wyznaczenie parametrów następników O_B oraz próba optymalizacji alokacji pewną ilość „kroków na przód” do miejsca (lub miejsc), gdzie napotkana zostanie ponownie „nieokreśloność”.

Podsumowując, pewne parametry modelu DFA-Static będącego klasą problemu, mogą być nieznane na etapie projektowania. Fakt ten wyklucza możliwość zastosowania szeregowania statycznego. Konieczne staje się użycie szeregowania dynamicznego w trakcie działania systemu, jednakże zakres optymalizacji alokacji zasobów jest ograniczony wierzchołkami, których określenie parametrów na daną chwilę jest niemożliwe.

3.3. Zadania hierarchiczne

Jak zostało wcześniej wspomniane, przewidziana została możliwość rozszerzenia modelu DFA-Static o *zadania hierarchiczne*. Rozszerzenie to przyjmuje, że wierzchołek etykietowany „T” może reprezentować operację atomową, jak również posiadać skomplikowaną strukturę, reprezentowaną przez podgraf DFA-Static.

Idea tworzenia grafowych struktur hierarchicznych opisana została w pracy [22] (Kotulski) oraz [64], natomiast efektywne algorytmy analizy oraz syntezy, służące do wyodrębniania komponentów oraz tworzenia grafu płaskiego (niehierarchicznego) zostały przedstawione

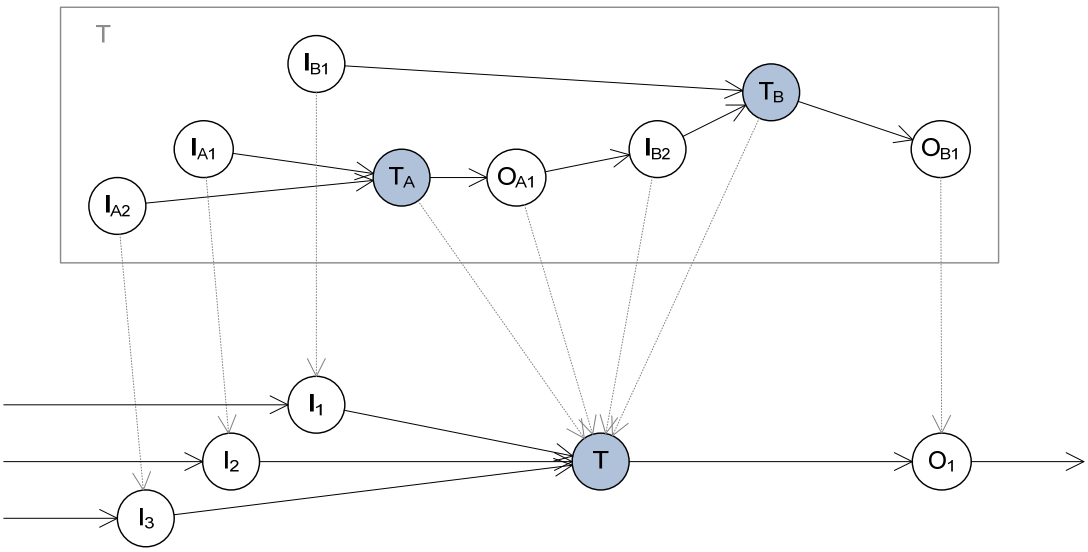
w [24] (Kotulski, Szpyrka). Na podstawie przytoczonej literatury założono, że istnieje możliwość jednoznacznej transformacji z jednej postaci do drugiej.

W praktyce znaczenie pojęcia „operacja atomowa” różni się w zależności od poziomu szczegółowości, na którym prowadzone są rozważania. Na potrzeby niniejszej pracy za operację atomową przyjęto funkcję dostępną w języku programowania wysokiego poziomu (na przykład dla języków C++, C#, Pascal, Java: +, *, log, sin, etc).

W przypadku *zadania hierarchicznego*, którego przykład przedstawiono na Rysunku 11, dane wejściowe oraz wyjściowe „T” mapowane są na odpowiednie dane wejściowe / wyjściowe podmodelu, natomiast „T” wskazuje wszystkie pozostałe elementy podzadania.

Zadania hierarchiczne (zwane zamiennie w niniejszej pracy *komponentami*) zdefiniowane zostały jako rozszerzenie, aby nie wprowadzać nadmiernej komplikacji opisu zależności wierzchołków modelu DFA-Static. Formalnie, dla zadanego wierzchołka T, zbiorów danych wejściowych { I_m } oraz wyjściowych { O_n } tego zadania przyjęto:

- struktura komponentu zadania T jest modelem DFA-Static, z którego usunięte zostały wierzchołki IS (Initial State) oraz FS (Final State),
- do każdego wierzchołka { I_m } istnieje krawędź etykietowana „c”, z dokładnie jednego wierzchołka „I” podmodelu, nie posiadającego poprzednika (Rysunek 11, wierzchołki I_{A1}, I_{A2}, I_{B1}),
- do każdego wierzchołka { O_m } istnieje krawędź etykietowana „c”, z dokładnie jednego wierzchołka „O” podmodelu, nie posiadającego następnika (Rysunek 11, wierzchołek O_{B1}),
- pozostałe wierzchołki podmodelu połączone są krawędzią etykietowaną „c” z wierzchołkiem „T”.



Rysunek 11. Przykładowe zadane hierarchiczne

(krawędzie etykietowane „c” zostały oznaczone linią przerywaną).

Parametry komponentu zależą od jego elementów składowych i przyjmują wartości nieokreślone, gdy przynajmniej jeden z elementów posiada wartość nieokreśloną. Schemat precyzyjnego określania parametrów wykracza poza zakres niniejszej rozprawy; nieformalnie: Processor Consumption jest maksymalną ścieżką obliczeń (sumując PC zadań), natomiast Memory Consumption jest maksymalnym przekrojem (sumując MC wierzchołków „I" oraz „O").

Wprowadzenie hierarchicznej struktury komponentów do modelu DFA-Static ma na celu możliwość zmiany jego ziarnistości poprzez zmniejszenie lub zwiększenie ilości przetwarzanych wierzchołków grafu przy założeniu, że wszystkie wierzchołki komponentu alokowane są na węzeł obliczeniowy, do którego przypisany został „T" (alokacja zadań oraz optymalizacja z użyciem komponentów zostanie szczegółowo opisana w dalszej części pracy).

Istnienie komponentów jest wielce pomocne także na etapie tworzenia modelu, dając projektantowi możliwość zdefiniowania złożonych struktur reprezentowanych przez pojedynczy wierzchołek, a tym samym zwiększając przejrzystość koncepcji.

3.4. Podsumowanie

W niniejszym rozdziale przedstawiona oraz omówiona została definicja modelu statycznego, pozwalającego w spójny sposób zapisać zadanie Dynamicznych Analiz Finansowych.

Obserwacje poczynione na etapie prac badawczych pozwoliły na dobór parametrów modelu zachowujący prostotę i klarowność opisu, przy jednoczesnym zachowaniu wystarczającej siły ekspresji. Znaczenie każdego z parametrów zostało szczegółowo przedyskutowane, wprowadzona i omówiona na przykładzie została „wartość nieokreślona" parametrów, która ma znaczący wpływ na planowanie alokacji zasobów, będące tematyką dalszej części niniejszej pracy.

Model DFA-Static bazujący na formalizmie grafów atrybutowanych opisuje w sposób spójny klasę problemu i może zostać poddany dalszej, w pełni zautomatyzowanej analizie, stanowiąc podstawę do kontroli alokacji zasobów.

4. Charakterystyka wybranych zagadnień DFA z użyciem modelu statycznego

W niniejszym rozdziale w ścisły sposób opisane zostaną zagadnienia oraz problemy związane z implementacją oraz działaniem procesów opartych na modelu DFA. W tym celu użyty zostanie uprzednio wprowadzony model DFA-Static, umożliwiający przedstawienie modelu Dynamicznych Analiz Finansowych w ścisły oraz spójny sposób. Wprowadzone oraz przedyskutowane zostaną także parametry charakteryzujące właściwości modelu, mające wpływ na wymagania środowiska wykonania oraz związaną z nimi szybkość przebiegu obliczeń.

Poruszane poniżej zagadnienia zostały zasygnalizowane w Rozdziale 1, opisującym DFA w języku naturalnym z pomocą notacji UML. Ze względu na brak formalizmu pozwalającego w pełni scharakteryzować problem, nie mogły być one przedstawione w sposób ścisły i jednolity, z pominięciem opisu w języku naturalnym.

Na bazie modelu DFA-Static wprowadzone zostaną także definicje, jak na przykład *niezależność zadań* i *niezależność grafów*, które używane będą w dalszej części pracy.

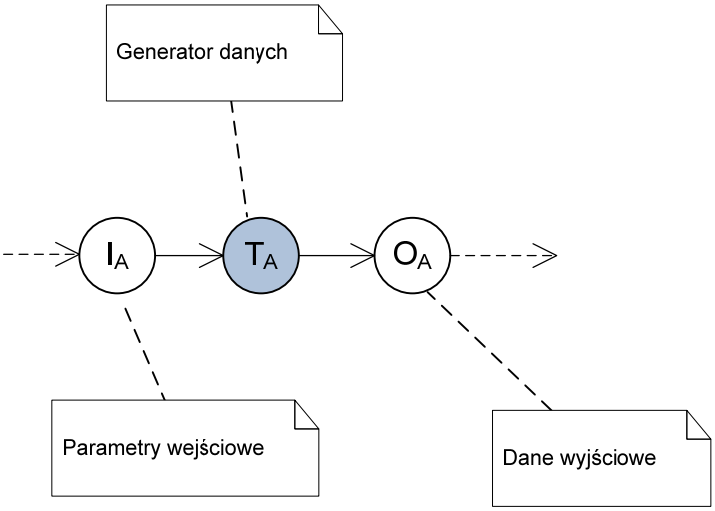
4.1. Stochastyczny sposób generowania danych wejściowych

Dane wejściowe w procesie Dynamicznych Analizach Finansowych generowane są głównie w sposób stochastyczny, co jest podstawą tej metody i pociąga za sobą daleko idące konsekwencje. Jak wspomniano wcześniej, jednym z powodów jest ograniczona wielkość zbioru danych historycznych będących przedmiotem badań. Bardzo często w takim przypadku stosowana jest metoda *bootstrap* lub jedna z jej odmian, polegająca na wielokrotnym losowaniu ze zwracaniem z zadanej próby [41] [42] [43] [44]. Pozwala ona na zwielokrotnienie stosunkowo ubogiego zbioru danych wejściowych przy zachowaniu parametrów statystyki z możliwym do oszacowania błędem. Innym rozwiązaniem jest produkcja danych na podstawie pewnego ustalonego modelu poprzez zadanie parametrów tego modelu. Przykładem może być losowanie liczb według rozkładu normalnego przy określonych parametrach: średniej oraz odchyleniu standardowym. Wygenerowane w ten sposób wartości przyjmowane są w dalszej części obliczeń jako rzeczywiste dane wejściowe systemu.

Zastosowanie modeli DFA wymusza konieczność powtarzalności eksperymentów. Ze względów prawnych istnieje nakaz przedstawienia przed organami kontroli wyników oddziaływania czynników oraz strategii na wypłacalność badanej firmy [4] [5]. Taki obowiązek dotyczy między innymi firm ubezpieczeniowych, w działalności których szacowanie ryzyka odgrywa kluczową rolę, a prawdopodobieństwo wystąpienia bankructwa jest ustawowo ograniczone. Takie ograniczenie wymusza użycie generatora liczb pseudolosowych, który gwarantuje powtarzalność ciągu losowanych liczb dla zadanego ziarna (a tym samym powtarzalność eksperymentu) i całkowicie wyklucza generatory liczb losowych [65] [66] [67].

Rysunek 12 przedstawia fragment statycznego modelu DFA będący generatorem zbioru danych pseudolosowych O_A. Zgodnie z przyjętymi założeniami spełniającymi wymogi legislacyjne [4] [5], dla ustalonego zbioru I_A, reprezentującego parametry wejściowe, zadanie T_A

za każdym razem będzie produkowało zbiór danych wyjściowych identyczny zarówno pod względem rozmiaru, jak i wartości. Ta własność ma niebagatelny wpływ na specyfikę obliczeń modeli DFA, czego konsekwencje opisane będą w dalszej części pracy.



Rysunek 12. Generator liczb pseudolosowych.

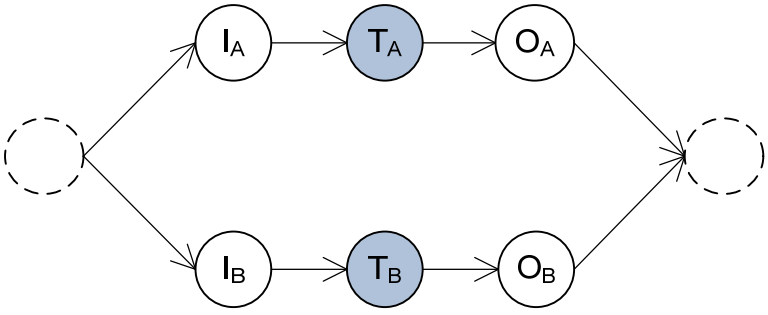
4.2. Niezależność zadań

Dla zachowania przejrzystości dalszych rozważań, na potrzeby niniejszej pracy wprowadzono pojęcie *niezależności zadań* oraz *niezależności podgrafów*.

Definicja 4.1

Zadania T_A i T_B są niezależne $\Leftrightarrow$ nie istnieje ścieżka w grafie DFA-Static łącząca wierzchołki reprezentujące zadania T_A i T_B.

■



Rysunek 13. Zadania niezależne T_A i T_B.

Na Rysunku 13 przedstawiono fragment statycznego modelu DFA, w którym zadania T_A oraz T_B są niezależne. Z niezależnością zadań wiąże się potencjalna możliwość równoległego ich przetworzenia w celu zmniejszenia czasu potrzebnego do zakończenia obliczeń. Fakt ten będzie szeroko dyskutowany w dalszej części rozprawy, przy poruszaniu tematów zarówno statycznego modelu Dynamicznych Analiz Finansowych, jak i problemów związanych z dynamiką procesów DFA.

Dodatkowo wprowadzona zostanie definicja niezależności podgrafów DFA-Static:

Definicja 4.2

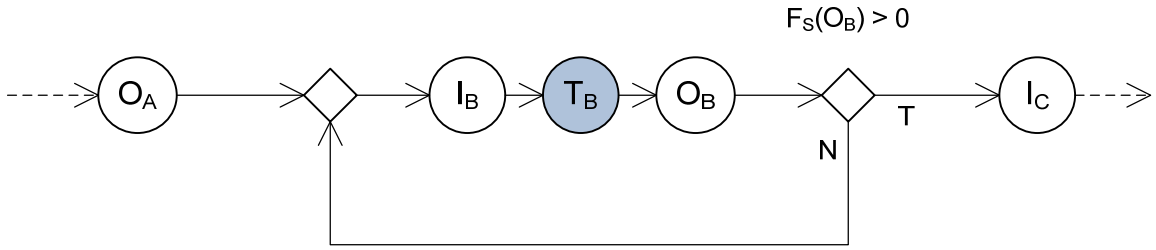
Podgrafy G_A oraz G_B grafu DFA-Static są niezależne ⇔ każde dwa wierzchołki reprezentujące zadania T_A i T_B takie, że $T_A \in G_A$ oraz $T_B \in G_B$ są niezależne.

■

Podobnie jak w przypadku wierzchołków, istnieje możliwość równoległego wykonania zbioru zadań należących do podgrafów niezależnych.

4.3. Efekt „eksplozji" danych

W Dynamicznych Analizach Finansowych obserwowany jest efekt „eksplozji" danych. Polega on na tym, iż wielkość danych wyjściowych komponentu lub wymagania pamięciowe komponentu są „dużo" większe, aniżeli dane wejściowe tego komponentu. Formalnie (na przykładzie fragmentu modelu z Rysunku 12): $I_A.DS \ll O_A.DS$. Jedną z przyczyn eksplozji danych, opisaną szczegółowo w poprzednim rozdziale, jest stochastyczny sposób generowania danych wejściowych.



Rysunek 14. Fragment modelu ilustrujący efekt eksplozji danych spowodowany rekurencją.

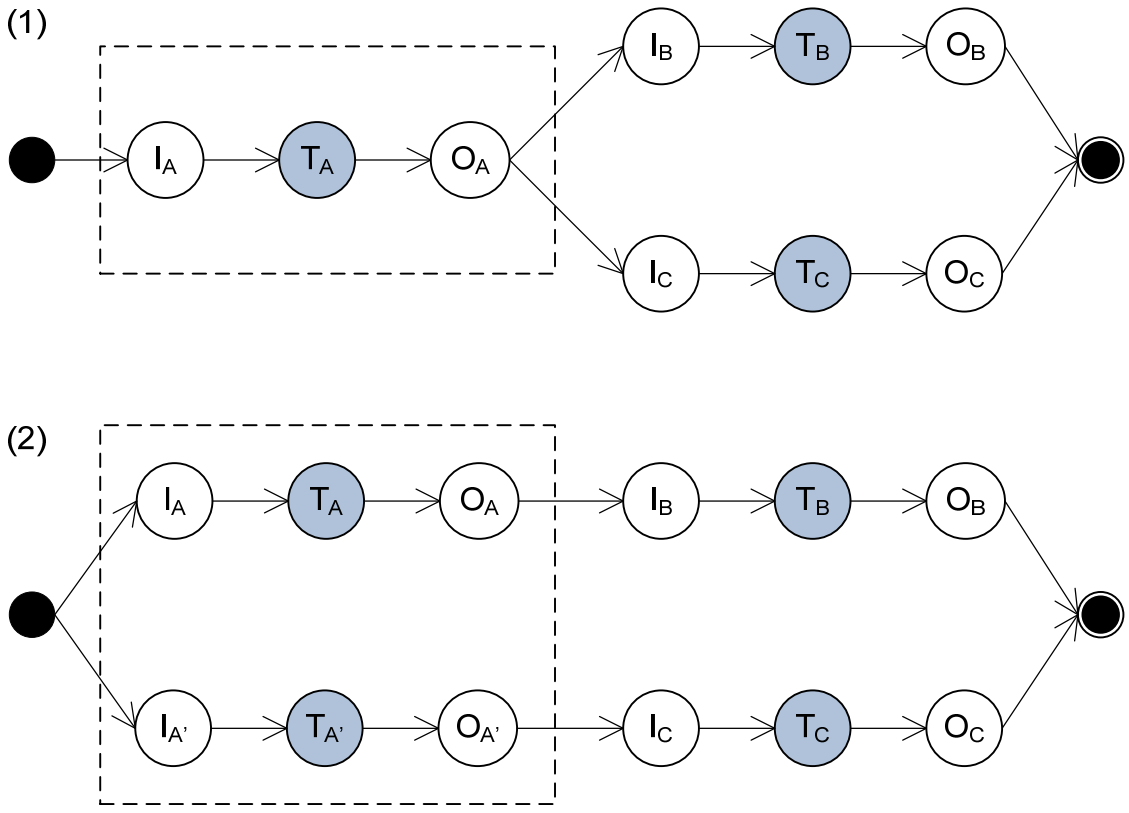
Efekt eksplozji danych występuje także bardzo często w przypadku algorytmów iteracyjnych, które w Dynamicznych Analizach Finansowych przetwarzają znaczne ilości danych. Na Rysunku 14 przedstawiono fragment pętli „while", której działanie może przedstawiać analizy na przestrzeni dziesiątek lat lub kwartałów. Tego typu algorytmy – reprezentowane przez wierzchołek T_B – wykorzystują dane ze wszystkich poprzednich okresów, co oznacza, że wielkość zbioru wejściowego I_B zwiększa się z każdą iteracją co najmniej w tempie liniowym, a w niektórych przypadkach w tempie wielomianowym stopnia większego niż jeden.

4.4. Replikacja danych oraz zadań

We wcześniejszych rozdziałach opisana została własność modelu DFA polegająca na powtarzalności obliczeń, w tym także powtarzalności generacji liczb pseudolosowych. Korzystając z tej własności, modele przedstawione na Rysunku 15 (1) oraz (2) są równoważne (produkują identyczne wyniki), gdy dane I_A i $I_{A'}$ oraz zadania T_A i $T_{A'}$ są takie same (zreplikowane).

Górna oraz dolna ścieżka (będąca podgrafem) w modelu (1) oraz (2) są niezależne, co daje potencjalną możliwość równoległego wykonania dwóch potoków obliczeniowych. W przypadku rozproszenia zadań koniecznością jest skopiowanie danych na inny węzeł obliczeniowy: dla modelu (1) są to dane reprezentowane przez wierzchołek I_B lub I_C. Przyjmując założenie $DS(I_A) << DS(O_A)$, w przypadku modelu (2) potencjalnie lepszym rozwiązaniem wydaje się sklonowanie niewielkiego zbioru I_A oraz uruchomienie zadania $T_{A'}$ generującego zbiór danych wyjściowych $O_{A'}$ równoważny O_A.

Szczegółowe rozważania dotyczące rozpraszania zadań oraz klonowania danych na inne węzły obliczeniowe przedstawione będą w dalszej części pracy. W niniejszym rozdziale zaznaczono jedynie możliwość modyfikacji modelu statycznego w sposób gwarantujący niezmienność wyników obliczeń.

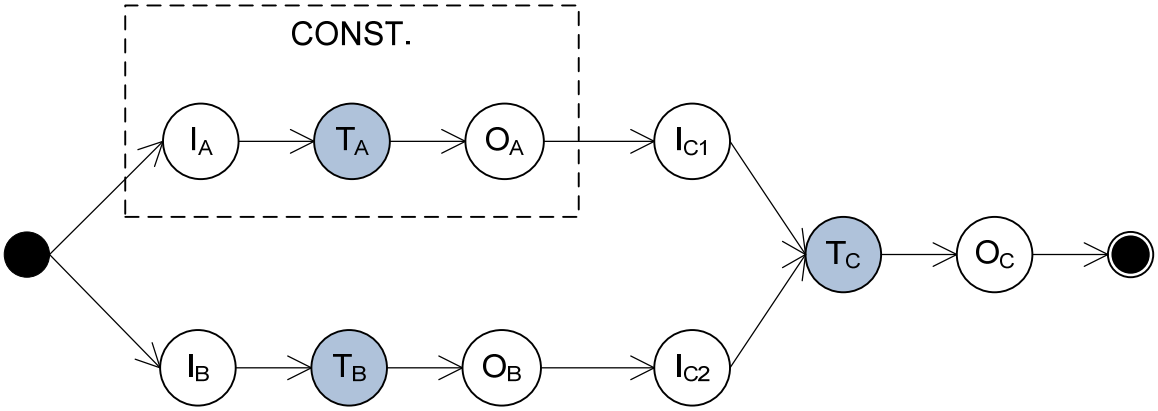


Rysunek 15. Przykład replikacji danych oraz zadań.

4.5. Wielokrotna ewaluacja niezmiennika modelu

Uruchamiane wielokrotnie modele DFA posiadają zazwyczaj fragment będący niezmiennikiem, produkującym podczas każdego uruchomienia identyczne wartości [7] [26] [8] [12]. Fragment taki może na przykład symulować środowisko ekonomiczne, w obszarze którego testowane są stopy zwrotu oraz ryzyko niewypłacalności modelowanego przedsiębiorstwa [33] [34].

Na Rysunku 16 przedstawiono model z zaznaczoną częścią stałą, składającą się z wierzchołków I_A, T_A oraz O_A. Informacja o istnieniu niezmiennika może zostać uzyskana poprzez obserwację kolejnych uruchomień lub zawarta na etapie konstruowania modelu.



Rysunek 16. Przykładowy model DFA z zaznaczoną częścią stałą.

Posiadając informację o fragmencie będącym niezmiennikiem, możliwe jest poprawienie wydajności procesu obliczeniowego w kolejnych iteracjach:

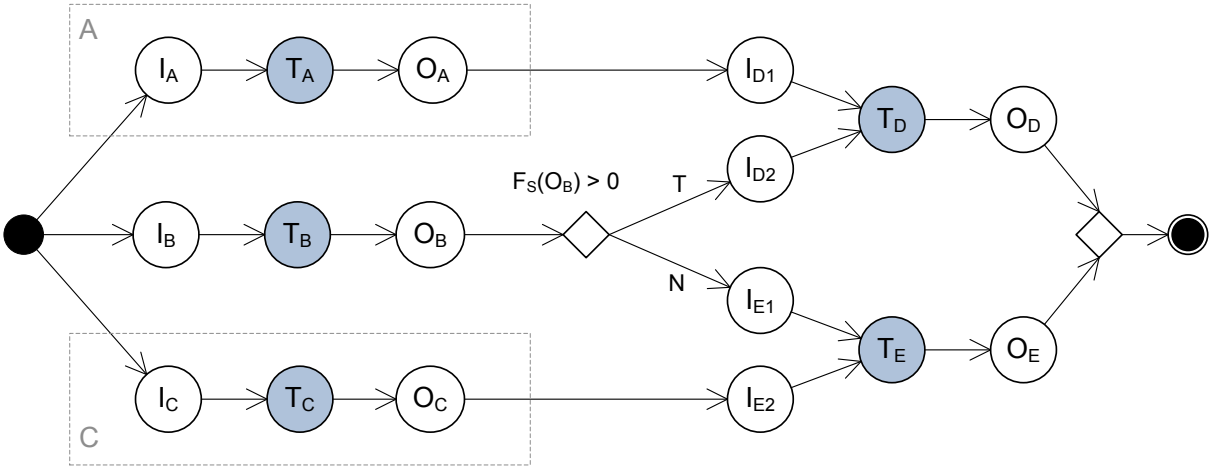
- dzięki aprioryczne znajomości wymagań obliczeniowych oraz pamięciowych możliwa jest statyczna alokacja niezmiennika,
- możliwe jest zapamiętanie i ponowne użycie wyników końcowych generowanych przez część stałą modelu bez konieczności powtarzania kosztownych obliczeń,
- możliwa jest jednokrotna replikacja wyników końcowych części stałej na wszystkie węzły obliczeniowe,
- możliwa jest replikacja warunków początkowych (dyskutowane wcześniej na podstawie Rysunku 15) na wszystkie węzły obliczeniowe i jednokrotna ewaluacja wartości niezmiennika.

4.6. Niedeterministyczny przebieg procesu ewaluacji

W rozdziale 2.2.3 zasygnalizowany został często pojawiający się problem warunkowej ewaluacji gałęzi zadań, jednakże brak formalnej notacji opisującej model DFA uniemożliwiał ścisłą charakterystykę zagadnienia, a w konsekwencji także automatyzację obliczeń.

Na Rysunku 17 przedstawiono przykładowy model, w którym po uzyskaniu danych wyjściowych O_B następuje proces decyzyjny, determinujący dalszą ścieżkę obliczeń (zawierającą T_D

albo T_E). W zależności od wyboru, dane generowane przez fragment modelu oznaczony A albo C zostaną niewykorzystane.

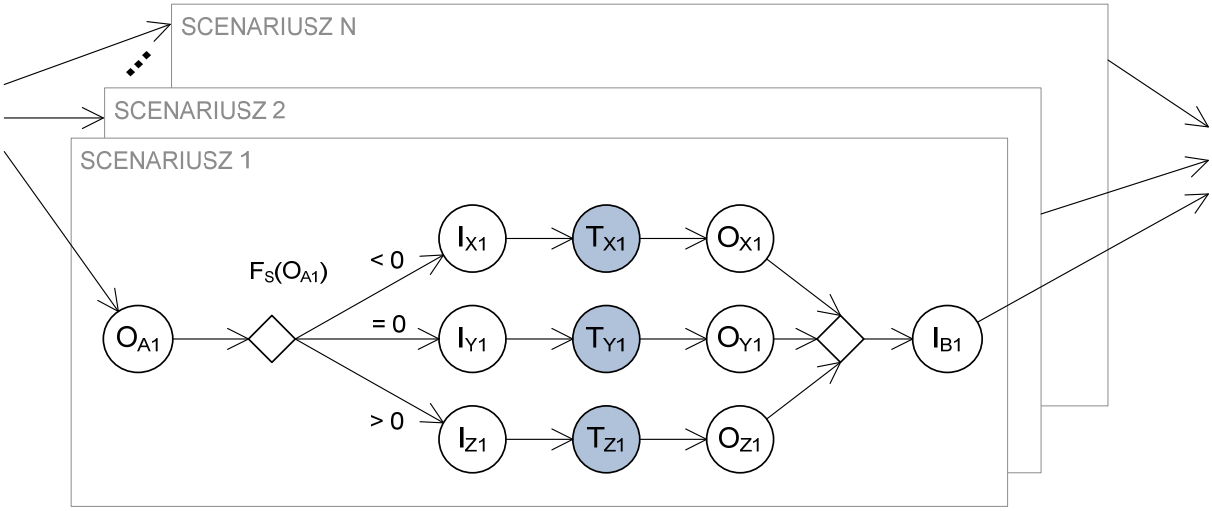


Rysunek 17. Warunkowa ewaluacja gałęzi obliczeń.

Analiza tego typu problemów pozwala na optymalizację wykonania procesów:

- w przypadku obliczeń na jednym procesorze uzyskanie wartości O_B w pierwszej kolejności, pozwala na zaniechanie ewaluacji fragmentu A lub C,
- istnieje potencjalna możliwość uruchomienia fragmentów A, C oraz zadania T_B równolegle na różnych procesorach, a następnie odrzucenie niepotrzebnych wyników.

4.7. Duża ilość przetwarzanych scenariuszy



Rysunek 18. Fragment modelu DFA składający się z N scenariuszy.

Obliczenia oparte na modelu DFA polegają na przetwarzaniu dużej liczby scenariuszy, co zostało szeroko przedyskutowane w całym Rozdziale 1, natomiast w Rozdziale 2.2.7 podano konkretny przykład przy użyciu języka UML.

Rysunek 18 przedstawia fragment modelu, w którym występuje N (N >> 1) niezależnych scenariuszy. Za pomocą formalizmu DFA-Static, możliwy jest ścisły opis zależności występujących pomiędzy wierzchołkami, jak na przykład:

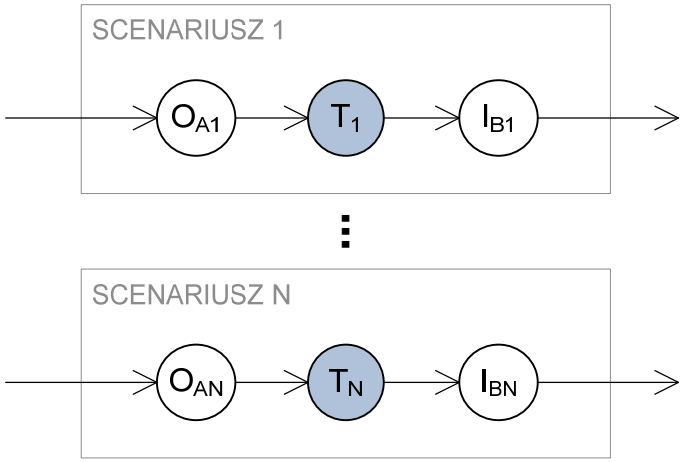
$$\forall i, j: \quad I_{Xi}.DS = I_{Xj}.DS, \quad T_{Xi}.OP = T_{Xj}.OP, \quad O_{Xi}.DS = O_{Xj}.DS \tag{4.1}$$

Taka charakterystyka poszczególnych elementów stanowi podstawę automatycznego szacowania *kosztu ewaluacji* (T_{EV}) oraz *kosztu transferu* (T_{TR}), w celu zbadania opłacalności przeprowadzenia obliczeń równoległych.

Komponentyzacja fragmentów przedstawiających scenariusze, pozwala na uproszczenie modelu poprzez redukcję liczby analizowanych wierzchołków. Rysunek 19 przedstawia scenariusze będące pojedynczym komponentem, o potencjalnie skomplikowanej strukturze wewnętrznej, posiadające własność:

$$\forall i, j: \quad T_i.OP = T_j.OP \tag{4.2}$$

W dalszej części pracy przeprowadzona zostanie dyskusja dotycząca wpływu redukcji liczby wierzchołków modelu na szybkość działania algorytmów optymalizacyjnych, a także na przyspieszenie zakończenia obliczeń. W przypadku tego typu analiz, przeprowadzanych w zautomatyzowany sposób, kluczowe znaczenie mają warunki typu (4.1) oraz (4.2), zdefiniowane przy pomocy modelu DFA-Static, co także zostanie omówione w dalszej części rozprawy.



Rysunek 19. Komponentyzacja scenariuszy.

4.8. Współdzielony dostęp do danych

Zbiory danych wejściowych oraz wyjściowych są danymi „tylko do odczytu". Oznacza to, że wygenerowany zbiór nie podlega modyfikacji i może jedynie zostać w całości usunięty i ewentualnie powtórnie utworzony. Własność ta umożliwia odczyt danych przez wiele procesów jednocześnie, bez konieczności wprowadzania mechanizmów synchronizacji.

Podobnie jak w przypadku omawianego wcześniej niezmiennika modelu, istnieje możliwość optymalizacji czasu wykonania procesu opartego na modelu DFA poprzez replikację zbiorów danych na inne węzły obliczeniowe. W praktyce bardzo często istnieje możliwość replikacji pokaźnych rozmiarów baz danych zawierających historyczne wskaźniki ekonomiczne, jeszcze przed etapem obliczeń właściwych, co przy wielokrotnej ewaluacji tego samego modelu znacznie podnosi wydajność obliczeń.

Istnieje także szereg prac, w których analizowane jest utrzymywanie danych w pamięci operacyjnej węzłów obliczeniowych, przy dostępie do których wykorzystywane są każdorazowo połączenia międzyprocesorowe [68] [69]. W wybranych przypadkach (szczególnie, gdy dostęp do danych ma charakter niesekwencyjny oraz sieć połączeń międzyprocesorowych jest wysokiej przepustowości i odporności na zakłócenia), rozwiązanie takie podnosi wydajność systemu, czego analiza możliwa jest na podstawie parametrów statycznego modelu DFA. W niniejszej rozprawie temat ten nie będzie szczegółowo omawiany; badania przydatności tego mechanizmu dla Dynamicznych Analiz Finansowych planowane są jednak jako temat przyszłych prac.

4.9. Proces tworzenia modelu ekonomicznego za pomocą DFA-Static

Wprowadzenie formalizmu grafowego ściśle opisującego DFA stało się przełomem tworzenia modeli ekonomicznych w środowisku biznesowym. Użycie notacji grafowej przyniosło następujące korzyści:

- porzucenie opisu w języku naturalnym na rzecz aparatu matematycznego umożliwiło ścisłe, precyzyjne i jednolite scharakteryzowanie problemów za pomocą wcześniej określonego zbioru parametrów,
- stworzone zostały narzędzia wizualne adresowane do aktuariuszy, służące do budowy modeli Dynamicznych Analiza Finansowych,
- łatwiejsze stało się śledzenie zmian oraz prowadzenie audytów ze względu na możliwość porównania grafów będących kolejnymi wersjami modelu,
- użycie komponentów umożliwiło zapanowanie nad rozległymi projektami złożonymi z dużej liczby zadań,
- komponentyzacja stworzyła także możliwość jednokrotnego zdefiniowania podzadań i wielokrotnego ich użycia w różnych miejscach w projekcie.

4.10. Podsumowanie

W niniejszym rozdziale opisane zostały zagadnienia związane z zadaniami Dynamicznych Analiz Finansowych, przy użyciu wcześniej zdefiniowanego formalizmu DFA-Static. Formalizm ten umożliwił ścisłą charakterystykę poszczególnych problemów zasygnalizowanych w Rozdziale 1, a także opisanie wielkości charakteryzujących zadania (takich jak na przykład rozmiar przetwarzanych zbiorów danych, złożoność obliczeniowa problemów) za pomocą zaproponowanych parametrów.

W ścisły sposób wyjaśniony został problem *eksplozji danych* oraz schemat stochastycznego sposobu generowania danych na bazie liczb pseudolosowych, spełniającego wspomniane wymogi legislacyjne.

Model DFA-Static pozwolił na wprowadzenie ścisłych definicji pewnych zjawisk (na przykład niezależności zadań), zwiększających przejrzystość analiz w kolejnych rozdziałach.

Zapisanie problemów za pomocą wprowadzonego modelu statycznego ułatwiło dokonanie obserwacji cech charakterystycznych DFA, które posłużyły w dalszej części rozprawy do sformułowania założeń optymalizacyjnych. Omówiona została możliwość replikacji danych oraz zadań, zjawisko występowania „niezmiennika" w obliczeniach, a także przetwarzanie scenariuszy.

Potwierdzone zostało osiągnięcie założonego celu wprowadzenia modelu statycznego: możliwe stało się opisanie zagadnień Dynamicznych Analiz Finansowych w sposób ścisły i całościowy, a tym samym rezygnacja z fragmentarycznych opisów w języku naturalnym, wspomaganych diagramami UML.

5. Kontrola alokacji zasobów za pomocą transformacji grafowych

W poprzednich rozdziałach zaprezentowany został statyczny model DFA, pozwalający w jednolity oraz precyzyjny sposób przedstawić klasę problemu ekonomicznego. Dzięki odstąpieniu od klasycznego formalizmu jakim jest UML na rzecz DFA-Static, wprowadzono możliwość ścisłej definicji elementów modelu, zależności między nimi oraz ich właściwości.

Statyczny model DFA przedstawiony w poprzednich rozdziałach niniejszej pracy jest definicją klasy problemu, dla której poszukiwane jest rozwiązanie. Kolejnym zagadnieniem jest przedstawienie w spójny oraz możliwy do zaimplementowania sposób przebiegu obliczeń przy uwzględnieniu faktu, iż system komputerowy wykorzystuje środowisko rozproszone i składa się z tysięcy komponentów, reprezentujących zarówno sprzęt, jak i procesy. W środowisku tym zauważalne są bardzo dynamiczne zmiany, dotyczące zarówno warstwy sprzętowej, jak i programowej. Reprezentacja oparta na ścisłym formalizmie powinna dawać możliwość automatyzacji procesu alokacji tak, aby wykluczyć konieczność udziału operatora, co dotychczas nie było możliwe.

Ponownie, formalizmem w naturalny sposób przedstawiającym powyżej scharakteryzowane środowisko sprzętowo-programowe są grafy, natomiast transformacje grafowe wydają się być najodpowiedniejszym formalnym narzędziem pozwalającym na opisanie dynamiki zachodzących w nim zmian [56] [57] [17] [58] [59] [21].

W niniejszym rozdziale zdefiniowany zostanie Dynamiczny Model DFA opisujący, a zarazem sterujący procesem mającym na celu ewaluację wcześniej zdefiniowanego problemu ekonomicznego. Model ten łączy w sobie informację zawartą w modelu statycznym, grafową reprezentację bieżącego stanu środowiska wykonania (konfigurację sprzętową, a także lokalizację danych i procesów) wraz z historią zmian, algorytmami sterującymi przebiegiem obliczeń oraz modułem synchronizującym graf stanu ze środowiskiem wykonania.

5.1. Wymagania dotyczące dynamicznego modelu DFA

W poprzednich rozdziałach wprowadzony oraz omówiony został Statyczny Model DFA, reprezentujący klasę problemu ekonomicznego. Jak zostało również wspomniane, zadania definiujące Dynamiczne Analizy Finansowe często przyjmują bardzo pokaźne rozmiary i wysoki stopień skomplikowania. Badania przeprowadzone przez autora przy użyciu wprowadzonego formalizmu pozwoliły na określenie estymowanej liczby wierzchołków grafów DFA-Static reprezentujących istniejące modele, urastającej do rzędu setek tysięcy, a nawet milionów. Z tego powodu przedstawienie dynamiki oraz kontrola takiego systemu w sposób, aby możliwa była jego implementacja w praktyce jest zadaniem skomplikowanym.

Przeprowadzone badania, uwzględniające zarówno wymagania dotyczące implementacji, jak i wdrożenia w środowisku produkcyjnym, pozwoliły na ustalenie podstawowych wymagań, które musi spełniać model przedstawiający przebieg ewaluacji obliczeń Dynamicznych Analiz Finansowych. Należą do nich:

(5.1) Całościowe oraz spójne przedstawienie bieżącego stanu oraz przebiegu obliczeń,

W dotychczasowych pracach oraz w praktyce problemy Dynamicznych Analiz Finansowych przedstawiane były fragmentarycznie, a koncentracja uwagi skupiona była na rozwiązaniu określonego podproblemu, będącego elementem całości zagadnienia [70] [6] [7] [9] [10] [11] [29] [44] [71] [72] [73]. Definicje poszczególnych etapów różniły się od siebie wykorzystaniem różnego rodzaju środków opisu bazujących głównie na diagramach UML. Bardzo często modele zawierały objaśnienia słowne wynikające z niewystarczającej zdolności opisowej użytych formalizmów [41] [42] [14] [43].

(5.2) Automatyzacja procesu alokacji zadań,

Niespójność opisu poszczególnych fragmentów modelu DFA była podstawową przyczyną niemożliwości automatyzacji alokacji zadań w celu przeprowadzenia obliczeń. Ewaluacja modelu odbywała się etapami, których synchronizacja wymagała udziału operatora. Z powodu wzrostu rozmiaru i skomplikowania modeli taki stan rzeczy jest nie do przyjęcia lub znacznie obniża wartość produktu w środowisku biznesowym. Z tego powodu istnieje duża presja na przeprowadzenie obliczeń w całości, bez nadzoru, a w szczególnych przypadkach zapewnienie także odporności na częściowe awarie sprzętowe.

(5.3) Optymalizacja alokacji zadań,

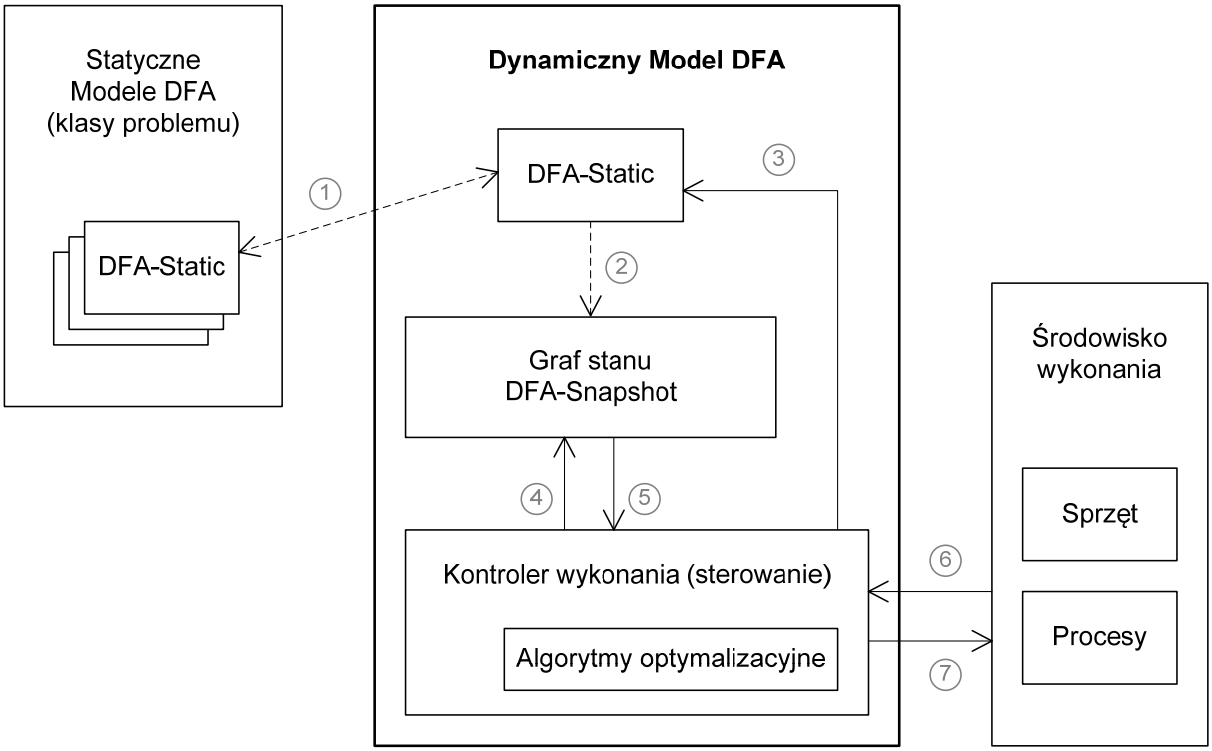
Naturalnym kryterium optymalizacji procesu alokacji jest minimalizacja czasu wykonania obliczeń. Niespójne przedstawienie oraz brak możliwości automatyzacji procesu obliczeniowego w znacznym stopniu utrudniało wprowadzenie optymalizacji globalnej, która była przede wszystkim optymalizacją statyczną dla fragmentów modelu oraz działaniem operatora sterującego poszczególnymi etapami obliczeń. Warto podkreślić, iż w związku z dużą konkurencją w środowisku przemysłowym, przyspieszenie czasu zakończenia obliczeń ma niebagatelny wpływ na jakość oferowanych usług.

(5.4) Wprowadzenie możliwości polepszenia alokacji w kolejnych iteracjach poprzez utrzymywanie historii oraz analizę poprzednich wykonań.

Automatyzacja procesu obliczeniowego oraz holistyczny i pełny opis problemu stwarza możliwość zapamiętania przebiegu poprzednich wykonań instancji modelu w celu przeprowadzenia analizy poprawiającej jakość alokacji. Poprawa optymalizacji odbywać się może na podstawie zarejestrowanych wymagań pamięciowych oraz czasowych, zarówno dla deterministycznych części modelu, jak i stochastycznych (w przypadku których przeprowadzana jest estymacja z pewnym błędem, określona na podstawie wcześniejszych wykonań).

W dotychczasowych pracach nie przedstawione zostały rozwiązania spełniające wyżej opisane wymagania. Z tego powodu wprowadzono model spełniający wymienione warunki, który w ścisły sposób definiuje oraz pozwala kontrolować proces ewaluacji modelu Dynamicznych Analiz Finansowych.

5.2. Elementy składowe dynamicznego modelu DFA



Rysunek 20. Schemat Dynamicznego Modelu DFA.

Schemat Dynamicznego Modelu DFA przedstawiony został na Rysunku 20. Zawiera on główne części składowe oraz przepływ informacji pomiędzy nimi, jak również określa powiązania z biblioteką modeli statycznych oraz środowiskiem wykonania. Poniżej objaśnione zostało znaczenie poszczególnych elementów, których szczegóły dyskutowane będą w dalszej części pracy (w nawiasach podano numery identyfikujące komunikację):

- DFA-Static – jest kopią klasy problemów, zadaniem będącym celem obliczeń, które pobierane jest (1) z repozytorium. Podczas procesu ewaluacji parametry modelu statycznego opisujące jego zachowanie (na przykład czas ewaluacji lub wymagania pamięciowe poszczególnych komponentów) mogą zostać zmodyfikowane. Uaktualniony model przekazywany jest (1) z powrotem do biblioteki, dzięki czemu algorytmy kontrolujące ponowne jego uruchomienie mogą wykorzystać informację uzyskaną podczas wcześniejszych iteracji, w celu lepszej alokacji podzadań.
- Graf stanu – w spójny sposób przedstawia środowisko sprzętowe w określonej chwili w czasie wraz ze statycznym modelem DFA; konstruowany jest na podstawie modelu DFA-Static pobranego z biblioteki (2) oraz środowiska sprzętowego (6). Graf stanu używany jest (5) do analizy problemu oraz podjęcia decyzji o przebiegu alokacji.
- Kontroler wykonania (zwany w dalszej części pracy *Kontrolerem*) – jest odpowiedzialny za sterowanie procesem obliczeniowym. Od strony sprzętowej: pobiera informacje dotyczące aktualnego stanu środowiska wykonania (sprzętu oraz procesów) (6) oraz steruje alokacją procesów (7). Uaktualnia (4) oraz analizuje (5) abstrakcyjną reprezentację środowiska

sprzętowego w celu podjęcia decyzji alokacji. *Kontroler* wykorzystuje algorytmy optymalizacyjne w celu polepszenia alokacji zadań, według założonego kryterium (na przykład minimalizacji czasu uzyskania wyników).

5.3. Graf stanu Dynamicznego Modelu DFA

Zadaniem modelu DFA-Dynamic jest przedstawienie dynamiki procesu obliczeniowego oraz historię zmian zachodzących w środowisku wykonania. W tym celu wprowadzona została reprezentacja stanu systemu w pewnej chwili w czasie oraz mechanizm opisujący zmiany zachodzące pomiędzy tymi stanami. W niniejszym rozdziale przedstawiony zostanie graf stanu modelu dynamicznego.

Chwilowy stan systemu przedstawiony będzie za pomocą grafu DFA-Snapshot, który konstruowany jest na podstawie statycznego modelu DFA pobranego z repozytorium oraz elementów środowiska wykonania (węzły obliczeniowe, procesy, dane) istniejących w określonym momencie w czasie. DFA-Snapshot przedstawia instancje zadań i danych, oraz ich alokację na węzłach obliczeniowych.

Definicja 5.1

Chwilowy stan modelu DFA (DFA-Snapshot) to graf aEDG $H = (V, D, \Sigma, \Gamma, \delta)$, gdzie $\Sigma = NL \times (NA \rightarrow_n NV)$ oraz $\Gamma = EL \times (EA \rightarrow_e EV)$, dla którego ustalone zostały:

$NL = \{ IS, FS, I, O, T, S, P, A, F, D, DP, AD, N, Start, Stop \}$,

$NA = \{ PC, MC, ME, OP, DS, SF, OP, Name, History \}$,

$EL = \{ n \}$,

$EA = \{ NB \}$,

oraz wprowadzone zostały ograniczenia sąsiedztwa wierzchołków, w zależności od przypisanych etykiet ze zbioru NL, zawarte w Tabeli 6 (strona 48).

■

Poniżej przedstawiono objaśnienie znaczenia etykiet wierzchołków grafu DFA-Snapshot (znaczenie etykietowania wierzchołków podgrafu będącego Statycznym Modelem DFA pozostaje niezmienione w stosunku do opisu DFA-Static z Rozdziału 3.1). Na potrzeby grafu DFA-Snapshot wprowadzone zostały dodatkowe etykiety: „P”, „A”, „F”, „D”, „DP”, „AD”, „N”, „Start”, „Stop”, „n”.

Znaczenie etykiet wierzchołków grafu DFA-Snapshot przedstawiono poniżej:

- IS – stan początkowy (ang. Initial State)
- FS – stan końcowy (ang. Final State)
- I – dane wejściowe (ang. Input Data)
- O – dane wyjściowe (ang. Output Data)
- T – zadanie (ang. Task)
- S – selekcja (ang. Selection)

- Start – start obliczeń
- Stop – zakończenie obliczeń
- P – proces pasywny (ang. Passive Process)
- A – proces aktywny (ang. Active Process)
- F – proces zakończony (ang. Finalized Process)
- D – dane fizyczne (ang. Data)
- DP – zalążek danych (ang. Data Placeholder)
- AD – dane archiwalne (ang. Archived Data)
- N – węzeł obliczeniowy (ang. Processing Node)
- n – połączenie węzłów obliczeniowych (ang. node connection)

W Tabeli 4 zawarte zostały atrybuty związane z poszczególnymi etykietami opisujące parametry elementów reprezentowanych przez wierzchołki grafu DFA-Snapshot (dla zachowania ścisłości oraz przejrzystości ponownie zawarto atrybuty wierzchołków o etykietach występujących także w grafie DFA-Static, których znaczenie nie uległo zmianie).

Etykieta	Atrybut	Opis atrybutu	Jednostka
T	PC	Processor Consumption	operacja
T	MC	Memory Consumption	B
T	ME	Memory Efficiency	-
T	OP	Operation	-
I	DS	Data Size	B
O	DS	Data Size	B
S	SF	Selection Function	-
S	OP	Operation	-
D	DS	Data Size	B
DP	DS	Data Size	B
DP	TTC	Time to Consume	operacja
DP	BTR	Blocked Transformation Rule	-
AD	DS	Data Size	B
N	CP	Computing Power	operacja/s
N	AM	Available Memory	B
n	NB	Network Bandwidth	B/s
A	TTC	Time to Consume	operacja
A	BTR	Blocked Transformation Rule	-

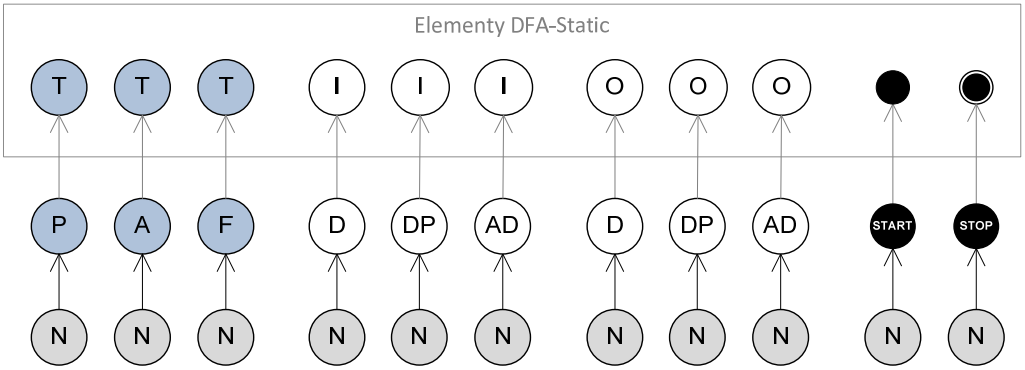
Tabela 4. Atrybuty wierzchołków grafu DFA-Snapshot.

Podobnie jak w przypadku modelu DFA-Static, wierzchołkom o różnych etykietach została nadana odmienna reprezentacja graficzna, przedstawiona w Tabeli 5.

P Passive Process	A Active Process	F Finalized Process
D Data	DP Data Placeholder	AD Archived Data
START Start	STOP Stop	N Node
T Task	I Input Data	O Output Data
Initial State	Final State	Selection

Tabela 5. Reprezentacje graficzne wierzchołków grafu DFA-Snapshot.

Dla zachowania przejrzystości, na Rysunku 21 przedstawiono relację pomiędzy elementami pochodzącymi z DFA-Static (będącymi klasami zadań oraz danych), a instancjami tych klas (wierzchołki etykietowane „P”, „A”, „F”, „D”, „DP”, „AD”, „Start”, „Stop”) zaalokowanymi na węzłach obliczeniowych „N”.



Rysunek 21. Relacje wierzchołków reprezentujących elementy środowiska wykonania z elementami pochodzącymi od grafu DFA-Static.

W Tabeli 6 zawarto dopuszczalne etykietowanie sąsiadów wierzchołka o zadanej etykiecie w grafie DFA-Snapshot.

Etykieta poprzednika	Etykieta wierzchołka	Etykieta następnika
I, P, A, F	T	O
IS, O, S, D, DP, AD	I	T
T, D, DP, AD	O	FS, I, S
O, S	S	I, S, FS
Start	IS	I
O, S, Stop	FS	-
N	P	T
N	A	T
N	F	T
N	D	I, O
N	DP	I, O
N	AD	I, O
N	N	N, P, A, F, D, DP, AD, Start, Stop
N	Start	IS
N	Stop	FS

Tabela 6. Dopuszczalne etykietowanie sąsiadów wierzchołków w grafie DFA-Snapshot.

Poniżej omówione zostaną funkcje wierzchołków o poszczególnych etykietach (pominięto opis znaczenia etykiet pochodzących z grafu DFA-Static, które pozostaje niezmienione).

5.3.1. Węzeł obliczeniowy (N)

Wierzchołki etykietowane „N” przedstawiają węzły obliczeniowe, na których alokowane mogą być procesy oraz dane. Atrybut „CP” (ang. Computing Power), określa moc obliczeniową węzła, natomiast „AM” (ang. Available Memory) rozmiar dostępnej pamięci.

Wierzchołki „N” mogą być połączone krawędziami etykietowanymi „n”, przedstawiającymi powiązania pomiędzy węzłami obliczeniowymi. Z każdą krawędzią „n” związany jest atrybut „NB” (ang. Network Bandwidth), charakteryzujący przepustowość łącza. Dla uproszczenia, na potrzeby niniejszej pracy, założono symetrię połączeń, lecz istnieje możliwość rozbudowy modelu o osobny opis przepustowości w każdą ze stron, a także wprowadzenia dodatkowych parametrów w dokładniejszy sposób modelujących środowisko wieloprocesorowe.

Wierzchołek „N” będący węzłem, na którym inicjalizowany jest proces obliczeniowy wskazuje na wierzchołek „Start”, który następnie wskazuje wierzchołek „IS” (pochodzący z modelu statycznego). Analogicznie, fakt zakończenia obliczeń sygnalizowany jest przez istnienie wierzchołka „Stop” pomiędzy wierzchołkiem „N”, a wierzchołkiem „FS”.

5.3.2. Proces pasywny (P) / Proces aktywny (A) / Proces zakończony (F)

Proces pasywny „P” zaalokowany na węźle „N” i połączony krawędzią z zadaniem „T”, wskazuje na potencjalną możliwość uruchomienia obliczeń. Możliwe jest wystąpienie wielu wierzchołków „P” przypisanych do różnych „N”, dla tego samego zadania „T”.

Wierzchołek „A” reprezentuje aktywny proces prowadzący obliczenia; podobnie jak w przypadku „P”, możliwe jest istnienie wielu procesów aktywnych dla tego samego zadania „T” na różnych węzłach „N”. Parametry TTC (ang. Time to Consume) oraz BTR (ang. Blocked Transformation Rule) służą do opisu opóźnień czasowych, sprecyzowanych w dalszej części pracy.

Zakończenie aktywności procesu oznaczane jest wierzchołkiem „F”, uruchamiającym tworzenie wierzchołków reprezentujących wygenerowane dane.

5.3.3. Dane fizyczne (D)

Wierzchołek etykietowany „D” reprezentuje instancję danych znajdującą się na węźle obliczeniowym „N”, z którym połączony jest krawędzią.

Każdy z wierzchołków „I” oraz „O” może być bezpośrednio połączony z wieloma wierzchołkami „D”, zaalokowanymi na różnych węzłach obliczeniowych „N” , co oznacza replikację tego samego zbioru danych na różne węzły środowiska.

Fizyczne dane reprezentowane przez „D”, są danymi „tylko-do-odczytu” (ang. read-only). Założenie to znacząco upraszcza rozważania dotyczące dostępu do danych przez wiele procesów jednocześnie, w szczególności eliminując klasyczny problem „czytelników i pisarzy”: pojawienie się wierzchołka „D” oznacza zakończony proces generowania danych i gwarantuje ich niezmienność.

5.3.4. Zalążek danych (DP)

Wierzchołek „DP” (Data Placeholder) jest wierzchołkiem pomocniczym używanym w procesie klonowania danych pomiędzy węzłami obliczeniowymi, sygnalizującym rozpoczęcie tego procesu. Parametry TTC (ang. Time to Consume) oraz BTR (ang. Blocked Transformation Rule) służą do opisu opóźnień czasowych, sprecyzowanych w dalszej części pracy.

Użycie wierzchołka „DP” zostanie omówione wraz z charakterystyką transformacji grafowych, odpowiedzialnych za replikację zbiorów danych.

5.3.5. Dane archiwalne (AD)

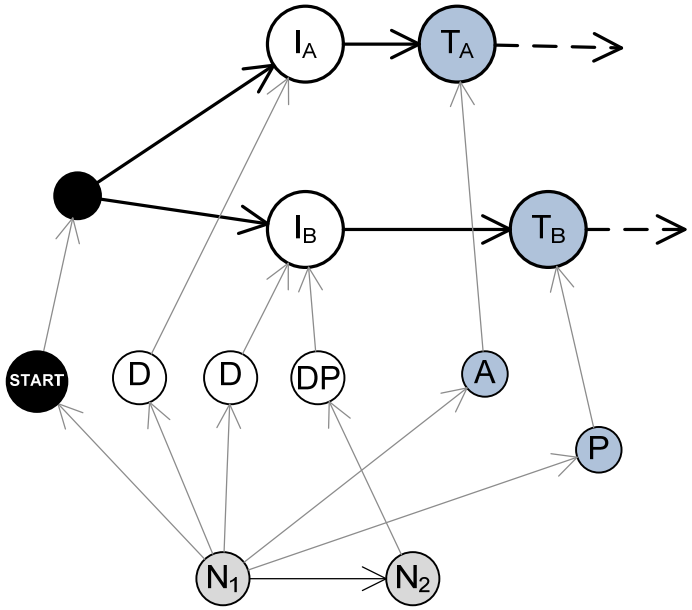
Wierzchołek „AD” (Archived Data) reprezentuje fizyczne dane, których przetwarzanie zostało zakończone.

Ze względu na możliwość wystąpienia pętli typu „while”, dane oznaczone „D” wykorzystane do obliczeń muszą zostać usunięte. Aby zachować informację dotyczącą alokacji danych, po zakończeniu procesu przetwarzania wierzchołki „D” zastępowane są przez „AD”, co zostanie szczegółowo omówione w dalszej części pracy poświęconej transformacjom grafowym.

5.3.6. Start oraz zakończenie obliczeń (Start / Stop)

Węzeł „Start” określa, na którym węźle środowiska inicjalizowany jest proces obliczeniowy. Podobnie węzeł „Stop” wyznacza, na którym węźle zakończony został proces ewaluacji.

5.3.7. Przykładowy graf DFA-Snapshot



Rysunek 22. Fragment przykładowego grafu DFA-Snapshot.

Na Rysunku 22 przedstawiono fragment przykładowego grafu DFA-Snapshot: w górnej części wierzchołki pochodzące z DFA-Static (IS, I_A, T_A, I_B, T_B), w dolnej węzły obliczeniowe (N_1, N_2), natomiast pomiędzy – instancje bytów określonych wierzchołkami modelu statycznego.

Znaczenie alokacji dla przykładowego grafu jest następujące: inicjalizacja obliczeń („Start”) nastąpiła na węźle N_1, na którym zaalokowane zostały instancje danych („D”) klas I_A oraz I_B. Na wierzchołku N_1 uruchomiony został proces („A”) klasy T_A, a także możliwe jest uruchomienie procesu („P”) klasy T_B. Na węzeł obliczeniowy N_2 prowadzona jest akcja klonowania danych („DP”) klasy I_B.

W dalszej części pracy zaprezentowany zostanie bardziej szczegółowy przykład, obejmujący zarówno dyskusję alokacji obiektów, jak i dynamikę zmian wykonania procesu DFA.

5.4. Dynamika zmian procesu obliczeniowego

Wprowadzony model DFA-Snapshot bazujący na formalizmie grafów atrybutowanych, pozwala na charakterystykę stanu obliczeń w pewnym momencie w czasie. Istnieje zatem potrzeba doboru mechanizmu opisującego transformację z jednego stanu w następny.

Gramatyki grafowe (bazujące na podejściu algorytmicznym) aedNLC (Kotulski) [61], czy ETPL(k) (Flasiński) [60] oferują bardzo efektywne algorytmy parsingu i problemu przynależności grafu do danej klasy obliczeń o złożoności $O(n^2)$, gdzie n jest liczbą wierzchołków w grafie. Osiągnięta efektywność wynika z pewnych ograniczeń nałożonych na gramatykę, z których część ma jedynie charakter techniczny, jak na przykład liniowe uporządkowanie wierzchołków w grafie. Ograniczenie polegające na tym, że graf występujący po lewej stronie produkcji powinien być jednoelementowy, pociąga za sobą konsekwencje powodujące zmniejszenie czytelności stosowanych

produkcji. Transformacja z grafu opisującego jeden stan systemu, w graf opisujący stan kolejny, wymaga uruchomienia sekwencji produkcji.

Dla zachowania przejrzystości zdecydowano się na użycie mechanizmu bazującego na podejściu algebraicznym w oparciu o koncepcję Single Push-Out (SPO) [18]. Produkcje w tym przypadku dopuszczają występowanie grafu o rozmiarze większym niż jeden (przyjmijmy rozmiar *k*), jednak istnieje wtedy $\binom{n}{k}$ potencjalnych mapowań na graf modyfikowany o rozmiarze *n*, co w ogólności jest rozwiązaniem nieefektywnym w praktyce.

Dla produkcji wprowadzonych w dalszej części niniejszej pracy udało się wyznaczyć co najwyżej dwa wierzchołki charakterystyczne – z tego względu proces mapowania odbywa się ze złożonością $O(n^2)$, gdzie *n* jest liczbą wierzchołków w modyfikowanym grafie (dowód złożoności przedstawiony zostanie w Rozdziale 6.4).

Produkcje typu SPO zdefiniowane są jako częściowy morfizm p: L $\rightarrow$ R, gdzie L, R $\in$ DFA-Snapshot. Założono, że wierzchołki w grafach L i R, posiadające te same indeksy (na rysunkach produkcji oznaczone jako v_i), reprezentują te same wierzchołki. Zastosowanie produkcji *p* na grafie G $\in$DFA-Snapshot jest interpretowane w następujący sposób:

1) Znajdowane jest wystąpienie grafu L w grafie G (w niniejszej pracy przyjmujemy, że odwzorowanie L na G jest izomorfizmem identycznościowym na poziomie struktury grafu i etykietowania),
2) Wierzchołki występujące zarówno w L oraz w R pozostają niezmienione,
3) Wierzchołki v $\in$ V(L) \ V(R) zostają usunięte z grafu G wraz z krawędziami przyległymi,
4) Wierzchołki v $\in$ V(R) \ V(L) zostają dodane do grafu G (z nadaniem unikalnych w ramach G indeksów) wraz z krawędziami łączącymi je z wierzchołkami określonymi w punkcie 2).

Na potrzeby modelu DFA-Dynamic wprowadzono *transformację rozszerzoną*:

Definicja 5.2

Transformacja rozszerzona (ang. Extended Transformation) to czwórka T_{EX} = (SPO, PR, EC, TC), gdzie:

SPO – jest transformacją grafową typu Single Pushout,

PR $\in$ $\mathbb{N}$ (ang. Priority) jest priorytetem transformacji,

EC – (ang. Execution Condition) to funkcja $f_C(H) \in \{ 0, 1 \}$, gdzie H $\subseteq$ G $\in$ DFA-Snapshot,

TC – (ang. Time Condition) jest trójką (TTC, prod, v):

TTC $\in$ $\mathbb{R}$ (ang. Time to Consume),

prod $\in$ $\mathbb{N}$ jest numerem blokowanej produkcji,

v $\in$ V(G), G $\in$ DFA-Snapshot ■

Priorytet transformacji (PR) narzuca kolejność uruchomienia transformacji w przypadku, gdy możliwe jest odpalenie więcej niż jednej dla zadanego grafu: w pierwszej kolejności wybierane są te, o najniższej wartości priorytetu.

Execution condition (EC) jest dodatkowym warunkiem koniecznym, który musi zostać spełniony, aby możliwe było odpalenie transformacji. Warunek ten testuje podgraf izomorficzny z lewą stroną produkcji, dla którego ma zostać uruchomiona transformacja. Formalnie, do odpalenia produkcji p: L $\rightarrow$ R konieczne jest spełnienie warunku

$$f_C(H) = 1 \tag{5.5}$$

gdzie

$H = m(L) \subseteq G \in$ DFA-Snapshot,

L – lewa strona produkcji SPO,

m – ustalony izomorfizm.

Time Condition (TC) jest również warunkiem koniecznym uruchomienia transformacji, który zostanie szczegółowo omówiony w kolejnym rozdziale. Wprowadzenie TC jako osobnego kryterium umotywowane jest zachowaniem przejrzystości definiowanych produkcji poprzez wyodrębnienie czynnika czasowego oraz jego wpływu na działanie systemu.

Stan początkowy systemu reprezentowany jest przez graf G_0, natomiast zachodzące w nim zmiany przedstawiają kolejne produkcje. Stan systemu po odpaleniu ciągu (p_1, .., p_n) produkcji przedstawia graf G_n (5.6).

$$G_0 \rightarrow^{P1} G_1 \rightarrow^{P2} \ldots \rightarrow^{Pn} G_n \tag{5.6}$$

5.5. Czynnik czasowy w transformacjach grafowych

W poprzednim rozdziale wprowadzony został mechanizm przedstawiający kolejne stany modelu obliczeniowego Dynamicznych Analiz Finansowych za pomocą grafów oraz transformacji opisujących zmiany pomiędzy tymi stanami (5.6).

W praktyce zadania obliczeniowe związane są z upływem czasu. Dla potrzeb modelu DFA wyszczególniono dwie operacje związane z opóźnieniem:

- proces obliczeniowy przetwarzający dane,
- klonowanie zbioru danych z jednego węzła obliczeniowego na drugi.

Definicja 5.3

Globalna zmienna czasowa t $\in \mathbb{R}$, o wartości początkowej $t_0 = 0$ określa miejsce w czasie, w którym znajduje się obecnie system.

■

Definicja 5.4

Uporządkowana w czasie, dyskretna reprezentacja systemu to dwójka TDSR = ($(G_k, t_k)_{k=0..n}$, $(p_k)_{k=1..n}$), gdzie:

G_k – jest grafem DFA-Snapshot,

$t_k \in \mathbb{R}$ określa wartość zmiennej czasowej,

p_k – jest produkcją przekształcającą $G_{k-1} \rightarrow G_k$,

oraz przyjęte zostało założenie:

$$\forall\, i < j:\ t_i \leq t_j \tag{5.7}$$

Inną (naprzemiennie używaną) formą zapisu TDSR jest

$$(G_0, t_0) \rightarrow^{P1} (G_1, t_1) \rightarrow^{P2} \ldots \rightarrow^{Pn} (G_n, t_n) \tag{5.8}$$

gdzie ciąg (t_k) jest ciągiem niemalejącym – czyli spełniającym (5.7) – o następującej interpretacji: graf G_k przedstawia stan systemu

(5.9) $\forall k < n$: $\begin{cases} \text{w przedziale czasowym } [t_k, t_{k+1}), & t_k < t_{k+1} \\ \text{o czasie } t_k, & t_k = t_{k+1} \end{cases}$

(5.10) dla k = n: od czasu t_n do obecnego momentu.

■

W celu umożliwienia algorytmicznego wyznaczenia wartości t_k, wierzchołkom etykietwanym A (Active Process) oraz DP (Data Placeholder) przypisano atrybuty (Rozdział 5.3):

- TTC $\in \mathbb{R}$ (ang. Time to Consume) – czas potrzebny do zakończenia zadania,
- BTR $\in \mathbb{N}$ (ang. Blocked Transformation Rule) – numer blokowanej transformacji grafowej.

Nieformalnie, transformacja BTR jest blokowana dla zadanego wierzchołka do momentu, gdy na rzecz wykonywanego zadania przeznaczony zostanie sumaryczny czas o wielkości co najmniej TTC. Upływ czasu globalnego potrzebny dla realizacji określonego zadania o czasie wykonania TTC zależy od algorytmów szeregowania systemu operacyjnego. Dla uproszczenia przyjęto, że czas procesora będzie dzielony proporcjonalnie dla wszystkich aktywnych zadań do niego przydzielonych: w przypadku współbieżnie (na tym samym procesorze) wykonywanych procesów obliczeniowych A_1 oraz A_2, pewien przedział czasowy Δt czasu globalnego zostanie podzielony na dwie części $t_1 + t_2 = \Delta t$ $(t_1, t_2 \geq 0)$, które odpowiednio pomniejszą wartości TTC wierzchołków A w tym przedziale czasowym.

Formalnie, warunek czasowy Time Condition (TC), blokujący możliwość uruchomienia produkcji p: L $\rightarrow$ R o numerze id_p ma postać:

$$\exists\, v \in V(H):\ (\, v.label \in \{A, DP\}\,) \ \wedge\ (\, v.TTC > 0\,) \ \wedge\ (\, v.BTR = id_p\,) \tag{5.11}$$

gdzie

$H = m(L) \subseteq G \in$ DFA-Snapshot,

L – lewa strona produkcji SPO,

m – ustalony izomorfizm.

Poniżej zaprezentowany został Algorytm 5.1 generujący pary (G_k, t_k) dla czasu zadanego *globalną zmienną czasową* t, której wartość zostaje także uaktualniona.

Algorytm 5.1. Generowanie par (G_k, t_k) dla zadanego czasu globalnego t.

```
1)  Uruchom produkcje dla czasu t (formalnie: spełniające warunki EC (5.5) ∧ TC (5.11)),
    generujące kolejne pary (Gk, tk), o wartości tk = t
2)  timedVertices = { (v, v.TTC, loadFactor(v, G)): v ∈ V(G) ∧ v.label ∈ {A, DP} }
3)  Δt = min { ttc * lf, (v, ttc, lf) ∈ timedVertices }
4)  t = t + Δt
5)  foreach (v, ttc, lf) ∈ timedVertices
6)      v.TTC = v.TTC - Δt / lf

                          ⎧ #{ n→a ∈ E(G): n→v ∧ n.label = N ∧ a.label = A },   v.label = A
loadFactor(v, G) =        ⎨
                          ⎩ #{ w ∈ V(G): w.label = DP },                        v.label = DP
```

Zasada działania powyższego algorytmu jest następująca (dla zadanego globalnego czasu *t*):

W 1) uruchamiane są wszystkie możliwe transformacje produkujące kolejne pary (G_k, t_k), generujące stany systemu (kolejne grafy G_k) o czasie t_k = t.

Spostrzeżenie: w pierwszej kolejności uruchomione zostaną produkcje, których TTC zostało obniżone (TTC $\leq$ 0) w poprzedniej iteracji uruchomienia algorytmu. Dowód: w poprzedniej iteracji uruchomione zostały wszystkie możliwe produkcje, a następnie obniżone zostały wartości TTC.

Tworzony jest 2) zbiór trójek: wierzchołków A oraz DP wraz z pozostałym obecnie czasem potrzebnym do zakończenia zadania oraz współczynnikiem określającym, jaka część czasu zostanie przeznaczona na zadanie (charakterystyka funkcji *loadFactor* zostanie omówiona poniżej).

Następnie 3) określany jest przedział czasu globalnego Δt, po którym nastąpi najbliższe zakończenie zadania. Czas globalny jest zwiększany 4), a następnie 5) dla każdego wierzchołka ze zbioru *timedVertices* uaktualniany jest pozostały czas potrzebny do zakończenia, zależny od Δt oraz współczynnika określającego ilość przydzielonego czasu.

Analiza czasowa powyższego algorytmu nie jest możliwa na obecnym etapie rozprawy (gdyż nie zostały zdefiniowane transformacje grafowe oraz złożoność związana z warunkami EC oraz TC) i przedstawiona zostanie w Rozdziale 6.4.

Funkcja *loadFactor* (Algorytm 5.1) określa współczynnik przydzielania czasu w następujący sposób:

- dla wierzchołków A (Active Process): zakłada taki sam priorytet każdego zadania na tym samym procesorze; zwraca ilość wierzchołków A przypisanych do tego samego węzła obliczeniowego N,
- dla wierzchołków DP (Data Placeholder): zakłada topologię, w której występują symetryczne interferencje w przypadku transferu danych pomiędzy dowolnymi parami wierzchołków (na przykład topologia magistrali / szynowa); zwraca ilość odbywających się transferów.

Funkcja *loadFactor* jest jedynie propozycją ściśle określoną dla pewnego typu środowiska obliczeniowego. Istnieje możliwość jej modyfikacji, na przykład aby uwzględnić priorytety procesów oraz popularną obecnie w obszarze sieci LAN topologię gwiazdy rozszerzonej, lecz zagadnienie to wykracza poza ramy niniejszej rozprawy, będąc przedmiotem przyszłych prac.

5.6. Sterowanie procesem obliczeniowym – Kontroler

Kontroler jest elementem modelu DFA-Dynamic (Rysunek 20, strona 44), odpowiedzialnym za sterowanie procesem obliczeniowym: pobiera i wysyła informacje ze środowiska sprzętowego, utrzymuje graf stanu DFA-Snapshot (będący modelem tego środowiska) oraz historię dokonywanych w nim zmian, a także podejmuje decyzje dotyczące alokacji.

Definicja 5.5

Rozwiązanie problemu obliczeniowego DFA zadanego grafem pierwotnym G_0, polega na znalezieniu TDSR (uporządkowanej w czasie, dyskretnej reprezentacji) $(G_0, t_0) \rightarrow^{P1} (G_1, t_1) \rightarrow^{P2} \ldots \rightarrow^{Pn} (G_n, t_n)$, dla której graf G_n zawiera wierzchołek etykietowany „STOP”. ■

Definicja 5.6

Czas rozwiązania problemu obliczeniowego DFA o postaci $(G_0, t_0) \rightarrow^{P1} (G_1, t_1) \rightarrow^{P2} \ldots \rightarrow^{Pn} (G_n, t_n)$, to wartość t_n. ■

Definicja 5.7

Rozwiązanie X problemu obliczeniowego DFA jest efektywniejsze niż rozwiązanie Y, jeżeli:

$\exists$ rozwiązanie X $\wedge$ [($\exists$ rozwiązanie Y i czas rozwiązania X $<$ czas rozwiązania Y) $\vee$ $\neg\exists$ rozwiązanie Y]

■

Algorytm 5.2 przedstawia schemat działania *Kontrolera*, polegający na synchronizacji grafu DFA-Snapshot z modelowanym środowiskiem (poprzez odczytanie aktualnego stanu węzłów obliczeniowych oraz procesów, które podlegają zmianom w trakcie wykonywania obliczeń), podjęciu decyzji dotyczącej alokacji zadań na podstawie tego grafu oraz wysłaniu komend do środowiska wykonania (sterujących alokacją zadań obliczeniowych).

Decyzja dotycząca alokacji zadań podejmowana jest przez moduł *algorytmów optymalizacyjnych* (Rysunek 20, strona 44), który dla zadanego problemu DFA G_0 poszukuje *maksymalnie efektywnego* rozwiązania w postaci *uporządkowanej w czasie, dyskretnej reprezentacji* TDSR. Szczegółowy opis technik optymalizacji opracowanych specjalnie na potrzeby Dynamicznych Analiz Finansowych znajduje się w Rozdziale 1.

Algorytm 5.2. Schemat działania Kontrolera.

```
1) REPEAT
2)     Odczytaj stan środowiska wykonania (węzły obliczeniowe, procesy)
3)     Uaktualnij graf DFA-Snapshot modelujący środowisko
4)     Podejmij decyzje optymalizacyjne (wyznacz produkcje wymuszone przez Kontroler)
5)     Nanieś zmiany na graf DFA-Snapshot oraz uaktualnij zmienną czasową (Algorytm 5.1)
6)     Wyślij instrukcje sterujące środowiskiem wykonania (procesami)
7) UNTIL (osiągnięty stan końcowy obliczeń)
8) Uaktualnij statystyki DFA-Static
```

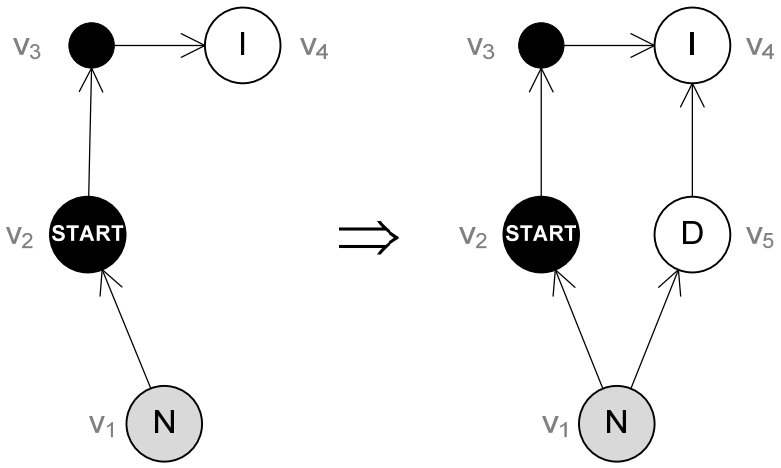
5.7. Transformacje grafowe opisujące dynamikę modelu DFA-Dynamic

W niniejszym rozdziale przedstawiona zostanie kompletna lista transformacji modelujących dynamikę wykonania procesu DFA. Następnie, zaprezentowany zostanie przebieg wykonania przykładowego procesu wraz z analizą czasową.

Dla każdej transformacji podano Priorytet, Opóźnienie (TC) oraz Warunek uruchomienia (EC). Zastosowano symbolikę:

- „TC = ∅” równoważne „TC = (0, 0, v*)”, gdzie v* jest dowolnym wierzchołkiem,
- „EC: brak” równoważny funkcji stałej $f_C(H) = 1$,
- „EC: kontroler” określającej jako warunek uruchomienia decyzję *Kontrolera*, podejmowaną w oparciu o działanie algorytmów optymalizacyjnych.

5.7.1. Transformacja – Inicjalizacja Obliczeń



Rysunek 23. Transformacja nr 1 – Inicjalizacja Obliczeń.

Inicjalizacja obliczeń polega na wygenerowaniu wierzchołków D reprezentujących dane wejściowe na arbitralnie wybranym węźle obliczeniowym N, połączonym krawędzią z (jedynym) wierzchołkiem etykietowanym START (Rysunek 23). Produkcja numer 1 uruchamiana jest dla każdego wierzchołka I (Input Data) połączonego z wierzchołkiem IS (Initial State).

- Priorytet: 0
- Opóźnienie: TC = ∅
- Warunek uruchomienia EC:

$$\neg \exists\, d \rightarrow i \in E(G):\ d.label = \text{"D"} \wedge i.label = \text{"I"} \tag{5.12}$$

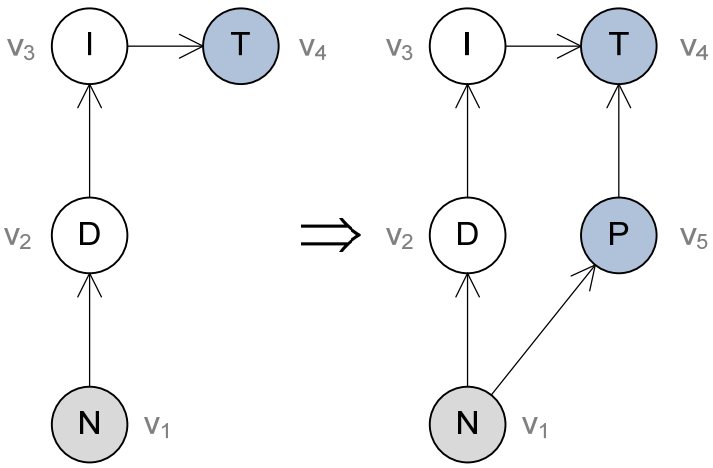
5.7.2. Transformacja – Tworzenie Stanu Pasywnego

Wierzchołek etykietowany P – Stan Pasywny – reprezentuje potencjalną możliwość uruchomienia zadania obliczeniowego klasy T, które występuje w momencie, gdy wszystkie zbiory danych wejściowych potrzebnych do uruchomienia tego zadania zostaną zgromadzone na tym samym węźle obliczeniowym N.

Utworzenie stanu pasywnego (Rysunek 24) jest sygnalizacją gotowości przejścia do stanu aktywnego (opisanego w dalszej części pracy), co może zostać przesunięte w czasie przez *Kontroler* lub nie nastąpić wcale, gdy podjęta zostanie decyzja dotycząca zmiany wierzchołka uruchomienia zadania. Dla jednego zadania T może występować wiele stanów pasywnych P przypisanych do różnych wierzchołków N. W zależności od decyzji podjętej przez *Kontroler*, uruchomione może zostać jedno lub więcej zadań równolegle, a dane wyjściowe zostaną pobrane od tego, które wcześniej zakończy pracę.

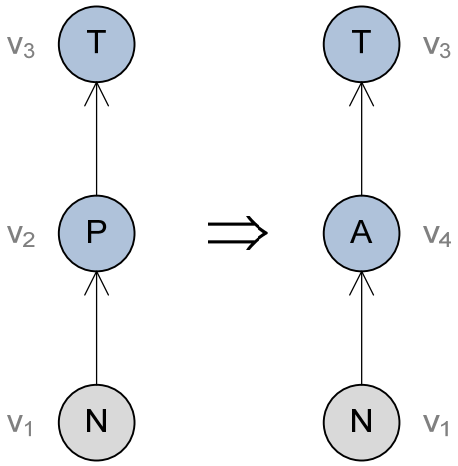
- Priorytet: 1
- Opóźnienie: TC = ∅
- Warunek uruchomienia:

$$\forall i_k: i_k \rightarrow t \in E(G): \exists n_k \rightarrow d_k \rightarrow i_k \in E(G) \wedge \forall p, q: n_p = n_q$$
$$\wedge i_k.label = "I" \wedge t.label = "T" \wedge n_k.label = "N" \wedge d_k.label = "D" \quad (5.13)$$
$$\wedge \neg (\exists v \rightarrow t \in E(G): v.label \in \{ "P", "A", "F" \})$$



Rysunek 24. Transformacja nr 2 – Tworzenie Stanu Pasywnego.

5.7.3. Transformacja – Aktywacja Procesu



Rysunek 25. Transformacja nr 3 – Aktywacja Procesu.

Wierzchołek etykietowany A reprezentuje aktywny proces klasy T. Decyzja dotycząca odpalenia produkcji nr 3 (Rysunek 25) – a tym samym uruchomienie procesu – podejmowana jest przez *Kontroler*.

Działanie procesu związane jest z opóźnieniem czasowym. Z tego względu wprowadzona została blokada uruchomienia produkcji nr 5 (Finalizacja Procesu), oznaczającej poprawne

zakończenie procesu; możliwe jest jednak wyjątkowe zakończenie obliczeń, na przykład pod wpływem awarii węzła środowiska wykonania (produkcje opisane w dalszej części pracy).

- Priorytet: 1
- Opóźnienie: TC = (T_{EV}(T, N), 5 (Finalizacja Procesu), v_4)

$$T_{EV}(T, N) = \frac{PC(T)}{CP(N)} * ME(T)\left(\frac{MC(T)}{AM(N)}\right) \tag{5.14}$$

gdzie:

PC (T) – Processor Consumption,

CP (N) – Computing Power,

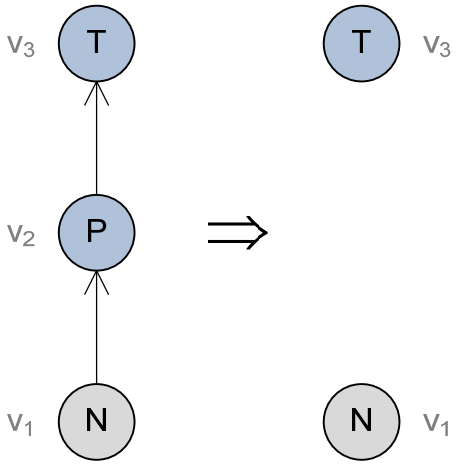
ME (T) – Memory Efficiency,

MC (T) – Memory Consumption,

AM (N) – Available Memory.

- Warunek uruchomienia: *kontroler*

5.7.4. Transformacja – Usuwanie Stanu Pasywnego



Rysunek 26. Transformacja nr 4 – Usuwanie Stanu Pasywnego.

Odpalenie produkcji usuwającej stan pasywny (Rysunek 26) inicjowane jest przez *Kontroler*. Decyzja o usunięciu potencjalnych miejsc uruchomienia zadania klasy T jest podejmowana na przykład w okolicznościach, gdy zadanie tej samej klasy zostało już uruchomione lub zakończone.

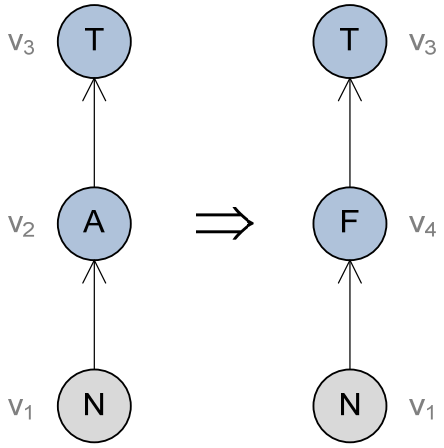
- Priorytet: 1
- Opóźnienie: TC = ∅
- Warunek uruchomienia: *kontroler*

5.7.5. Transformacja – Finalizacja Procesu

Wierzchołek etykietowany F przedstawia proces, którego wykonanie zakończyło się pomyślnie. Uruchomienie produkcji zmieniającej wierzchołek A (proces aktywny) w F (proces zakończony) (Rysunek 27) może nastąpić dopiero po pewnym czasie, co zostało zaznaczone w opisie produkcji nr 3 przedstawiającej uruchomienie procesu (z tego względu warunek EC (5.15) transformacji nr 5 jest redundantny, lecz został dodany dla zachowania przejrzystości).

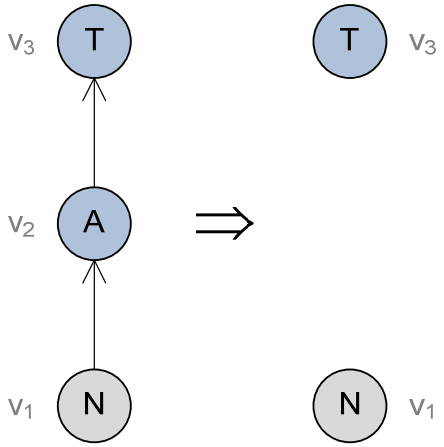
- Priorytet: 1
- Opóźnienie: TC = ∅
- Warunek uruchomienia:

$$a.label = "A" \wedge a.TTC \leq 0 \tag{5.15}$$



Rysunek 27. Transformacja nr 5 – Finalizacja Procesu.

5.7.6. Transformacja – Usuwanie Stanu Aktywnego



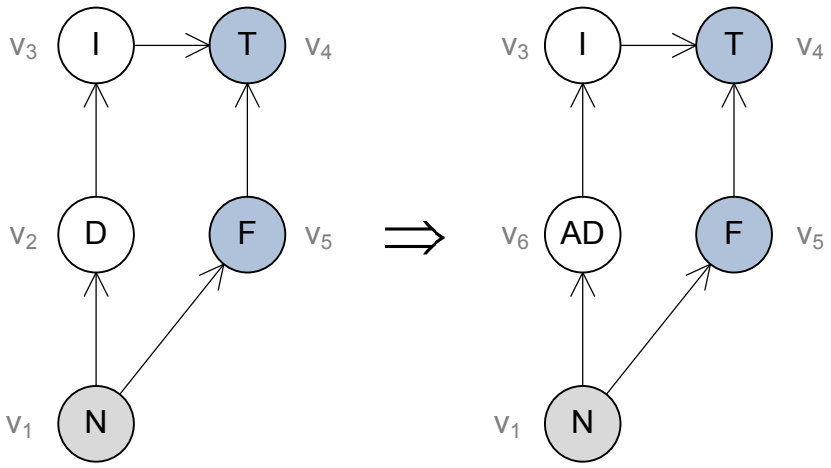
Rysunek 28. Transformacja nr 6 – Usuwanie Stanu Aktywnego.

Usuwanie stanu aktywnego (Rysunek 28) – przedstawiającego działający proces – jest produkcją uruchamianą przez *Kontroler*. Jest ona używana w sytuacji, gdy:

- zostało uruchomionych wiele procesów tej samej klasy T równolegle w celu jak najszybszego uzyskania wyników i jeden z procesów zakończył działanie, przez co pomyślne zakończenie pozostałych jest uważane za nieistotne,
- nastąpiła awaria węzła obliczeniowego N, co implikuje natychmiastowe zakończenie zaalokowanych na nim procesów i ewentualne ponowne utworzenie stanu pasywnego.

- Priorytet: 1
- Opóźnienie: TC = ∅
- Warunek uruchomienia: *kontroler*

5.7.7. Transformacja – Archiwizacja Danych Wejściowych i Archiwizacja Klonu Danych Wejściowych



Rysunek 29. Transformacja nr 7 – Archiwizacja Danych Wejściowych.

W następstwie pomyślnego zakończenia działania procesu dane wejściowe konieczne do przeprowadzenia obliczeń są oznaczane jako „archiwalne” (Rysunek 29). Produkcję tą wprowadzono z następujących powodów:

- zmiana etykiety D na AD zapobiega ponownemu wygenerowaniu wierzchołka P i w efekcie uruchomienia tego samego zadania po raz kolejny,
- konieczność usunięcia „starego” wierzchołka D jest podyktowana potencjalną możliwością wystąpienia pętli, co spowoduje utworzenie kolejnego wierzchołka o takiej samej etykiecie, reprezentującego nowy zbiór danych,
- wykorzystane dane nie są usuwane i pozostają do dyspozycji w przypadku sytuacji awaryjnej, umożliwiając powtórzenie obliczeń od miejsc nie objętych awarią (przypadek ten, ze względu na ograniczony rozmiar niniejszej rozprawy został jedynie zasygnalizowany, a jego szczegółowy opis planowany jest w dalszych pracach).

- Priorytet: 1
- Opóźnienie: TC = ∅
- Warunek uruchomienia: *brak*

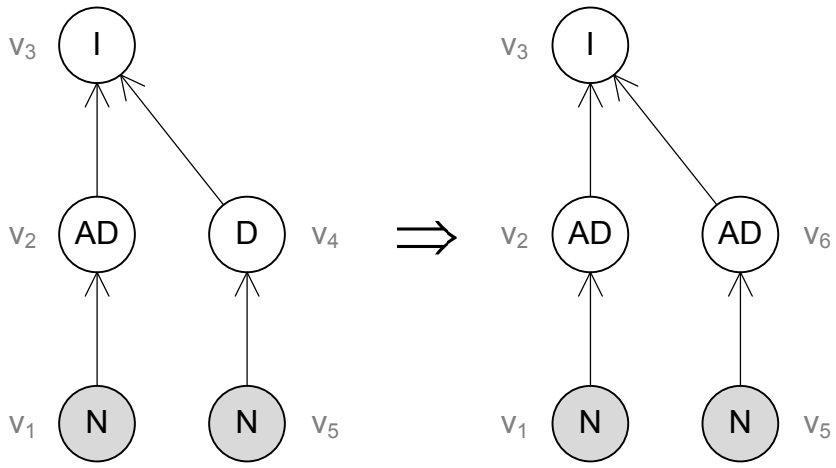
Dane wejściowe D tej samej klasy I mogą występować wielokrotnie, zaalokowane na osobnych węzłach obliczeniowych N. W momencie, gdy wszystkie instancje zadania T konsumującego dane I zakończą działanie, następuje archiwizacja klonów I, które nie zostały użyte do obliczeń (Rysunek 30).

- Priorytet: 2
- Opóźnienie: TC = ∅
- Warunek uruchomienia:

$$\forall d': d \rightarrow i \leftarrow d' \in E(G) \wedge d.label = "D" \wedge i.label = "I" \wedge d'.label = "D"$$

$$\neg \exists (n_1 \rightarrow d \rightarrow i \rightarrow t \leftarrow v \leftarrow n_2 \in E(G) \wedge n_1 = n_2 \wedge v.label \in \{ "P", "A", "F" \} \quad (5.16)$$

$$\wedge n_1.label = "N" \wedge d.label = "D" \wedge i.label = "I" \wedge t.label = "T")$$



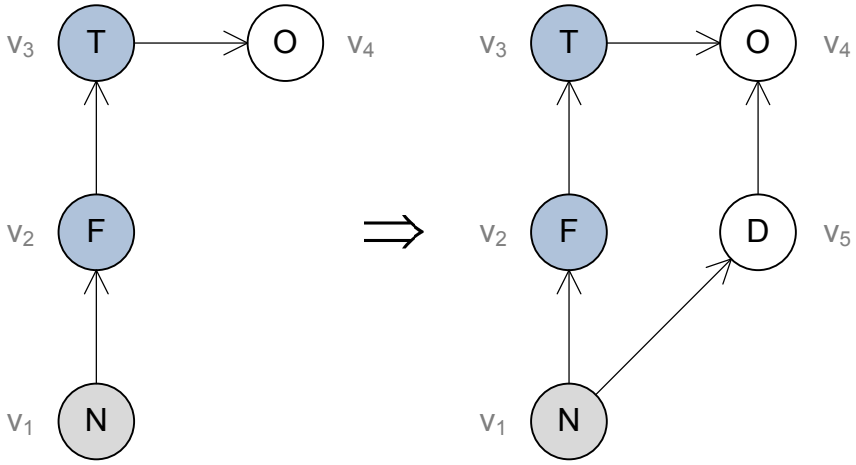
Rysunek 30. Transformacja nr 8 – Archiwizacja Klonu Danych Wejściowych.

5.7.8. Transformacja – Generacja Danych Wyjściowych

Kolejnym następstwem pomyślnego zakończenia działania procesu jest generacja danych wyjściowych – rezultatów obliczeń (Rysunek 31). Jest to druga w kolejności produkcja uruchamiana po pojawieniu się wierzchołka F (priorytet 2), kolejno dla każdego wierzchołka O będącego następnikiem T. Wszystkie instancje zbiorów danych wyjściowych zadania T alokowane są na tym samym węźle obliczeniowym, na którym uruchomiony był proces klasy T.

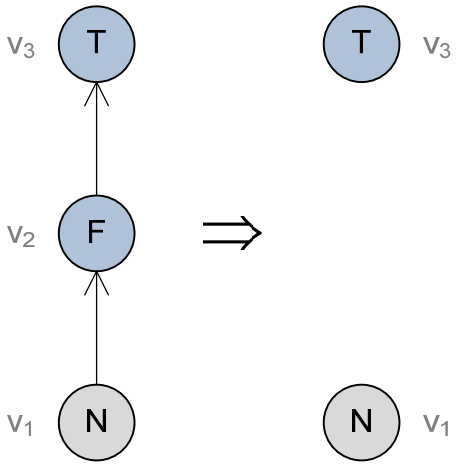
- Priorytet: 2
- Opóźnienie: TC = ∅
- Warunek uruchomienia:

$$\neg\, \exists\, d \rightarrow o \in E(G):\ d.label = "D" \wedge o.label = "O" \tag{5.17}$$



Rysunek 31. Transformacja nr 9 – Generacja Danych Wyjściowych.

5.7.9. Transformacja – Usuwanie Stanu Finalnego

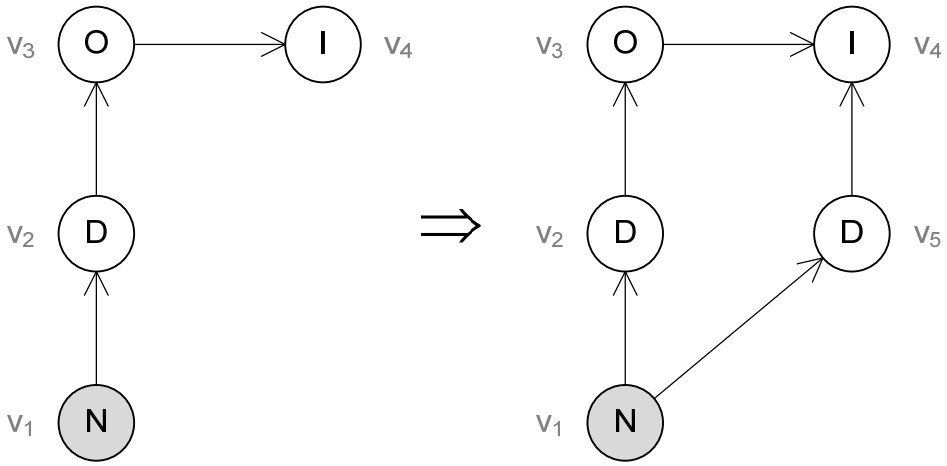


Rysunek 32. Transformacja nr 10 – Usuwanie Stanu Finalnego.

Po zakończeniu archiwizacji danych wejściowych i generacji danych wyjściowych (priorytet 3) wierzchołek F jest usuwany (Rysunek 32), co podyktowane jest potencjalną możliwością wykonania kolejnego procesu klasy T w pętli, powodujące generację kolejnego wierzchołka F.

- Priorytet: 3
- Opóźnienie: TC = ∅
- Warunek uruchomienia: *brak*

5.7.10. Transformacje – Propagacja Danych oraz Archiwizacja Danych Wyjściowych



Rysunek 33. Transformacja nr 11 – Propagacja Danych.

Dane wyjściowe (wierzchołek O) mogą być jednocześnie wejściem (wierzchołki I) wielu zadań. Produkcja numer 11 (Rysunek 33) odpowiedzialna jest za propagację danych wyjściowych: każdemu następnikowi I wierzchołka O przypisywany jest zbiór danych, zaalokowany na tym samym węźle obliczeniowym, co zbiór pierwotny.

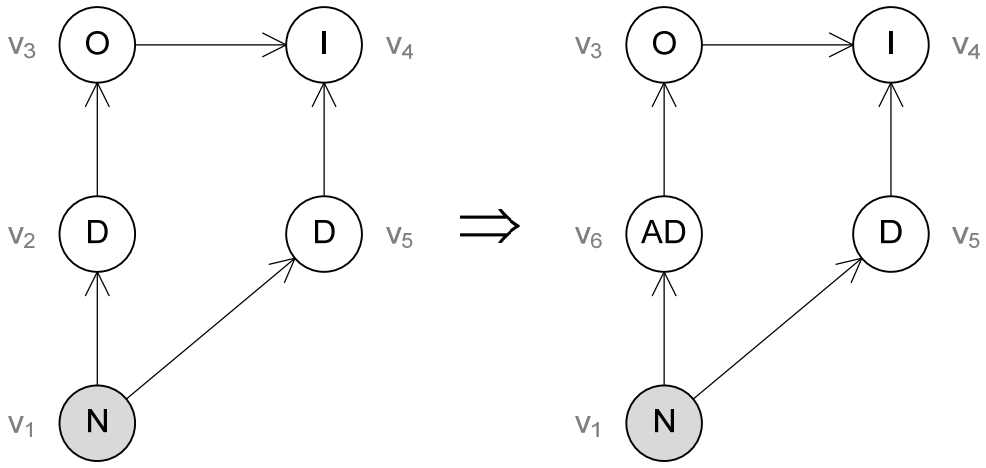
- Priorytet: 1
- Opóźnienie: TC = ∅
- Warunek uruchomienia:

$$\neg \exists\, d \rightarrow i \in E(G):\ d.label = \text{"D"} \wedge i.label = \text{"I"} \tag{5.18}$$

Po zakończonym procesie propagacji (priorytet 2) dane wyjściowe są archiwizowane (Rysunek 34). Powody wprowadzenia archiwizacji danych wyjściowych, są analogiczne do opisanych wcześniej dla produkcji numer 7:

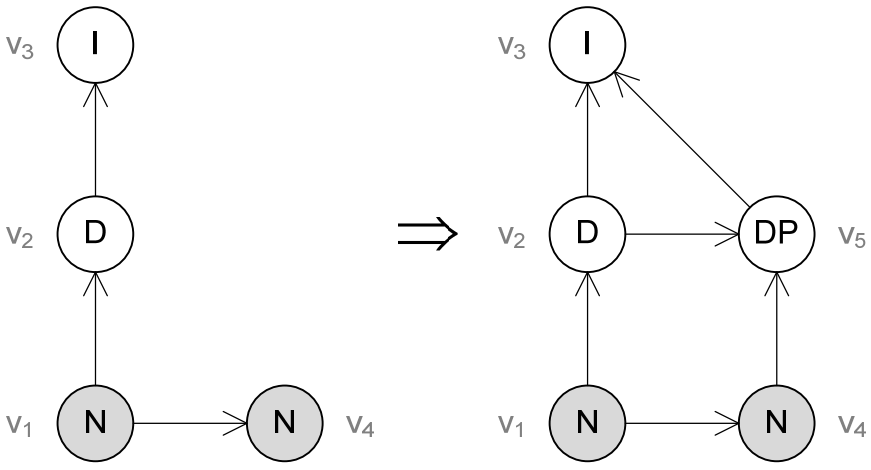
- zmiana etykiety D na AD zapobiega ponownej propagacji danych wierzchołka O na wierzchołki I,
- konieczność usunięcia „starego” wierzchołka D jest podyktowana potencjalną możliwością wystąpienia pętli,
- wykorzystane dane nie są usuwane i pozostają do dyspozycji na przykład w sytuacji awarii systemu.

- Priorytet: 2
- Opóźnienie: TC = ∅
- Warunek uruchomienia: *brak*



Rysunek 34. Transformacja nr 12 – Archiwizacja Danych Wyjściowych.

5.7.11. Transformacje – Inicjalizacja oraz Finalizacja Klonowania Danych



Rysunek 35. Transformacja nr 13a – Inicjalizacja Klonowania Danych.

Klonowanie danych na inny węzeł obliczeniowy jest procesem dwuetapowym, o rozpoczęciu którego decyduje *Kontroler*. Pierwszym krokiem jest Inicjalizacja Klonowania Danych (Rysunek 35),

tworząca wierzchołek DP (Data Placeholder) na węźle docelowym oraz wprowadzająca opóźnienie czasowe.

Wszystkie krawędzie w grafie DFA-Dynamic są krawędziami skierowanymi: dwa różne wierzchołki N mogą być więc połączone na dwa sposoby. Z tego też względu wprowadzone zostały dwie produkcje nr 13: 13a przedstawiona na Rysunku 35, oraz 13b z krawędzią pomiędzy wierzchołkami N, skierowana w przeciwną stronę. W celu zachowania przejrzystości, w dalszej części pracy produkcje 13a i 13b będą określane jako „produkcja nr 13", jeżeli zastosowanie jednej z nich jednoznacznie wynika z kontekstu.

- Priorytet: 1
- Opóźnienie: TC = ($T_{TR}(I, N_1 \rightarrow N_2)$, 14 (Finalizacja Klonowania Danych), v_5)

$$T_{TR}(I,\ N_1 \rightarrow N_2) = \frac{DS(I)}{NB(N_1 \rightarrow N_2)} \tag{5.19}$$

gdzie:

DS (I) – Data Size,

NB ($N_1 \rightarrow N_2$) – Network bandwidth.

- Warunek uruchomienia: *kontroler*

Na potrzeby niniejszej pracy przyjęto, że opóźnienie czasowe związane z klonowaniem danych na inny węzeł obliczeniowy zależne jest jedynie od wielkości przesyłanego zbioru danych (parametr DS) oraz przepustowości kanału, którym odbywa się transmisja (parametr NB). Istnieje jednak możliwość dokładniejszego opisu procesu transferu poprzez wprowadzenie między innymi charakterystyki obciążenia procesora. Zagadnienie to jednak wykracza poza ramy niniejszej rozprawy i jest przewidziane jako temat przyszłych badań.

Uruchomienie produkcji Finalizacja Klonowania Danych (Rysunek 36) możliwe jest po upływie czasu, założonego wraz z uruchomieniem poprzednio opisanej produkcji numer 13. Finalizacja przekształca tymczasowy wierzchołek DP w zbiór danych D, możliwy do wykorzystania przez proces na węźle docelowym.

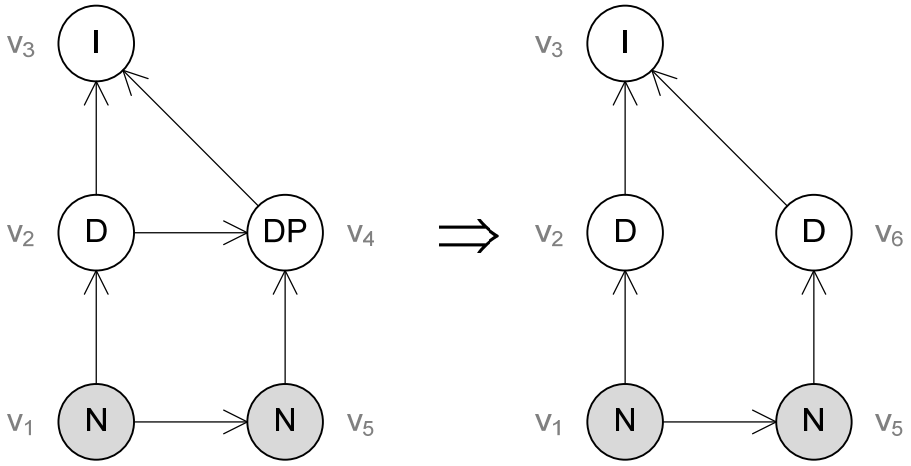
Analogicznie jak w poprzednim przypadku, występują dwie produkcje: 14a oraz 14b, różniące się między sobą skierowaniem krawędzi pomiędzy wierzchołkami N, określane jako „produkcja nr 14", jeżeli zastosowanie odpowiedniej wynika z kontekstu.

Podobnie jak dla produkcji nr 5 (Finalizacja Procesu), warunek EC (5.20) jest redundantny i został wprowadzony jedynie dla zachowania przejrzystości.

- Priorytet: 1
- Opóźnienie: TC = ∅

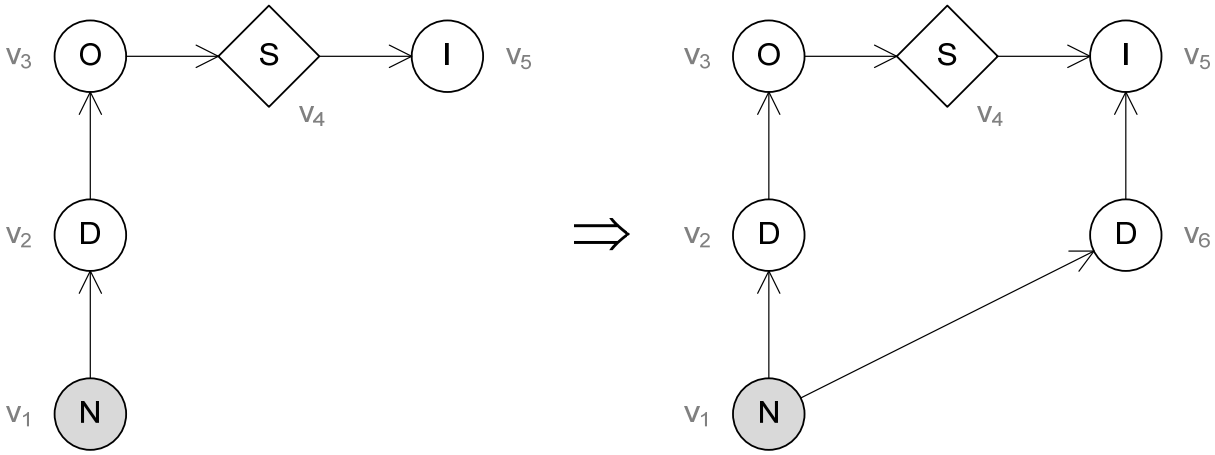
- Warunek uruchomienia:

$$dp.label = "DP" \wedge dp.TTC \leq 0 \tag{5.20}$$



Rysunek 36. Transformacja nr 14a – Finalizacja Klonowania Danych.

5.7.12. Transformacje – Selekcja oraz Finalizacja Selekcji

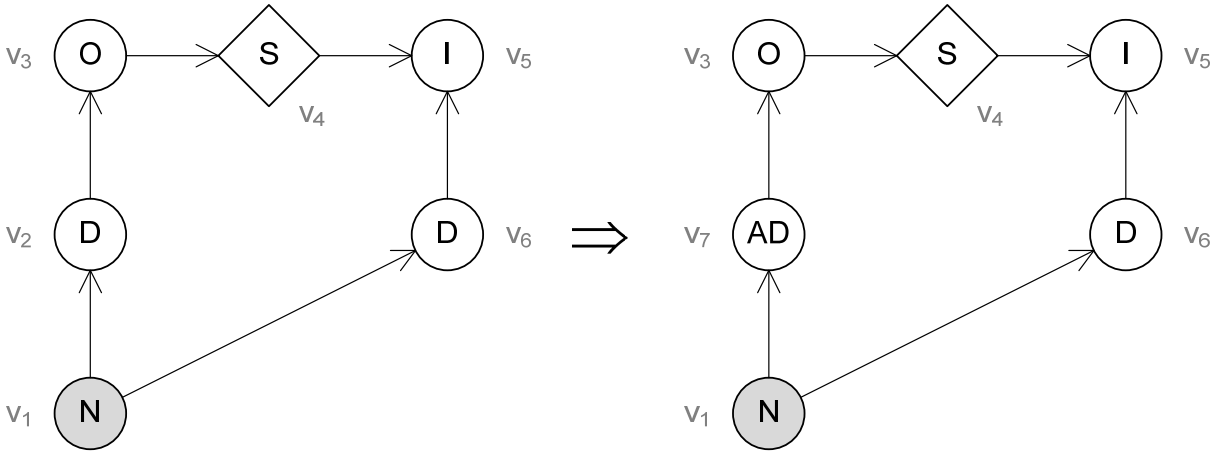


Rysunek 37. Transformacja nr 15 – Selekcja.

Produkcja Selekcji (Rysunek 37) jest pod względem funkcjonalnym podobna do produkcji numer 11 propagującej dane, lecz jest warunkowo wykonywana tylko wtedy, gdy spełniony jest warunek definiowany przez wierzchołek S określony funkcją $f_{SF}(O_{VAL})$, gdzie O_{VAL} są to wartości zbioru danych, reprezentowanego przez wierzchołek etykietowany O.

- Priorytet: 1
- Opóźnienie: TC = ∅
- Warunek uruchomienia:

$$f_{SF}(O_{VAL}) > 0 \ \wedge (\neg \exists \ d \rightarrow i \in E(G): \ d.label = "D" \wedge i.label = "I") \tag{5.21}$$



Rysunek 38. Transformacja nr 16 – Finalizacja Selekcji.

Po zakończonym procesie selekcji (priorytet 2), produkcja numer 16 powoduje archiwizację danych wyjściowych (Rysunek 38). Powody wprowadzenia archiwizacji danych wyjściowych są analogiczne do opisanych wcześniej dla produkcji numer 12 (Archiwizacja Danych Wyjściowych).

- Priorytet: 2
- Opóźnienie: TC = ∅
- Warunek uruchomienia: *brak*

5.7.13. Transformacje – Finalizacja Obliczeń oraz Selekcja Finalizacji Obliczeń

Produkcja finalizująca obliczenia (Rysunek 39) uruchamiana jest w momencie, gdy uzyskane zostaną wszystkie zbiory danych wyjściowych, zarówno dla wierzchołków O bezpośrednio połączonych z FS (Final State), jak i tych, dla których jako następnik istnieje wierzchołek decyzyjny S, będący poprzednikiem FS.

- Priorytet: 4
- Opóźnienie: TC = ∅

- Warunek uruchomienia:

$$\forall\, o \in V(G): \tag{5.22}$$

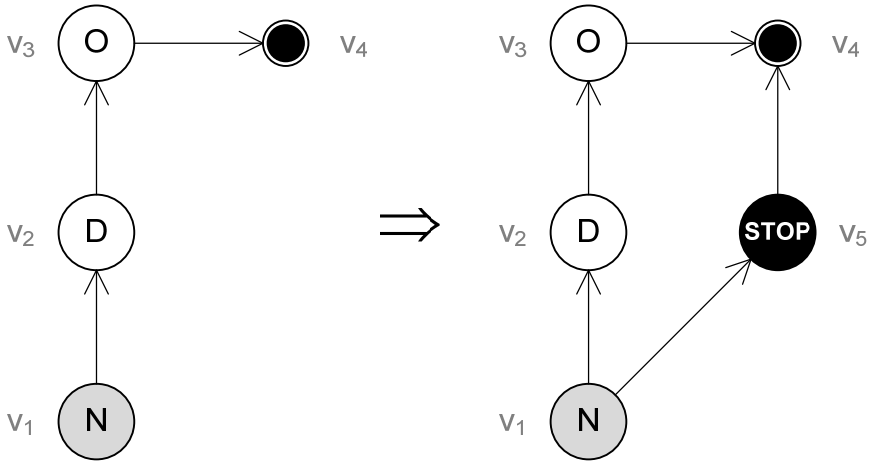
$$\{ \exists\, o \rightarrow fs \in E(G) \wedge o.label = \text{"O"} \wedge fs.label = \text{"FS"}$$

$$\Rightarrow \exists\, d \rightarrow o \in E(G) \wedge d.label = \text{"D"} \}$$

$$\vee$$

$$\{ \exists\, o \rightarrow s \rightarrow fs \in E(G) \wedge o.label = \text{"O"} \wedge fs.label = \text{"FS"} \wedge s.label = \text{"S"}$$

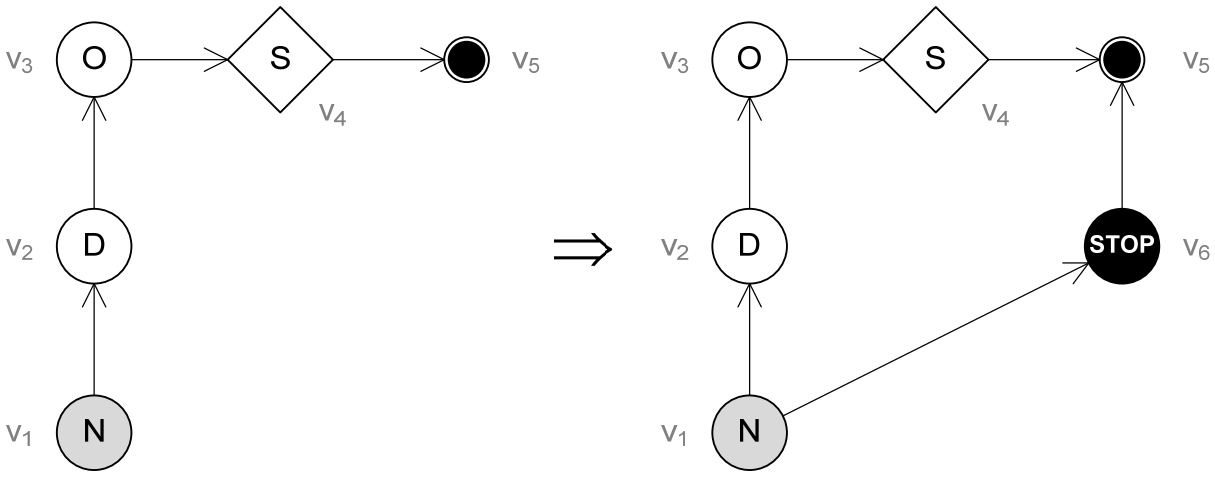
$$\Rightarrow \exists\, d \rightarrow o \in E(G) \wedge d.label = \text{"D"} \wedge f_{SF}(O) > 0 \}$$



Rysunek 39. Transformacja nr 17 – Finalizacja Obliczeń.

Produkcja Selekcja Finalizacji Obliczeń (Rysunek 40), podobnie jak produkcja wcześniejsza, uruchamiana jest w celu zakończenia wykonywania obliczeń, gdy wyprodukowane zostaną wszystkie zbiory danych wyjściowych.

- Priorytet: 4
- Opóźnienie: TC = ∅
- Warunek uruchomienia: (5.22)



Rysunek 40. Transformacja nr 18 – Selekcja Finalizacji Obliczeń.

5.7.14. Transformacje – podsumowanie

Przedstawione powyżej transformacje grafowe umożliwiają zamodelowanie podstawowych akcji dynamiki modelu DFA.

Ze względu na ograniczony rozmiar pracy pominięto produkcje zmian środowiska sprzętowego: przyłączanie oraz usuwanie węzłów obliczeniowych, dodawanie oraz usuwanie połączeń między węzłami obliczeniowymi.

Z tego samego powodu nie podano produkcji związanych z awarią węzłów obliczeniowych, powodujących nagłe przerwanie obliczeń oraz utratę danych, a jedynie zasygnalizowano możliwość wykorzystania niektórych z podanych transformacji w szczególnych przypadkach.

Pominięto również temat związany z odpornością systemu na częściowe awarie oraz cofnięcie obliczeń do stanu „stabilnego” i kontynuację pracy.

W przypadku dużych sieci LAN oraz sieci WAN, będących środowiskami obliczeniowymi [74], często zachodzi zjawisko czasowego odłączenia podsieci, w których kontynuowane są obliczenia [75]. Istnieje w takim przypadku możliwość decentralizacji podejmowania decyzji oraz dokonywania transformacji przy zachowaniu spójności grafu, dyskutowana w pracach [61], [22] (Kotulski) oraz [76].

Wyżej wymienione zagadnienia ze względu na ograniczenia objętości niniejszej rozprawy przewidziane są jako temat przyszłych prac.

5.8. Analiza przebiegu przykładowego procesu DFA

Wprowadzone w poprzednim rozdziale produkcje gramatyki pozwalają kontrolować proces alokacji i kolejność wykonania zadań realizujących obliczenia w ramach Dynamicznej Analizy Finansowej. Przykładowy proces zilustrowany jest Rysunkami 41 i 42 oraz danymi zawartymi w Tabeli 7.

Na Rysunku 41 przedstawiono sekwencję grafów obrazujących kolejne etapy wykonania przykładowego modelu DFA.

Tabela 7 zawiera listę odpalonych produkcji z wyszczególnieniem wierzchołków charakterystycznych biorących udział w transformacji, pozwalających jednoznacznie określić, jakie nastąpiły zmiany.

Rysunek 42 przedstawia analizę czasową wykonania przykładowego modelu: poziome prostokąty odpowiadają opóźnieniom czasowym powodowanym transferem danych oraz wykonywaniem obliczeń. Liczby w nawiasach kwadratowych odpowiadają wartościom kolumny „Krok" w Tabeli 7, wskazując miejsce odpalenia produkcji.

W przedstawionym przykładzie za miejsce startowe obliczeń arbitralnie przyjęto wierzchołek N_1, który połączono z IS (Initial State) poprzez wierzchołek START. Poniżej, na podstawie Rysunku 41, omówione zostaną kolejne transformacje pierwotnego grafu.

Graf I

Na wierzchołku startowym N_1, za pomocą produkcji nr 1 w kroku [1] i [2] alokowane są fizyczne zbiory danych D, odpowiadające klasom abstrakcji reprezentowanym przez I_A oraz I_B.

Graf II

Gdy nie można wykonać produkcji nr 1 w kontekście danego węzła obliczeniowego (etykietowanego przez „N"), wykona się produkcja nr 2, o czym zdecyduje niższy priorytet tej produkcji w stosunku do produkcji nr 1. Wykonanie produkcji nr 2 powoduje to utworzenie stanu pasywnego P (krok [3]). Podobna sytuacja ma miejsce dla wierzchołka T_B (krok [4]).

Graf III

Kontroler podejmuje decyzję o aktywacji procesu klasy T_A na wierzchołku N_1 [5]. Przejście w stan aktywny i prowadzenie obliczeń wiąże się z opóźnieniem czasowym $T_{EV}(T_A, N_1)$, zależnym od parametrów wierzchołka T_A i węzła obliczeniowego N_1, o przyjętej wartości Δ_5 = 30 min. Opóźnienie w tym przypadku polega na możliwości odpalenia produkcji nr 4 finalizującej obliczenia dopiero po upływie wspomnianego czasu Δ_5 (w tym przykładzie zadanie otrzyma procesor na własność). Zaczynając się w t_0, punkt zakończenia obliczeń klasy T_A został oznaczony t_1.

Jednocześnie, aby zapewnić maksymalną efektywność, *Kontroler* podjął decyzję o przeniesieniu części obliczeń na węzeł N_2 w celu przyspieszenia czasu zakończenia ewaluacji modelu. Uruchomiona została produkcja Inicjalizacji Klonowania Danych [6], która powoduje opóźnienie czasowe $T_{TR}(I_B, N_1 \rightarrow N_2)$, blokujące produkcję nr 14 (Finalizacja Klonowania Danych), zależne od rozmiaru danych I_B oraz przepustowości łącza oznaczonego krawędzią $N_1 \rightarrow N_2$ (ustalone na Δ_6 = 5 min). Czas rozpoczęcia klonowania danych to t_0, natomiast czas zakończenia został oznaczony jako t_2.

Graf IV

Po upływie czasu Δ_6 możliwe jest odpalenie produkcji Finalizacji Klonowania Danych I_B (krok [7]) oraz utworzenia stanu pasywnego zadania T_B na wierzchołku N_2 [8].

Graf V

Kontroler podejmuje decyzję o uruchomieniu procesu dla zadania T_B na węźle N_2. Odpalana jest produkcja Aktywizacji Procesu (krok [9]) o czasie t_2. Opóźnienie związane z przetwarzaniem danych zostało ustalone na Δ_9 = 10 min, natomiast czas zakończenia obliczeń oznaczony t_3.

Graf VI

Po upływie czasu Δ_9 uruchamiana jest produkcja finalizacji procesu klasy T_B (krok [10]). Zakończenie obliczeń sukcesem powoduje, że *Kontroler* podejmuje decyzję o usunięciu stanu pasywnego T_B dla wierzchołka N_1 [11].

Dane wejściowe klasy I_B zaalokowane na węźle N_2 zostają zarchiwizowane, gdyż zostały już wykorzystane do obliczeń [12]. Następuje także archiwizacja danych wejściowych I_B, zaalokowanych na węźle N_1 [13]. Ostatecznie generowany jest zbiór danych wyjściowych klasy O_B na węźle N_2, na którym wykonywane były obliczenia [14].

Graf VII

Po zakończonym procesie generowania danych, usuwany jest stan finalny T_B [15]. Następuje propagacja danych wyjściowych O_B na I_{C2} [16].

Graf VIII

Dane wyjściowe klasy O_B zostają zarchiwizowane [17], gdyż nastąpiła propagacja na wszystkie wierzchołki będące następnikami. O czasie t_3, *Kontroler* podejmuje także decyzję o odpaleniu produkcji inicjalizacji klonowania danych klasy I_{C2} z węzła N_2 na N_1 [18], która wiąże się z opóźnieniem spowodowanym transferem danych Δ_{18} = 8 min i trwa do czasu t_4.

Dla zachowania przejrzystości, ukryte zostały wierzchołki reprezentujące dane archiwalne klasy I_B, zaalokowane na węzłach N_1 i N_2.

Graf IX

O czasie t_4 istnieje możliwość uruchomienia produkcji finalizującej klonowanie danych klasy I_{C2} z węzła N_2 na N_1 [19]. Trwa następnie oczekiwanie na zakończenie obliczeń prze proces klasy T_A, aby po uzyskaniu wyników uruchomić proces klasy T_C.

Graf X

O czasie t_1 możliwe jest odpalenie produkcji finalizującej uruchomienie procesu klasy T_A na węźle N_1 (krok [20]). Dla zachowania przejrzystości, ukryty zostaje wierzchołek danych archiwalnych klasy O_B na węźle N_2.

Graf XI

Po zakończeniu obliczeń T_A następuje archiwizacja danych wejściowych I_A na węźle N_1 [21]. Generowane są także zbiory danych wyjściowych O_{A1} [22] oraz O_{A2} [23] na węźle N_1, na którym przeprowadzane były obliczenia.

Graf XII

Po zakończeniu generacji danych wyjściowych, następuje usunięcie stanu finalnego klasy T_A [24]. Dane wyjściowe O_{A2} propagowane są na wierzchołek I_{C1} [25], a następnie uruchamiana jest produkcja archiwizacji danych O_{A2} [26].

Graf XIII

Wszystkie zbiory danych wejściowych zadania klasy T_C zostały zgromadzone na tym samym węźle obliczeniowym N_1, w rezultacie czego następuje uruchomienie produkcji tworzenia stanu pasywnego dla zadania T_C [27]. Ukryty zostaje także wierzchołek danych archiwalnych klasy I_A.

Graf XIV

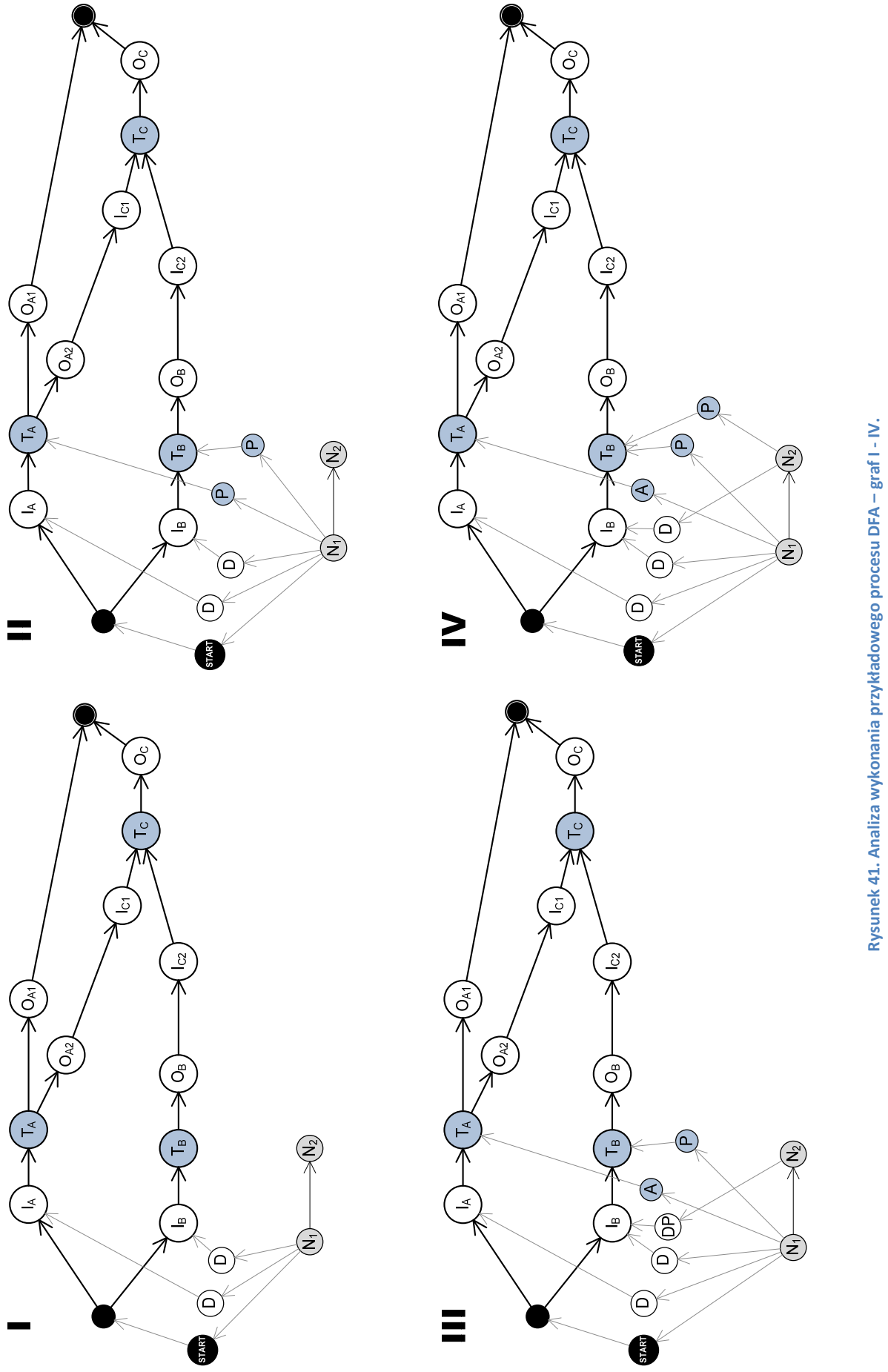
Kontroler podejmuje decyzję o uruchomieniu obliczeń klasy T_C na wierzchołku N_1, w wyniku czego o czasie t_1 odpalona zostaje produkcja aktywacji procesu [28] wprowadzająca opóźnienie $\Delta_{28} = 15$ min, którego zakończenie oznaczono t_5. Ukryty zostaje także wierzchołek danych archiwalnych O_{A2} na węźle N_1.

Graf XV

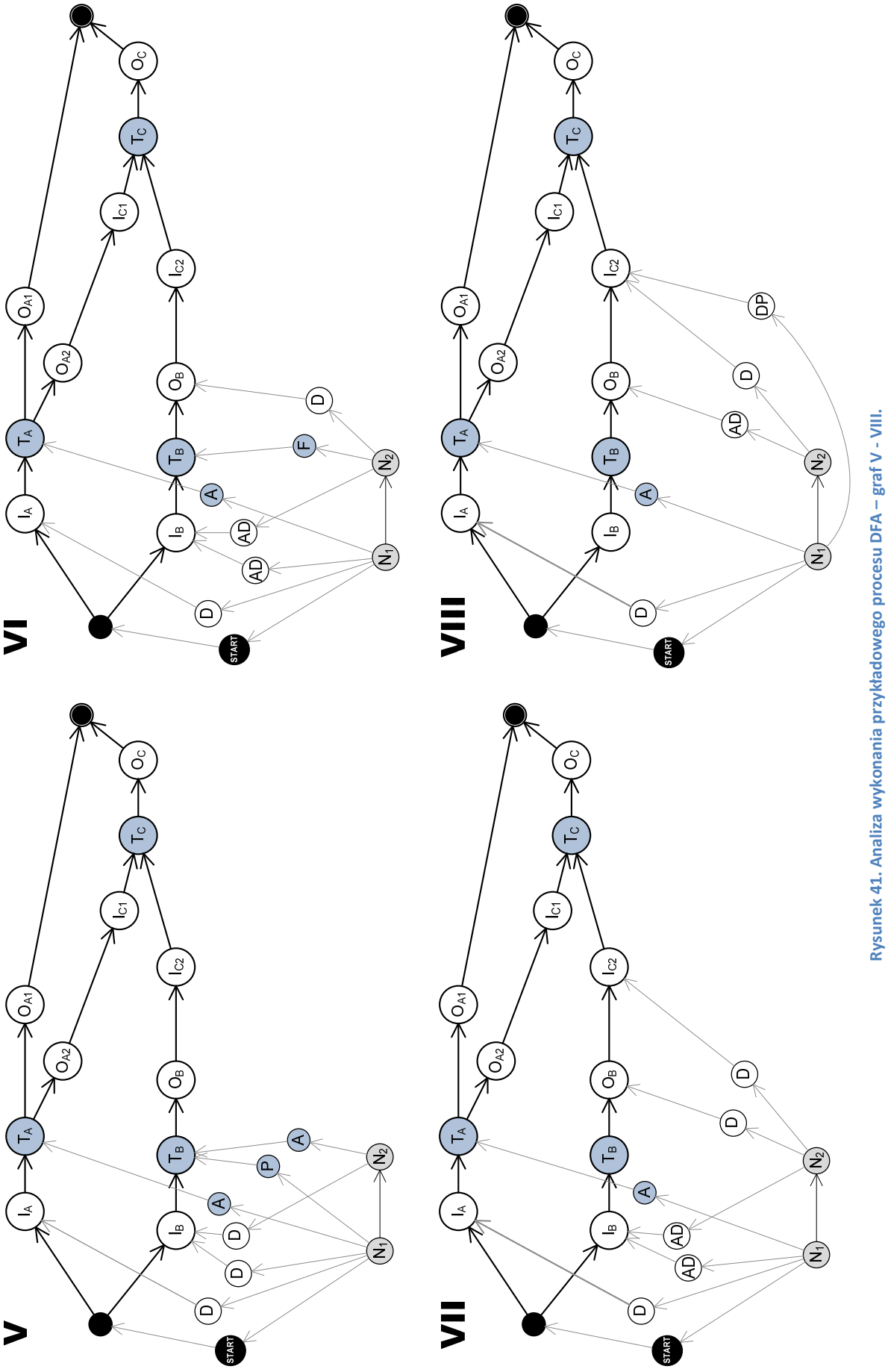
O czasie t_5 następuje odpalenie produkcji finalizacji obliczeń zadania T_C [29]. Następnie archiwizowane są wszystkie zbiory danych wejściowych zadania T_C zaalokowane na węźle N_1: I_{C1} [30] oraz I_{C2} [31]. Archiwizowany jest także klon danych I_{C2} na węźle N_2 [32] oraz odpalana produkcja generująca dane wyjściowe O_C na węźle N_1 [33].

Graf XVI

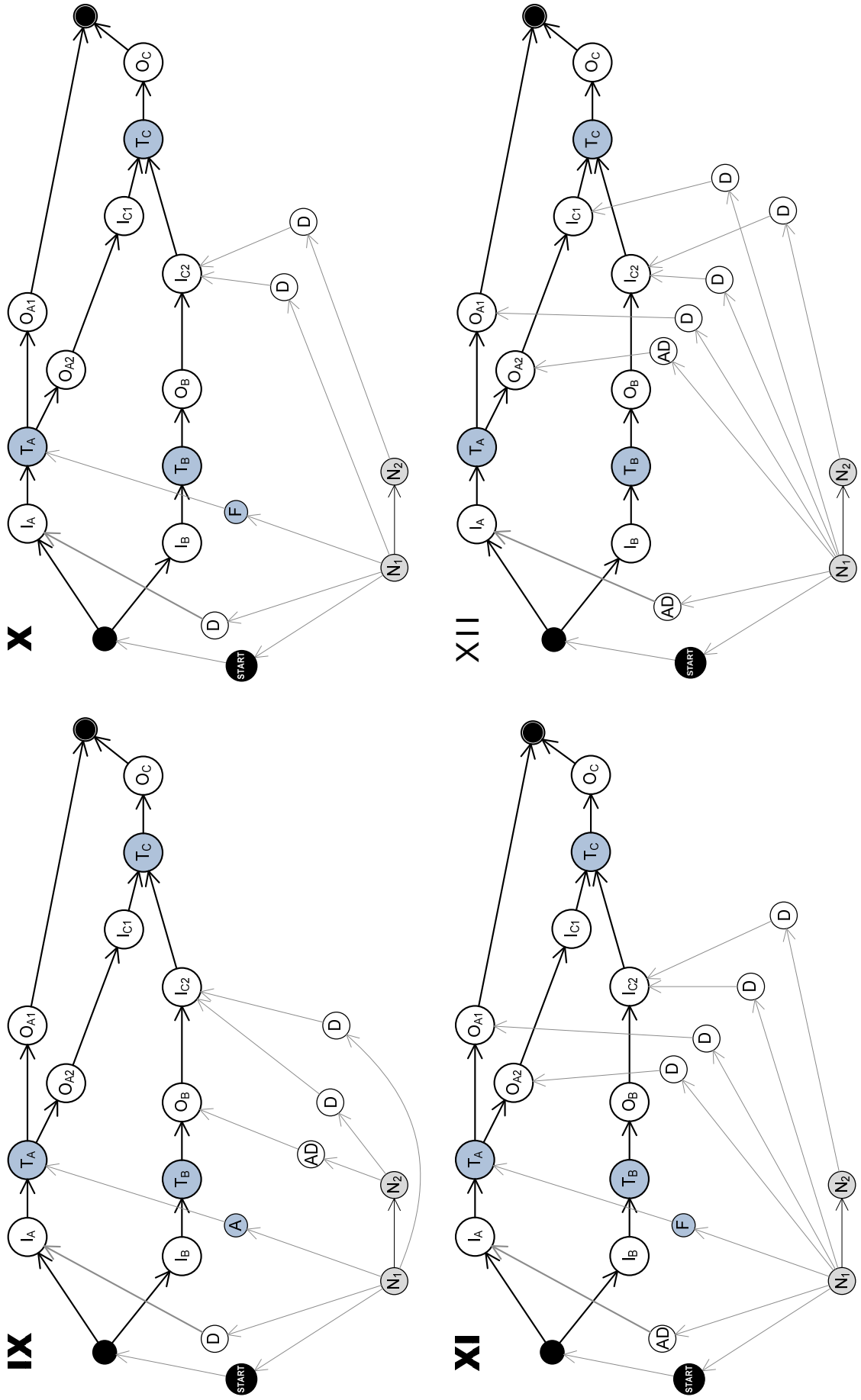
Po zakończeniu generacji danych O_C, usuwany jest stan finalny dla zadania klasy T_C [34]. Ostatecznie, na skutek uzyskania zbiorów danych wszystkich klas połączonych z wierzchołkiem FS (Final State), następuje uruchomienie produkcji finalizacji całego procesu obliczeniowego [35].



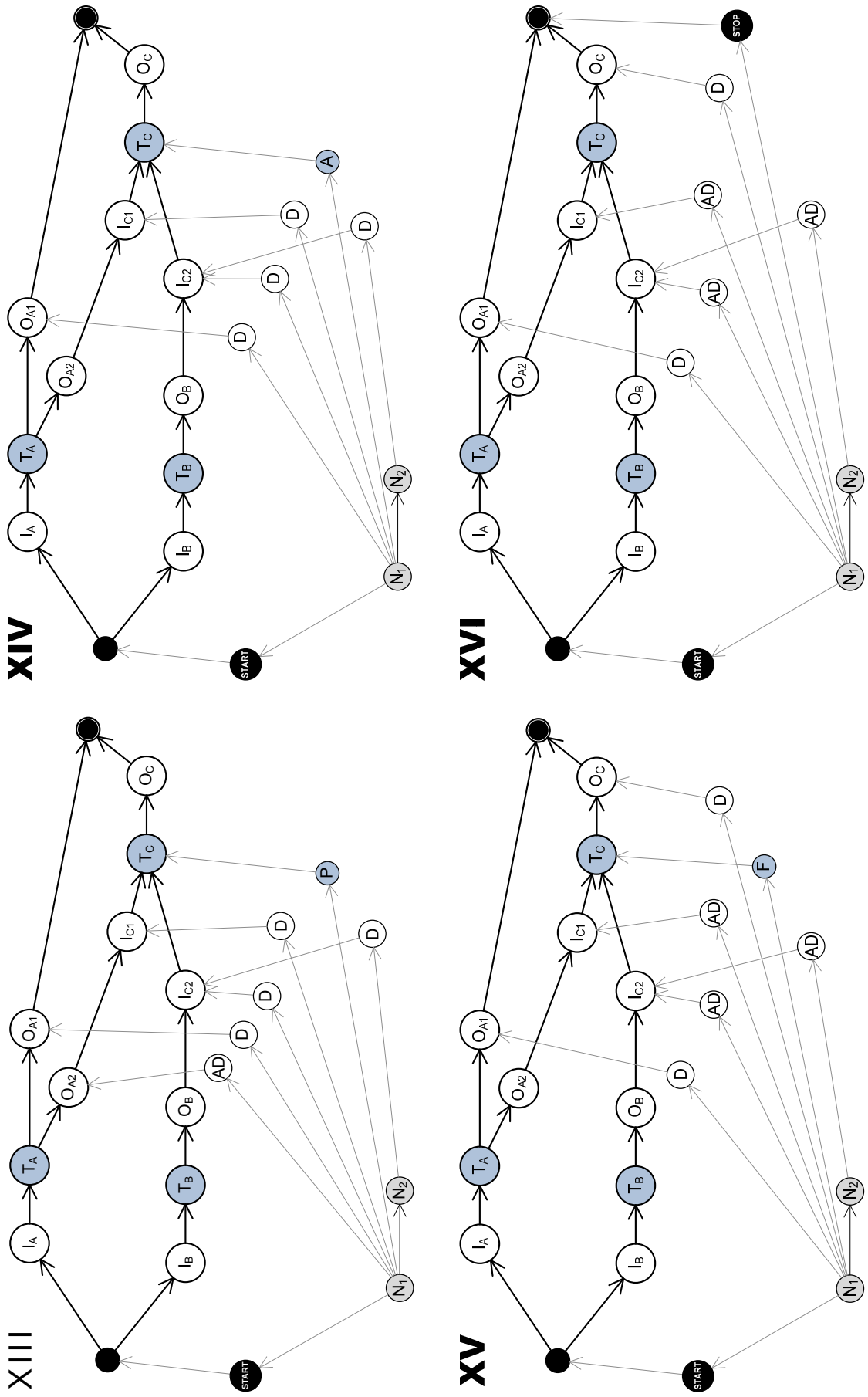
Rysunek 41. Analiza wykonania przykładowego procesu DFA – graf I - IV.



Rysunek 41. Analiza wykonania przykładowego procesu DFA – graf V - VIII.



Rysunek 41. Analiza wykonania przykładowego procesu DFA – graf IX - XII.

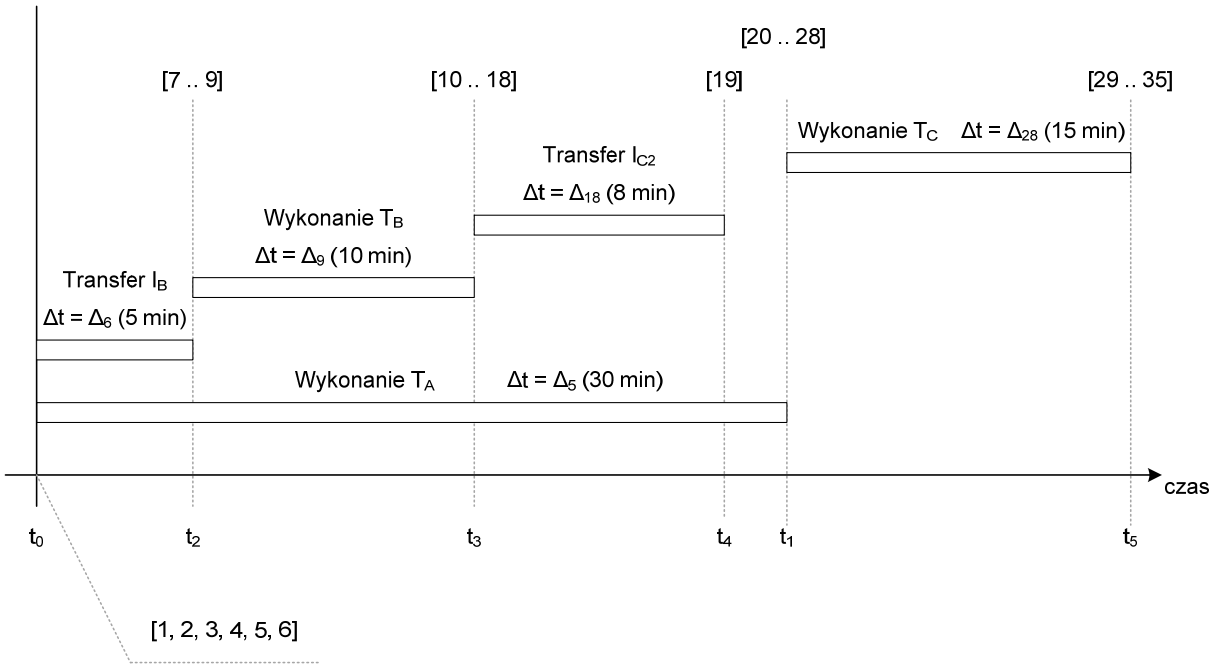


Rysunek 41. Analiza wykonania przykładowego procesu DFA – graf XIII - XVI.

Krok	Rysunek 41, graf nr	Nr produkcji	Opis akcji	Wierzchołki charakterystyczne
1	I	1	Inicjalizacja Obliczeń	I_A
2	I	1	Inicjalizacja Obliczeń	I_B
3	II	2	Tworzenie Stanu Pasywnego	T_A i N_1
4	II	2	Tworzenie Stanu Pasywnego	T_B i N_1
5	III	3	Aktywacja Procesu	T_A i N_1
6	III	13	Inicjalizacja Klonowania Danych	I_B i N_2
7	IV	14	Finalizacja Klonowania Danych	I_B i N_2
8	IV	2	Tworzenie Stanu Pasywnego	T_B i N_2
9	V	3	Aktywacja Procesu	T_B i N_2
10	VI	5	Finalizacja Procesu	T_B i N_2
11	VI	4	Usuwanie Stanu Pasywnego	T_B i N_1
12	VI	7	Archiwizacja Danych Wejściowych	I_B i N_2
13	VI	8	Archiwizacja Klonu Danych Wejściowych	I_B i N_1
14	VI	9	Generacja Danych Wyjściowych	O_B i N_2
15	VII	10	Usuwanie Stanu Finalnego	T_B i N_2
16	VII	11	Propagacja Danych	O_B i I_{C2}
17	VIII	12	Archiwizacja Danych Wyjściowych	O_B i N_2
18	VIII	13	Inicjalizacja Klonowania Danych	I_{C2} i N_1
-	VIII	-	Ukryto wierzchołek AD	I_B i N_1
-	VIII	-	Ukryto wierzchołek AD	I_B i N_2
19	IX	14	Finalizacja Klonowania Danych	I_{C2} i N_1
20	X	5	Finalizacja Procesu	T_A i N_1
-	X	-	Ukryto wierzchołek AD	O_B i N_2
21	XI	7	Archiwizacja Danych Wejściowych	I_A i N_1
22	XI	9	Generacja Danych Wyjściowych	O_{A1} i N_1
23	XI	9	Generacja Danych Wyjściowych	O_{A2} i N_1
24	XII	10	Usuwanie Stanu Finalnego	T_A i N_1
25	XII	11	Propagacja Danych	O_{A2} i I_{C1}
26	XII	12	Archiwizacja Danych Wyjściowych	O_{A2} i N_1
27	XIII	2	Tworzenie Stanu Pasywnego	T_C i N_1
-	XIII	-	Ukryto wierzchołek AD	I_A i N_1
28	XIV	3	Aktywacja Procesu	T_C i N_1
-	XIV	-	Ukryto wierzchołek AD	O_{A2} i N_1

29	XV	5	Finalizacja Procesu	T_C i N_1
30	XV	7	Archiwizacja Danych Wejściowych	I_{C1} i N_1
31	XV	7	Archiwizacja Danych Wejściowych	I_{C2} i N_1
32	XV	8	Archiwizacja Klonu Danych Wejściowych	I_{C2} i N_2
33	XV	9	Generacja Danych Wyjściowych	O_C i N_1
34	XVI	10	Usuwanie Stanu Finalnego	T_C i N_1
35	XVI	17	Finalizacja Obliczeń	O_{A2} i N_1

Tabela 7. Analiza wykonania przykładowego procesu DFA.



Rysunek 42. Analiza czasowa wykonania przykładowego procesu DFA

(liczby w nawiasach kwadratowych odpowiadają wartościom kolumny „Krok" w Tabeli 7).

5.9. Podsumowanie

W wyniku przeprowadzonych przez autora badania nad procesami Dynamicznych Analiz Finansowych, postawione zostały wymagania (5.1) – (5.4) dotyczące modelu obliczeniowego DFA. Stwierdzono, że dotychczas nie został zdefiniowany żaden formalizm, mogący zaspokoić wymienione postulaty.

Odpowiedzią jest wprowadzony w niniejszym rozdziale model DFA-Dynamic, spełniający wszystkie wymienione punkty:

(5.1) całościowo oraz spójnie przedstawia bieżący stan oraz przebieg obliczeń,

- zdefiniowano graf DFA-Snapshot przedstawiający stan systemu w pewnej chwili w czasie, zawierający graf DFA-Static pobierany z repozytorium i reprezentujący klasę problemu, rozszerzony o elementy środowiska wykonania: węzły obliczeniowe oraz procesy i instancje danych,
- do opisu zmian użyto transformacji grafowych, wzbogaconych między innymi o czynnik czasowy,
- zdefiniowano upływ czasu dla transformacji grafowych,
- zdefiniowano kompletny zbiór transformacji grafowych opisujących dynamikę DFA.

(5.2) automatyzuje proces alokacji zadań,

- wprowadzono *Kontroler*, podejmujący decyzję o wyborze transformacji sterujących przebiegiem obliczeń.

(5.3) optymalizuje alokację zadań,

- ściśle zdefiniowano kryterium optymalizacji obliczeń,
- wyodrębniono jednostkę *Kontrolera* zawierającą algorytmy optymalizacyjne oraz określono jej użycie w algorytmie podejmowania decyzji.

(5.4) umożliwia polepszenie alokacji w kolejnych iteracjach wykonania.

- wprowadzono parametr modelu wykorzystywany przez algorytmy optymalizacyjne do przechowywania informacji dotyczącej wykonania procesu opartego na tym modelu,
- przewidziano możliwość zapamiętania w repozytorium charakterystyki wykonania wraz z modelem.

W niniejszym rozdziale przedstawiona została także lista transformacji grafowych opisujących dynamikę zmian systemu DFA, wraz z ze szczegółowym opisem użycia każdej z nich.

Całość ilustruje kompletny przykład przebiegu procesu obliczeniowego, wykorzystujący zaproponowane transformacje, zawierający analizę każdego z kroków transformacji.

Ze względu na ograniczony rozmiar niniejszej rozprawy, do szczegółowej analizy pozostaje szereg problemów:

- Zaproponowana funkcja *loadFactor* (Algorytm 5.1, strona 54) modeluje zachowanie wzajemnych interferencji zadań jedynie dla ściśle określonego środowiska sprzętowego. Planowane jest opracowanie tejże funkcji dla innych środowisk, w tym dla często pojawiającej się topologii gwiazdy rozszerzonej.
- Opóźnienie związane z klonowaniem danych na inny węzeł obliczeniowy modelowane jest jedynie za pomocą dwóch wielkości: rozmiaru danych i przepustowości łącza. W praktyce jednak proces ten jest o wiele bardziej skomplikowany: z obserwacji wynika, że czas potrzebny na przesłanie N (N >> 1) wiadomości o wielkości 1 bajta jest wielokrotnie większy,

niż czas potrzebny na przesłanie jednej wiadomości o rozmiarze N bajtów, co związane jest między innymi z wiązaniem danych oraz ich podziałem na pakiety [77]. Spostrzeżenie to wymaga dalszej analizy oraz ewentualnego rozszerzenia zbioru parametrów modelu.

- Lista transformacji grafowych modelujących dynamikę zmian nie zawiera pozycji modyfikujących topologie środowiska sprzętowego: dodawania / usuwania węzłów obliczeniowych oraz dodawania / usuwania połączeń pomiędzy węzłami.
- Pominięto także modelowanie częściowych awarii sprzętowych oraz analizę możliwości kontynuacji procesu obliczeniowego. Przewidziano jednak wprowadzenie takiej opcji w przyszłości poprzez zapamiętanie historii obliczeń (5.8) oraz pozostawienie referencji do wykorzystanych danych w postaci wierzchołków etykietowanych „AD" (Archived Data).
- W przypadku dużych sieci LAN oraz sieci WAN obserwowane jest zjawisko czasowego rozłączania podsieci. Dalszych badań wymaga analiza możliwości kontynuowania obliczeń w takim przypadku poprzez decentralizacje podejmowania decyzji oraz agregację wyników częściowych po odzyskaniu spójności środowiska. Opcja ta została wzięta pod uwagę na etapie wyboru grafów jako formalizmu opisu modelu DFA, dla których opracowane zostały metody równoległej modyfikacji zachowującej spójność globalną (Kotulski [22] [61]).

6. Kryteria optymalizacji wykonania procesów DFA

Wprowadzony w poprzednim rozdziale model DFA-Dynamic pozwolił w ścisły sposób opisać proces DFA wraz ze środowiskiem sprzętowym na pewnym etapie wykonania, natomiast zdefiniowane transformacje grafowe umożliwiły opis dynamiki zmian tego modelu w trakcie trwania obliczeń, co dotychczas nie było możliwe ze względu na brak ścisłej, całościowej definicji problemu.

Co więcej, model DFA-Dynamic umożliwił kontrolę wykonania obliczeń oraz zastosowanie algorytmów sterujących wykonaniem procesów. Jednym z najprostszych sposobów szeregowania zadań dla jednego procesora w celu osiągnięcia stanu końcowego – podgrafu przedstawiającego zakończenie obliczeń – jest użycie algorytmu DFS (Deep-First Search) [78]. Dotychczasowy brak modelu DFA uniemożliwiał w pełni automatyczną ewaluację zadań oraz wiązał się z koniecznością ingerencji operatora.

Ze względu na duże wymagania pamięciowe związane z ewaluacją procesów DFA, w praktyce konieczne staje się użycie wolniejszej pamięci pomocniczej. Pomimo niezmiennika – złożoności obliczeniowej algorytmów DFA – czas zakończenia obliczeń ulega znacznemu wydłużeniu, co ściśle charakteryzują funkcje Memory Efficiency każdej z operacji. W literaturze spotkać można wiele rozwiązań, które mają na celu optymalizację użycia pamięci dla drzew obliczeń, z których godnymi uwagi są propozycje zawarte w [79] [80].

Ze względu na rozmiar problemów DFA oraz ich estymowany rozwój, użycie jednego węzła obliczeniowego stało się niewystarczające. Model DFA-Dynamic został zaprojektowany z myślą o obliczeniach rozproszonych: w niniejszym rozdziale poruszony będzie problem szeregowania zadań modelu DFA-Dynamic na wiele procesorów oraz pokazana zostanie możliwość przyspieszenia obliczeń opisanych tym modelem w przypadku użycia wielu węzłów obliczeniowych. Omówiona zostanie także możliwość zmiany ziarnistości problemu przy użyciu grafów hierarchicznych [23] [24] oraz jej wpływ na proces optymalizacji alokacji zadań. Dodatkowo wprowadzona zostanie heurystyka specyficzna dla modeli DFA, polegająca na możliwości przeprowadzenia obliczeń wektorowych dla dużej liczby scenariuszy, a tym samym skróceniu czasu ewaluacji.

6.1. Obliczenia równoległe modelu DFA

Modele DFA są zadaniami obliczeniowymi, których sekwencyjny czas wykonania estymowany jest w godzinach lub dniach, w przypadkach ekstremalnych dochodząc nawet do kilkunastu miesięcy. Powodem takiego stanu rzeczy są wysokie wymagania zarówno obliczeniowe (użycie czasu procesora), jak i pamięciowe (konieczność użycia wolnej pamięci pomocniczej). Istnieje jednak możliwość zrównoleglenia, a w szczególności rozproszenia podzadań, w celu skrócenia czasu potrzebnego na wykonanie obliczeń. Zdefiniowany w Rozdziale 1 model DFA-Dynamic jest formalizmem, dzięki któremu proces automatycznego rozproszenia zadań Dynamicznych Analiz Finansowych jest wykonalny oraz możliwa jest optymalizacja tego procesu i wykazanie jego przydatności.

Poniżej wprowadzone zostaną definicje niezbędne do formalizacji dyskusji dotyczącej optymalizacji czasu wykonania procesów DFA.

Definicja 6.1

Ścieżka prosta to ścieżka, w której żaden z wierzchołków się nie powtarza. ■

Definicja 6.2

Cykl to ścieżka o długości ≥ 1, o takim samym pierwszym i ostatnim wierzchołku. ■

Definicja 6.3

Odległość wierzchołków *v* i *w:* dist(v, w) – to długość najkrótszej ścieżki zawierającej *v* oraz *w* albo ∞, gdy nie istnieje taka ścieżka. ■

Definicja 6.4

Odległość wierzchołka *w* od zbioru wierzchołków *V*: dist(w, V) = min{ dist(w, v): v $\in$ V }. ■

Definicja 6.5

Wartość ścieżki *s* przy zadanej funkcji *f*: V(G)$\rightarrow\mathbb{R}$: $\mathrm{val}(s, f) = \sum_{v \in s} f(v)$. ■

Definicja 6.6

Ilość operacji ścieżki *s*: $\mathrm{OPP}(s) = \mathrm{val}(s, f)$, $f(x) = \begin{cases} \mathrm{OP}(v), & v.\mathrm{Label} = T \\ 0, & v.\mathrm{Label} \neq T \end{cases}$ ■

Definicja 6.7

Ilość operacji pomiędzy wierzchołkami *v* i *w* – $\mathrm{OPV}(v, w)$ – to minimum zbioru wartość OPP(s) ścieżek zawierających wierzchołki *v* i *w*. Formalnie: $\mathrm{OPV}(v, w) = \min \{ \mathrm{OPP}(s): v, w \in s \}$. ■

Definicja 6.8

Ilość operacji pomiędzy wierzchołkiem *v*, a zbiorem wierzchołków *V*: $\mathrm{OPV}(w, V) = \min_{v \in V} \{ \mathrm{OPV}(v, w) \}$. ■

Definicja 6.9

Wierzchołki aktywne grafu DFA-Dynamic – AV(G), to wierzchołki obliczeniowe będące w trakcie ewaluacji, dane gotowe do przetwarzania, miejsca decyzyjne lub punkty startowe / końcowe zaalokowane obecnie na węzłach obliczeniowych *N* poprzez wierzchołki pośrednie (ang. *proxy*). Formalnie:

$$\mathrm{AV}(G) = \{v \in V(G): v.\mathrm{Label} \in \{I, O, T, S, IS, FS\} \wedge \exists n \in V(G) \wedge n.\mathrm{Label} = N \wedge \mathrm{dist}(v, n) = 2\}.$$

Przykład: dla grafu z Rysunku 43 (strona 85) zbiór AV(G) = { I_A }; wierzchołek I_A zaalokowany jest na węźle obliczeniowym *N* poprzez wierzchołek *proxy D*. ■

Optymalizacja czasu wykonania całego modelu DFA-Dynamic, określona w Rozdziale 1 (Definicja 5.7, strona 55), jest w praktyce rzadko wykonalna ze względu na niemożliwość określenia parametrów zadań na pewnych etapie obliczeń, co zostało zasygnalizowane w Rozdziałach 1 – 1, podczas omawiania własności DFA. Z tego względu brany jest pod uwagę pewien podgraf G_{SUB} grafu $G_{DFA\text{-}Snapshot}$, dla którego istnieje możliwość wyliczenia czasu wykonania, będącego kryterium optymalizacji. Dodatkowo, wprowadzone zostały ograniczenia rozmiaru grafu, mające na celu takie zdefiniowanie problemu optymalizacyjnego, aby możliwe było jego praktyczne wykorzystanie, co zostanie omówione w dalszej części pracy. Podgraf ten przedstawia model pomiędzy stanem bieżącym, a *tymczasowymi stanami końcowymi* – wierzchołkami spełniającymi co najmniej jeden z warunków (6.1) – (6.4), oznaczającymi:

(6.1) miejsca, gdzie nie można określić w danej chwili parametrów modelu (Rysunek 43d). Przypadki tego typu, ilustrowane przykładami, zostały przedyskutowane w Rozdziale 3.2. Parametry OP (Operation) oraz DS (Data Size) wierzchołków opisane w Rozdziale 3, których nie istnieje możliwość określenia na danym etapie obliczeń są oznaczane symbolem „?”,

(6.2) stany końcowe modelu oznaczone wierzchołkami o etykiecie FS (Rysunek 43a),

(6.3) wierzchołki, dla których *ilość operacji* OPV od *wierzchołków aktywnych* AV($G_{DFA\text{-}Snapshot}$) jest większa niż OP_{MAX} operacji. Założeniem tego warunku jest, że przy średniej mocy procesora równej P_{AV} operacji na sekundę (parametr CP wierzchołków N), analizowana jest alokacja obliczeń w przedziale czasu $K \leq OP_{MAX} / P_{AV}$ godzin do przodu (Rysunek 43b). Parametr OP_{MAX} może ulegać dynamicznym zmianom tak, aby zachowany został przedział czasowy K wraz ze zmieniającym się parametrem środowiska wykonania P_{AV}, gwarantując założone ograniczenie problemu wejściowego,

(6.4) wierzchołki, dla których *odległość* DIST od *wierzchołków aktywnych* AV($G_{DFA\text{-}Snapshot}$) jest większa niż pewien a priori ustalony parametr $DIST_{MAX}$ (Rysunek 43c). Warunek ten także został wprowadzony w celu ograniczenia rozmiaru problemu optymalizacyjnego.

W praktyce efektywny algorytm wyznaczania grafu G_{SUB} polega na utworzeniu zbioru wierzchołków o etykiecie *N* (węzły obliczeniowe), znalezieniu *wierzchołków aktywnych* AV zaalokowanych na węzłach *N*, a następnie uruchomieniu algorytmu DFS (przeszukiwanie w głąb) [78] z każdego z wierzchołków aktywnych zatrzymując się, gdy osiągnięty zostanie przynajmniej jeden z wyżej opisanych warunków. Graf G_{SUB} zawiera wszystkie wierzchołki odwiedzone przez algorytm DFS (zbiór V_{SUB}) oraz krawędzie grafu G łączące dowolne wierzchołki z V_{SUB}, oprócz wierzchołków wychodzących z *tymczasowych stanów końcowych*. Formalnie:

Definicja 6.10

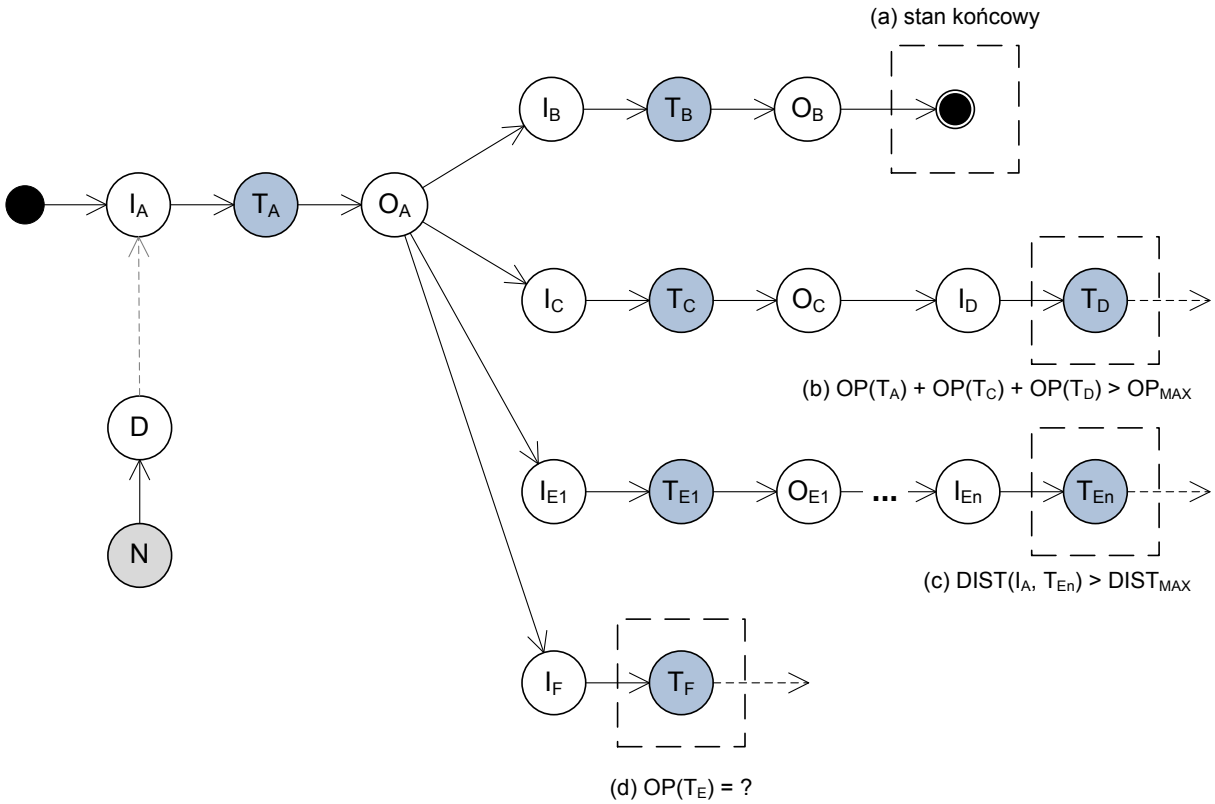
Graf $\mathbf{G_{SUB}} = (V_{SUB}, E_{SUB}) \subset G$, gdzie:

$V_{SUB} = V(G)|_{DFS\ \text{z warunkami (6.1) – (6.4)}}$

$E_{SUB} = \{ (v_1, v_2) \in E(G): v_1, v_2 \in V_{SUB} \wedge \neg v_1\text{-}tymczasowy\ stan\ końcowy \}$

■

Złożoność czasowa algorytmu DFS dla zadanego grafu G jest rzędu O(#V(G) + #E(G)). W praktyce grafy DFA-Dynamic są grafami rzadkimi, w przypadku których stosunek liczby krawędzi do liczby wierzchołków *ev* = #E(G) / #V(G) można w przybliżeniu określić rozkładem normalnym o parametrach N(1.5, 0.25). Z tego spostrzeżenia wynika wniosek, że użycie algorytmu DFS dla grafów Dynamicznych Analiz Finansowych jest w praktyce efektywne.



Rysunek 43. Wierzchołki przykładowego fragmentu grafu DFA-Snapshot wyznaczające „tymczasowe stany końcowe”.

Lemat 6.1

Dla dowolnego grafu reprezentującego model DFA, algorytm DFS nie badający następników wierzchołka, dla którego zachodzi przynajmniej jeden z warunków (6.1) – (6.4) ma własność stopu.

Dowód:

Zgodnie z definicją modelu DFA (Rozdział 3), każdy cykl zawiera wierzchołek etykietowany „S” (Selection), którego wartość atrybutu OP jest zawsze wartością oznaczoną symbolem „?”. Z warunku (6.1) wynika, że algorytm DFS nie odwiedza następników wierzchołka „S”, przez co żaden z wierzchołków nie jest odwiedzany ponownie. Zbiór wierzchołków jest zbiorem skończonym, co gwarantuje zakończenie działania algorytmu.

■

Lemat 6.2

Dowolny graf G_{SUB} nie zawiera cykli.

Dowód:

Załóżmy, że graf $G_{SUB} = (V_{SUB}, E_{SUB})$ zawiera cykl. Zgodnie z definicją modelu DFA (Rozdział 3), każdy cykl zawiera wierzchołek etykietowany „S” (Selection), który zgodnie z warunkiem (6.1) jest *tymczasowym stanem końcowym*. Na podstawie Definicji 66.10 wierzchołki należące do V_{SUB}, będące *tymczasowymi stanami końcowymi* nie posiadają krawędzi wychodzących, przez co uzyskujemy sprzeczność. Z tego wynika, że graf G_{SUB} nie zawiera cykli, CBDO.

■

Mając zadany podgraf tymczasowy G_{SUB}, możliwe staje się określenie ścisłego kryterium optymalizacji procesu alokacji zadań na węzły środowiska obliczeniowego, polegającego na minimalizacji czasu osiągnięcia wszystkich stanów końcowych. Problem alokacji optymalnej określony warunkiem z Rozdziału 1 (Definicja 5.7, strona 55), jest problemem rzędu co najmniej $O(n^t)$, gdzie n jest ilością węzłów obliczeniowych (wierzchołki etykietowane „N”), natomiast t – ilością zadań oraz danych (wierzchołki etykietowane „T”, „I” oraz „O”), co w praktyce dyskwalifikuje tego typu rozwiązanie z powodu niedopuszczalnie długiego czasu oczekiwania na odpowiedź, nawet dla grafów o stosunkowo niewielkich rozmiarach (poza trywialnym przypadkiem, w którym występuje jedynie jeden węzeł obliczeniowy). Konieczna staje się więc konstrukcja algorytmu heurystycznego, którego propozycja zostanie przedstawiona poniżej.

Definicja 6.11

Ścieżka maksymalna $S_{MAX} = (v_0, ..., v_n)$, o początku w wierzchołku v_0, to ścieżka spełniająca warunek: $\forall S = (w_0, ..., w_k): w_0 = v_0 \Rightarrow OPP(S_{MAX}) \geq OPP(S)$. ■

Szkic postępowania – będący rozwiązaniem suboptymalnym – przedstawia Algorytm 6.1, polegający na alokacji najbardziej czasochłonnej ścieżki obliczeń na najszybszy procesor, a następnie próbach poprawy czasu wykonania poprzez przeniesienie części pozostałych zadań

na inne węzły. Złożoność czasowa zaproponowanego rozwiązania jest rzędu O(#V(G_{SUB}) + #E(G_{SUB})), co zostanie omówione podczas analizy algorytmu.

Algorytm 6.1. Heurystyczny algorytm alokacji zadań problemu opisanego grafem G_{SUB}.

```
1) Wybierz dowolną ścieżkę maksymalną S_MAX o początku w wierzchołku aktywnym
2) Zaalokuj ścieżkę maksymalną S_MAX = (v_0, ..., v_n) i dokonaj migracji wierzchołka v_0
   na najmocniejszy procesor, jeżeli zmniejszy się czas osiągnięcia v_n
3) Dopóki istnieje niezaalokowane zadanie T i niezajęty procesor
4)     Wybierz dowolną ścieżkę S zawierającą zadanie T
5)     Podejmij decyzję o przeniesieniu podścieżki S' ⊂ S na najmocniejszy niezajęty
       procesor N, jeżeli zmniejszy to czas osiągnięcia stanów końcowych
```

W kroku 1) (Algorytm 6.1) wybierana jest dowolna *ścieżka maksymalna*, zaczynająca się w (także dowolnym) *wierzchołku aktywnym*. W tym celu każdemu wierzchołkowi przypisywany jest parametr, określający rozmiar pozostałych do wykonania obliczeń. Formalnie, dla każdego wierzchołka $n \in V(G_{SUB})$ obliczana jest wartość *branchParameters(n)* według wzoru (6.5), gdzie *branchParameters(n)* $\in P_1 \times \ldots \times P_i$, natomiast P_j jest zbiorem wartości parametru *j*. Rozpatrywanymi parametrami mogą być na przykład atrybuty wierzchołków o etykiecie T, takie jak Processor Consumption, Memory Consumption lub Memory Efficiency, mające znaczenie przy określaniu czasu zakończenia obliczeń oraz atrybut History każdego z wierzchołków, przechowujący informację dotyczącą wcześniejszych ewaluacji modelu.

$$\text{branchParameters}(n) = f_{\text{branchParameters}}^{k}[\text{parameters}(n), branchParameters(m_1), \ldots, branchParameters(m_k)] \quad (6.5)$$

gdzie $\{m_1, \ldots, m_k\}$ to zbiór następników węzła n

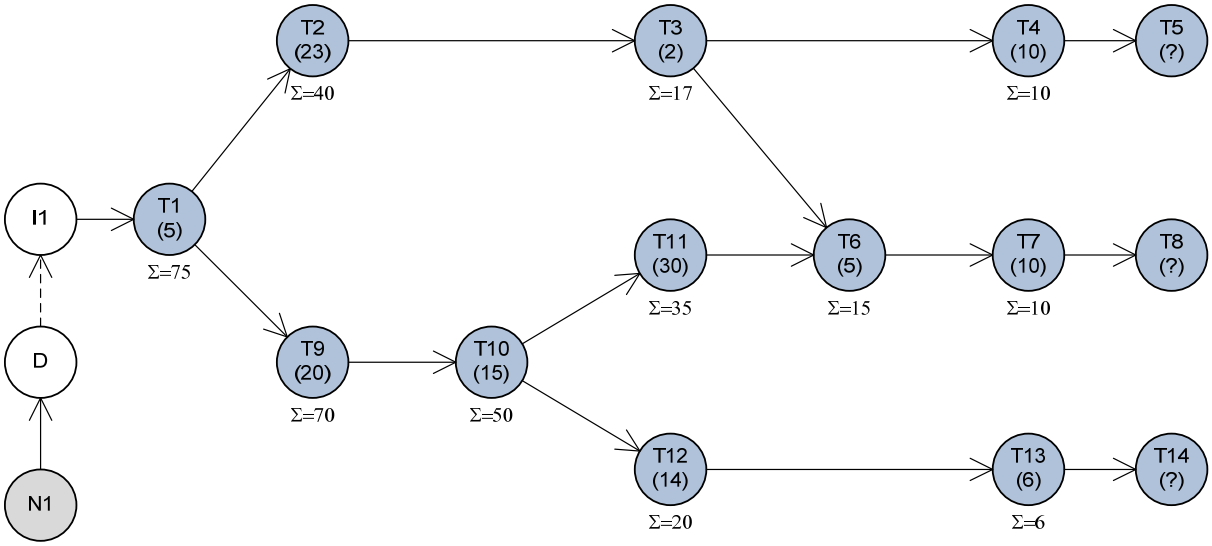
Funkcje ze zbioru { $f_{branchParameters}^{k}$ } – gdzie *k* jest liczbą następników przetwarzanego węzła *n* – określają parametry gałęzi dla wierzchołka *n* w zależności od wartości parametrów zadanego węzła oraz wartości parametrów gałęzi dla następników węzła *n*.

Dla zobrazowania zagadnienia na Rysunku 44 przedstawiono przykładowy graf $G_{SUB}|_T$, będący uproszczoną formą G_{SUB} z pominiętymi wierzchołkami danych wejściowych i wyjściowych, a także z dodanymi wierzchołkami o nieokreślonych parametrach oraz z wierzchołkiem aktywnym z miejscem alokacji. Za wartości funkcji *branchParameters* przyjęto zbiór liczb rzeczywistych, natomiast funkcja $f_{branchParameters}^{k}$ została określona według wzoru (6.6), gdzie CP to wartość Computing Power zadanego węzła. Dla zadanego przykładu wartość *branchParameters* węzła *n* jest *ilością operacji ścieżki maksymalnej* OPP(S_{MAX}) zaczynającej się w tym węźle.

$$f_{branchParameters}{}^{k}(n) = CP(n) + \max_{i=1..k} \{ branchParameters(m_i) \} \quad (6.6)$$

gdzie

$\{m_1, \ldots, m_k\}$ to zbiór następników węzła n



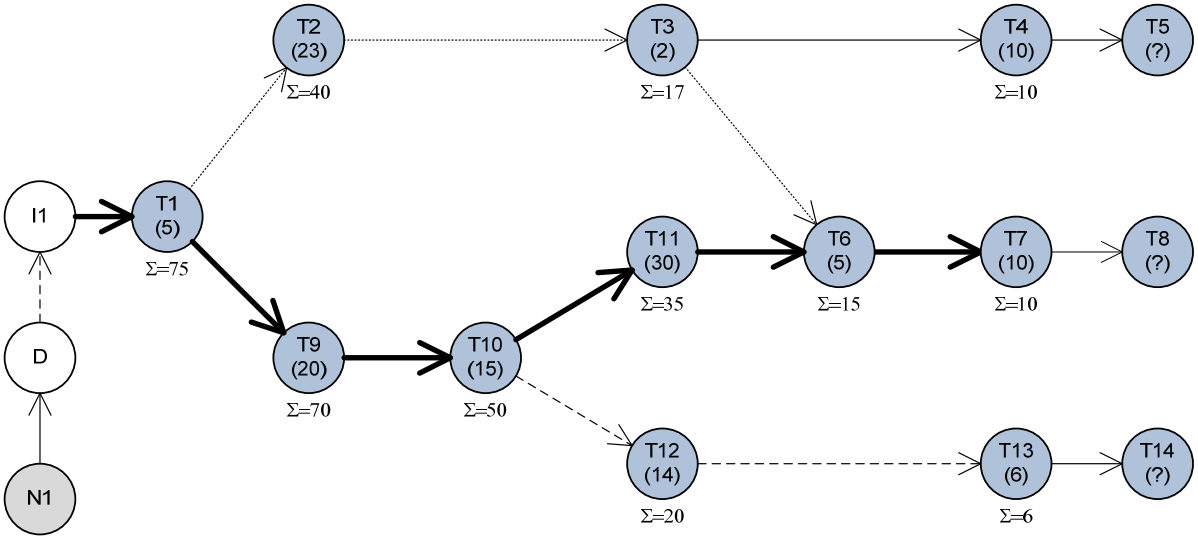
Rysunek 44. Przykładowy graf $G_{SUB}|_T$ z wartościami funkcji $f_{branchParameters}$ oznaczonymi symbolem Σ.

Efektywny sposób wyznaczania wartości *branchParameters* polaga na uruchomieniu algorytmu BFS (przeszukiwanie wszerz) [78] z następującymi modyfikacjami:

- każdy wierzchołek posiada parametr *successorCount* o wartości pierwotnej równej ilości następników,
- inicjalizacja kolejki Q (wierzchołki do odwiedzenia) polega na dodaniu wierzchołków grafu G_{SUB} nie posiadających następnika,
- po odwiedzeniu wierzchołka *v*, wartość parametru *successorCount* każdego poprzednika jest pomniejszana o 1; do kolejki Q dodawani są tylko ci poprzednicy *v*, dla których wartość *successorCount* równa jest 0.

Złożoność czasowa oraz pamięciowa przedstawionego algorytmu dla zadanego grafu G jest rzędu O(#V(G) + #E(G)), co kwalifikuje to rozwiązanie jako użyteczne w praktyce. Graf G_{SUB} nie posiada cykli (Lemat 6.2, strona 86), co gwarantuje, że algorytm posiada własność stopu.

Po określeniu wartości parametru *branchParameters* dla każdego wierzchołka, możliwe jest wyznaczenie *ścieżki maksymalnej*, polegające na wybraniu spośród wierzchołków aktywnych dowolnego, o maksymalnej wartości wspomnianego parametru oraz kolejnych następników, także o wartościach maksymalnych (Rysunek 45). Złożoność czasowa wyboru ścieżki maksymalnej dla danego grafu G jest rzędu O(#V(G) + #E(G)).



Rysunek 45. Ścieżka maksymalna (pogrubiona, T1-T7), ścieżka powracająca (kropkowana, T1-T6) oraz ścieżka niepowracająca (kreskowana, T10-T13) przykładowego grafu G_C.

W przedstawionym przykładzie ilość pracy pozostała do wykonania mierzona jest wartością parametru Processor Consumption wierzchołków „T", lecz w bardziej skomplikowanych przypadkach – jak wcześniej wspomniano – pod uwagę mogą być brane takie wartości, jak na przykład średnia lub maksymalna ilość pamięci potrzebna do przeprowadzenia obliczeń lub w pewien sposób uśredniona charakterystyka funkcji Memory Efficiency.

W kroku 2) (Algorytm 6.1) testowana jest opłacalność migracji ścieżki maksymalnej $S_{MAX} = (v_0, ..., v_n)$ na najsilniejszy węzeł, o ile nie jest ona obecnie zaalokowana na tym węźle.

Definicja 6.12

Najsilniejszy węzeł obliczeniowy N_{MAX}, to wierzchołek taki, że $\forall v \in V(G)$: v.label = „N" $\wedge$ $f_{POW}(v) \leq f_{POW}(N_{MAX})$.

Funkcja f_{POW}: $V(G) \rightarrow \mathbb{R}$ wyznacza moc obliczeniową węzła, a jej wartość określają parametry CP (Computing Power) i AM (Available Memory). Może być zdefiniowana w następujący sposób:

$$f_{POW}(v) = v.CP + \frac{v.AM}{v.AM + 1} \tag{6.7}$$

W tym przypadku w pierwszej kolejności pod uwagę brana jest moc obliczeniowa zadanego węzła, a następnie ilość dostępnej pamięci.

■

Formalnie, dla danej ścieżki $S_{MAX} = (T, v_1, ..., v_n)$ i zadania T przypisanego do węzła N_{CUR}, ścieżka S_{MAX} przypisywana jest do dowolnego ustalonego *najsilniejszego węzła obliczeniowego* N_{MAX}, gdy zachodzi poniższy warunek:

$$\sum_{D \in inD_T} T_{TR}(D: N_{CUR} \rightarrow N_{MAX}) + \sum_{T \in S_{MAX}} T_{EV}(T @ N_{MAX}) < \sum_{T \in S_{MAX}} T_{EV}(T @ N_{CUR}) \tag{6.8}$$

gdzie inD_T, to zbiór węzłów reprezentujących dane wejściowe zadania T:

$$inD_T = \{ v \in V(G): \exists (v, T) \in E(G) \}$$

W krokach 3) – 5) (Algorytm 6.1) podejmowana jest decyzja o przeniesieniu pozostałych, niezaalokowanych ścieżek na inne węzły środowiska obliczeniowego w celu poprawy czasu zakończenia obliczeń.

W pierwszej kolejności tworzona jest lista węzłów obliczeniowych, do których nie zostało przypisane żadne zadanie i wybierany jest *najsilniejszy węzeł obliczeniowy* N_M według wcześniej wprowadzonego kryterium (6.7). Jeżeli istnieje niezajęty węzeł N_M, wybierana jest dowolna ścieżka $S = (v_0, ..., v_n)$ grafu G_{SUB} o długości ≥ 2 taka, że zachodzi (6.9) $\wedge$ [(6.10) $\vee$ (6.11)] $\wedge$ (6.12).

- v_0 należy do już zaalokowanej ścieżki (oznaczmy węzeł alokacji symbolem N_0), (6.9)

- v_n należy do już zaalokowanej ścieżki (oznaczmy węzeł alokacji symbolem N_n), (6.10)

- v_n nie należy do zaalokowanej ścieżki i v_n nie posiada następnika w grafie G_{SUB}, (6.11)

- $\forall i: 0 < i < n$, v_i nie należy do zaalokowanej ścieżki, (6.12)

Wybierana jest więc ścieżka, której pierwszy element jest już przypisany do pewnego procesora, elementy „środkowe" nie zostały jeszcze zaalokowane, natomiast ostatni element jest także przypisany do jakiegoś procesora lub nie posiada następnika.

Definicja 6.13

Ścieżka $S = (v_0, ..., v_n)$ jest **ścieżką powracającą** (przykład: Rysunek 45, węzły T1-T2-T3-T6), jeżeli zachodzą warunki (6.9) $\wedge$ (6.10) $\wedge$ (6.12).

■

Definicja 6.14

Ścieżka $S = (v_0, ..., v_n)$ jest **ścieżką niepowracającą** (przykład: Rysunek 45, węzły T10-T12-T13), jeżeli zachodzą warunki (6.9) $\wedge$ (6.11) $\wedge$ (6.12).

■

W przypadku *ścieżek powracających*, dla których zachodzi konieczność migracji danych wynikowych na określony węzeł, dla znalezienia optymalnych punktów migracji konieczne jest zbadanie wszystkich przedziałów $(v_i, ..., v_j) \subset S = (v_0, ..., v_n)$, co wiąże się z kosztem $O(n^2)$, gdzie *n* jest długością rozpatrywanej ścieżki. Kryterium opłacalności migracji przedziału

(v_i, ..., v_j) = S' ⊂ S na węzeł N_M (przy założeniu, że S jest *ścieżką powracającą*, której v_0 został zaalokowany na N_0, natomiast v_n na N_n), zostało zdefiniowane warunkiem (6.13):

$$\sum_{D \in inD_i} T_{TR}(D: N_0 \rightarrow N_M) + \sum_{T \in S'} T_{EV}(T @ N_M) + \sum_{D \in inD_j} T_{TR}(D: N_M \rightarrow N_n) < 2 * \sum_{T \in S'} T_{EV}(T @ N_0) + \sum_{D \in inD_n} T_{TR}(D: N_0 \rightarrow N_n) \quad (6.13)$$

gdzie inD_k, to zbiór węzłów reprezentujących dane wejściowe zadania T_k:

$$inD_k = \{ v \in V(G): \exists(v, T_k) \in E(G) \}$$

Warunek (6.13) określa, czy koszt migracji danych do i z węzła N_M jest rekompensowany odciążeniem węzła N_0, na którym pierwotnie miałaby odbywać się ewaluacja obliczeń.

Przedział optymalny to taki, który spełnia warunek (6.13), a dla którego wartość funkcji f_{OP}: (v_i, ..., v_j) → ℝ określonej wzorem (6.14) jest maksymalna.

$$f_{OP}(S) = \sum_{T \in S} T.OP \quad (6.14)$$

W przypadku *ścieżek niepowracających* nie istnieje konieczność migracji danych na węzeł obliczeniowy N_n, na którym zaalokowane zostało już wcześniej ostatnie zadanie. Z tego względu znalezienie optymalnego punktu migracji przedziału (v_i, ..., v_n) ⊂ S = (v_0, ..., v_n) wymaga sprawdzenia O(n) przypadków. Kryterium opłacalności migracji przedziału (v_i, ..., v_n) = S' ⊂ S na węzeł N_M (przy założeniu, że S jest *ścieżką niepowracającą*, której v_0 został zaalokowany na N_0), definiuje warunek (6.15):

$$\sum_{D \in inD_i} T_{TR}(D: N_0 \rightarrow N_M) + \sum_{T \in S'} T_{EV}(T @ N_M) < 2 * \sum_{T \in S'} T_{EV}(T @ N_0) \quad (6.15)$$

gdzie inD_k, to zbiór węzłów reprezentujących dane wejściowe zadania T_k:

$$inD_k = \{ v \in V(G): \exists(v, T_k) \in E(G) \}$$

W przypadku *ścieżek niepowracających*, przedział optymalny to taki, który spełnia warunek (6.15) oraz maksymalizuje wartość funkcji określonej wzorem (6.14).

Znajdowanie przedziałów optymalnych – szczególnie dla *ścieżek powracających* – jest zadaniem stosunkowo kosztownym. Operację tą należy powtórzyć dla każdego testowanego przedziału, co powoduje, że złożoność czasowa dla grafu G_{SUB} jest rzędu $O(n^3)$, gdzie n = #V(G_{SUB}) + #E(G_{SUB}). Przeprowadzone doświadczenia wykazały, że dla grafów definiujących

modele DFA, ustalenie przedziału migracji na (v_2, ..., v_{n-1}) dla *ścieżek powracających* oraz (v_2, ..., v_n) dla *ścieżek niepowracających* nie pogarsza zauważalnie czasu zakończenia obliczeń. Jednocześnie dla obydwu przypadków złożoność czasowa wyboru przedziału znacząco spada i jest rzędu O(1). Przy nałożeniu tego dodatkowego warunku, złożoność czasowa kroków 3) – 5) Algorytmu 6 jest rzędu O(#V(G_{SUB}) + #E(G_{SUB})).

Podsumowując, kroki 1) – 5) w rezultacie kształtują złożoność czasową, a także pamięciową Algorytmu 6.1 rzędu O(#V(G_{SUB}) + #E(G_{SUB})). Jak wspomniano wcześniej, w praktyce grafy modelu DFA-Dynamic są grafami rzadkimi, w przypadku których stosunek liczby krawędzi do liczby wierzchołków *ev* = #E(G) / #V(G) można w przybliżeniu określić rozkładem normalnym o parametrach N(1.5, 0.25). Z rozważań teoretycznych wynika wniosek, że użycie Algorytmu 6.1 do optymalizacji alokacji zadań dla grafów Dynamicznych Analiz Finansowych jest w praktyce efektywne.

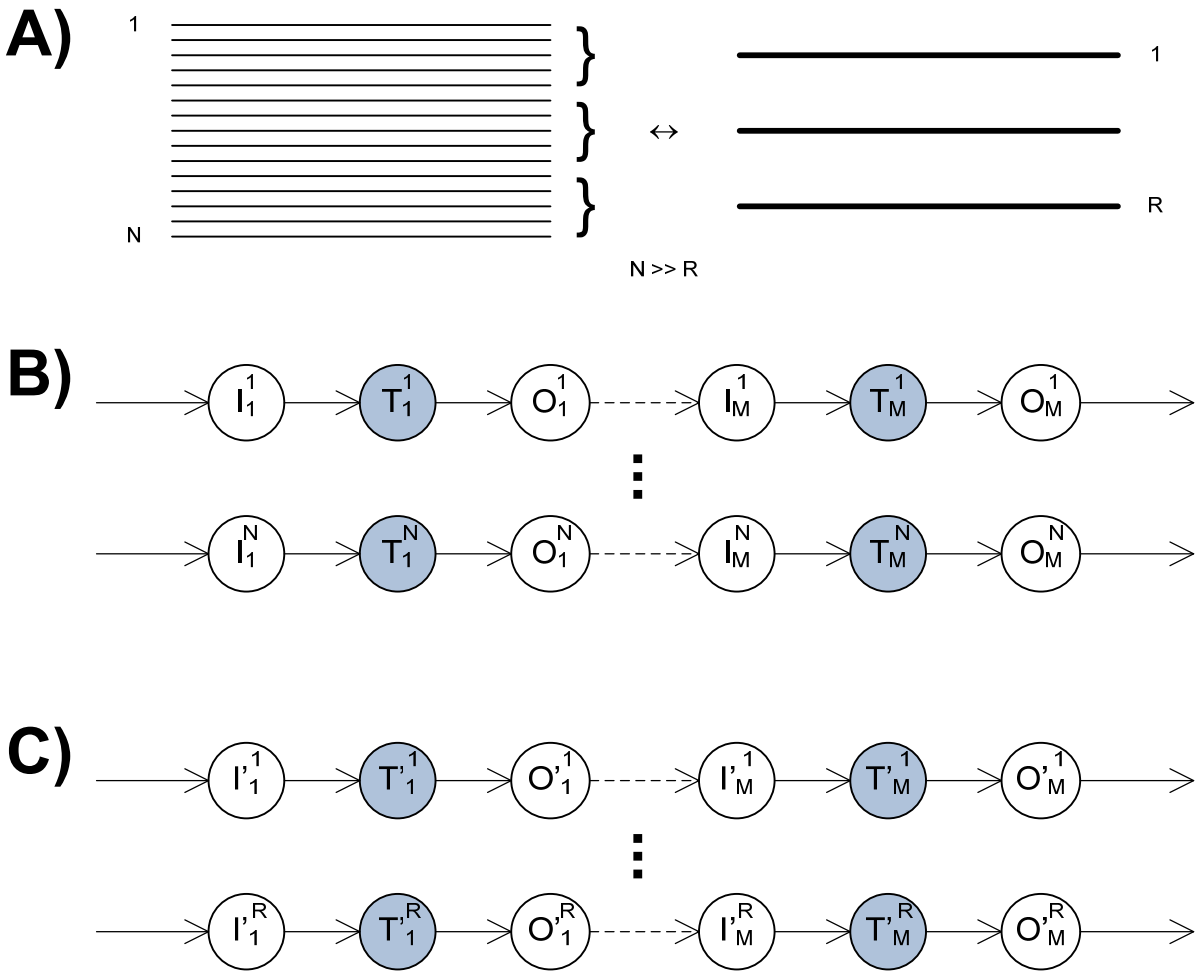
6.2. Proces scalania scenariuszy

W modelach DFA przetwarzana jest duża ilość scenariuszy, sięgająca rzędu 10^6. Coraz większe wymagania użytkowników związane ze zmniejszeniem błędu symulacji dążą do jak największego przesunięcia tej granicy w górę. Każdy ze scenariuszy jest na wielu etapach sekwencją takich samych operacji, wykonywanych na różnych danych wejściowych. W związku z tym istnieje możliwość wektoryzacji obliczeń, a tym samym zmniejszenia nakładów potrzebnych na przetwarzanie i alokację zadań. Wektoryzacja obliczeń polega na zastąpieniu łańcuchów takich samych operacji przetwarzających różne zbiory danych o takiej samej wielkości jednym łańcuchem operacji zastępczych, operujących na wektorze danych (będącym sumą danych z kolejnych łańcuchów), dającym analogiczne wyniki. Rysunek 46A przedstawia *N* łańcuchów operacji, które są grupowane, a następnie zamieniane na *R* łańcuchów operacji wektorowych. Rysunek 46B przedstawia szczegółowo elementy łańcuchów 1-*N*, natomiast Rysunek 46C – łańcuchów 1-*R* i posłuży jako ilustracja omawianego zagadnienia.

Praktycznym rezultatem transformacji wektoryzacji jest:

- znaczne zmniejszenie liczby wierzchołków grafu modelu DFA-Static, co pociąga za sobą zmniejszenie rozmiaru problemu alokacji zadań obliczeniowych na procesory, a tym samym przyspieszenie tego procesu,
- zauważalna w praktyce znacząca redukcja mocy obliczeniowej potrzebnej do uzyskania wyników, formalnie określona wzorem (6.16). Podobne wyniki obserwowane są w innych zastosowaniach – na przykład wektorowym przetwarzaniu operacji graficznych, czego rezultatem było wprowadzenie przez firmę Intel zestawu instrukcji SIMD znanych pod nazwą MMX oraz ich kolejnych generacji,
- zwiększenie ziarnistości (omówione szczegółowo w następnym rozdziale), którego – teoretycznie – konsekwencją jest pogorszenie procesu optymalizacji alokacji zadań. Jednak w praktyce dobranie wartości parametru *R* na poziomie ilości dostępnych procesorów skutecznie niweluje wspomniany problem do praktycznie niezauważalnych różnic.

$$K \gg R \Longrightarrow \sum_{p=1}^{K} \sum_{m=1}^{M} T_m^p . CP \gg \sum_{r=1}^{R} \sum_{m=1}^{M} T'^r_m . CP \qquad (6.16)$$



Rysunek 46. Ilustracja procesu scalania scenariuszy. A) Szkic zagadnienia. B) Przykładowe łańcuchy obliczeń przed scaleniem scenariuszy. C) Przykładowe łańcuchy obliczeń po scaleniu scenariuszy.

Formalnie, proces wektoryzacji można określić w następujący sposób: dla zbioru wierzchołków $\{ I_l^k , O_l^k , T_l^k \}\ k = 1..K, l = 1..M$ (Rysunek 46B) takiego, że zachodzi warunek (6.17), istnieje możliwość stworzenia transformacji redukującej liczbę wierzchołków oraz zachowującej wartości danych. Rezultatem procesu wektoryzacji jest transformacja grafowa, przekształcająca podgraf z Rysunku 46B, na podgraf z Rysunku 46C.

$$\forall p,q = 1..K \;\; \forall m = 1..M \;\; \forall n = 1..M-1:$$

$$\begin{aligned} & I_m^p.Label = \text{"}I\text{"} \wedge I_m^p.DataSize = \; I_m^q.DataSize \\ \wedge\; & O_m^p.Label = \text{"}O\text{"} \wedge O_m^p.DataSize = \; O_m^q.DataSize \\ \wedge\; & T_m^p.Label = \mathrm{T} \wedge T_m^p.Operation = T_m^q.Operation \\ \wedge\; & \forall I_m^p \; \exists!\, e \in E(G): e = \left(I_m^p, v\right) \Longrightarrow v = T_m^p \\ \wedge\; & \forall T_m^p \; \exists!\, e \in E(G): e = \left(v, T_m^p\right) \Longrightarrow v = I_m^p \\ \wedge\; & \forall T_m^p \; \exists!\, e \in E(G): e = \left(T_m^p, v\right) \Longrightarrow v = O_m^p \\ \wedge\; & \forall O_m^p \; \exists!\, e \in E(G): e = \left(v, O_m^p\right) \Longrightarrow v = T_m^p \\ \wedge\; & \forall O_n^p \; \exists!\, e \in E(G): e = \left(O_n^p, v\right) \Longrightarrow v = I_{n+1}^p \\ \wedge\; & \forall I_{n+1}^p \; \exists!\, e \in E(G): e = \left(v, I_{n+1}^p\right) \Longrightarrow v = O_n^p \end{aligned} \tag{6.17}$$

W praktyce przeszukiwanie wszystkich pozdzbiorów wierzchołków grafu G, dla których dodatkowo konieczne jest sprawdzanie warunku (6.17), jest problemem co najmniej wykładniczym. Na podstawie analizy grafów reprezentujących modele DFA zaproponowano rozwiązanie alternatywne (Algorytm 6.2), o złożoności obliczeniowej umożliwiającej jego praktyczne zastosowanie. Do jego konstrukcji wykorzystano następujące spostrzeżenie: łańcuchy operacji spełniające warunek (6.17) zaczynają się w tym samym wierzchołku (Rysunek 47, strona 95).

Algorytm 6.2. Wyszukiwanie łańcuchów scenariuszy zaczynających się w tym samym wierzchołku.

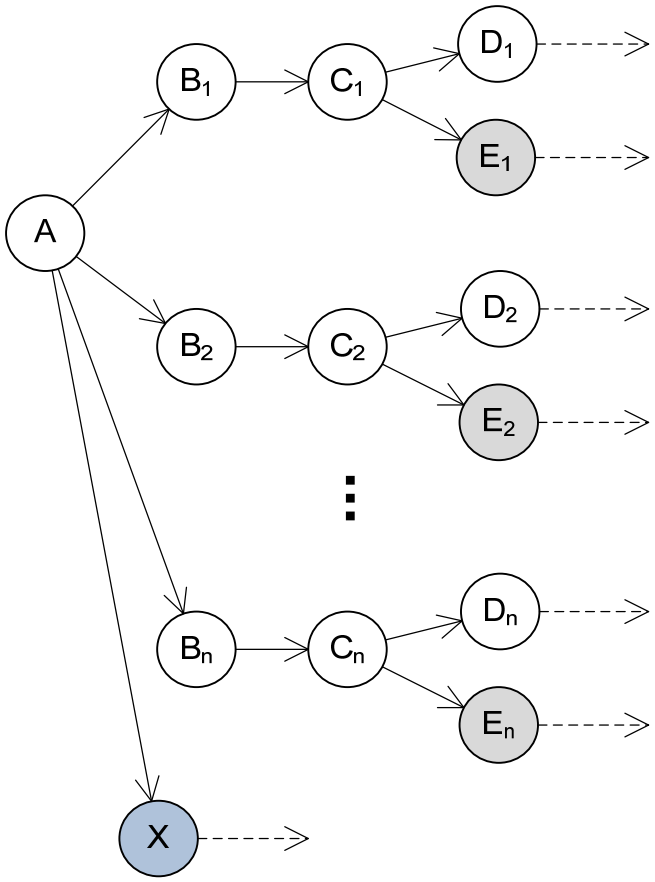
```
1)  Wygeneruj zbiór V_TEST = { v ∈ V(G): #{ (v, w) ∈ E(G) } ≥ minSuccessorCount }
2)  Dla każdego v ∈ V_TEST
3)      Używając algorytmu BFS sprawdź kompatybilność wierzchołków jednakowo odległych od v
```

W kroku 1) Algorytmu 6.2 zawężany jest zbiór wierzchołków do tych, które posiadają co najmniej *minSuccessorCount* następników. Zabieg ten nie zmienia ogólnej złożoności czasowej algorytmu, lecz ma niebagatelne znaczenie w praktyce; dla grafów reprezentujących modele DFA zaobserwowano, że $\#V_{TEST} << \#V(G)$, przy odpowiednio dobranej wartości parametru *minSuccessorCount*. Złożoność czasowa tworzenia zbioru V_{TEST} jest rzędu O(#V(G) + #E(G)).

W kroku 3) wykorzystywany jest algorytm BFS (przeszukiwanie wszerz) w celu weryfikacji, czy wszystkie wierzchołki jednakowo odległe od wierzchołka startowego są tego samego typu oraz posiadają takie same parametry. Złożoność czasowa tego kroku jest również rzędu O(#V(G) + #E(G)). W bardziej złożonych przypadkach, algorytm wyszukiwania łańcuchów może wyznaczać wierzchołki spoza łańcucha (Rysunek 47, wierzchołek „X"), a także łańcuchy z odgałęzieniami (Rysunek 47, wierzchołki „E_i"), przy niezmienionej złożoności czasowej.

Ze względu na ograniczony rozmiar niniejszej rozprawy, omówienie tych przypadków przewidziane jest jako przedmiot przyszłych prac.

Podsumowując, złożoność czasowa Algorytmu 6 jest rzędu O(p * q), gdzie p = #V(G), natomiast q = #V(G) + #E(G). Jak na wstępie niniejszego rozdziału zaznaczono, ilość scenariuszy może dochodzić (a także niekiedy przekraczać) wielkość 10^6, co implikuje: #V(G) > 10^6. Złożoność co najmniej sześcienna zaproponowanego rozwiązania (ze względu na ilość wierzchołków grafu) potencjalnie dyskwalifikowałaby jego praktyczną użyteczność, dlatego też wprowadzona została wstępna filtracja testowanych wierzchołków – krok 1) Algorytmu 6, dzięki której wartość *p* jest znacząco pomniejszona: $p = \#V_{TEST} << \#V(G)$. Jak wcześniej stwierdzono, dla modeli DFA stosunek *ev* = #E(G) / #V(G) można w przybliżeniu określić rozkładem normalnym o parametrach N(1.5, 0.25). Z tego też względu w praktyce wartość parametru q = #V(G) + #E(G) rośnie liniowo w stosunku do liczby wierzchołków grafu. Na podstawie powyższych rozważań wynika, że przedstawione rozwiązanie jest efektywne w praktyce, co zapewnia jego użyteczność.



Rysunek 47. Łańcuchy operacji zaczynające się w tym samym wierzchołku; wierzchołek spoza łańcucha (X); wierzchołki należące do łańcuchów z rozgałęzieniami (E_i).

Reasumując, proces wektoryzacji scala sekwencje takich samych operacji w jeden (lub kilka) łańcuchów operacji zastępczych operujących na wektorze danych i dających w rezultacie analogiczne wyniki. Grafem wejściowym procesu wektoryzacji może być zarówno model DFA-Snapshot,

jak i DFA-Static. Najczęściej przetwarzaniu wstępnemu (ang. *pre-processing*) poddawany jest graf DFA-Static, jeszcze przed fazą obliczeń właściwych, czego głównymi konsekwencjami są:

- znaczne zmniejszenie liczby wierzchołków grafu modelu, pociągające za sobą zmniejszenie rozmiaru problemu alokacji zadań obliczeniowych na procesory,
- znacząca redukcja mocy obliczeniowej potrzebnej do uzyskania wyników, a co za tym idzie przyspieszenie całego procesu obliczeniowego.

6.3. Zmiana ziarnistości modelu DFA

W Rozdziale 3.3 wprowadzone zostało pojęcie komponentu, bazujące na publikacji (Kotulski) [22], bytu funkcjonalnie podobnego do zadania (wierzchołki etykietowane „T"), lecz o złożonej strukturze wewnętrznej, będącej de facto modelem DFA. Komponentyzacja ma za zadanie podział wierzchołków grafu modelu DFA na podzbiory, a następnie zastąpienie każdego z podzbiorów jednym wierzchołkiem, przy jednoczesnym zachowaniu informacji o strukturze pierwotnej, zakładając możliwość operacji odwrotnej. Proces komponentyzacji może być dokonywany wielokrotnie, tworząc kolejne poziomy zagłębienia.

Komponenty mogą być traktowane jako pewnego rodzaju obiekty izolowane (tzw. czarne skrzynki), zawierające pewny ciąg operacji do wykonania. Możliwe staje się operowanie nie na pojedynczych operacjach zawartych w komponentach, lecz na całych komponentach. Dzięki takiemu zabiegowi rozmiar problemu wejściowego operującego na grafie ulega logarytmicznemu zmniejszeniu.

Definicja 6.15

Ziarnistość modelu DFA to określenie, ile razy przeprowadzony został proces komponentyzacji na grafie pierwotnym. **Zmniejszenie ziarnistości** to uruchomienie procesu komponentyzacji na grafie bieżącym, natomiast **zwiększenie ziarnistości** to przeprowadzenie procesu odwrotnego.

■

Zmniejszenie liczby wierzchołków grafu poddawanego analizie w celu optymalizacji czasu wykonania procesów opartych na modelu DFA w środowisku rozproszonym ma niebagatelny wpływ na szybkość przeprowadzania procesu optymalizacji. W praktyce liczba wierzchołków modelu DFA może dochodzić do rzędu 10^8. Zakładając, że zmniejszenie ziarnistości o jeden poziom powoduje zmniejszenie wielkości problemu o jeden rząd wielkości, praktyczny zysk redukcji rozmiaru zadania wejściowego ma ogromne znaczenie dla wydajności działania omawianego procesu. Ponadto, wyznaczanie komponentów dokonywane jest jednokrotnie w oparciu o graf modelu DFA-Static, przed rozpoczęciem obliczeń właściwych. Zysk z dokonania podziału daje ogromne korzyści, szczególnie w przypadku zadań uruchamianych z dużą częstotliwością.

Jednocześnie odpowiednio ustalona ziarnistość problemu wejściowego nie ma większego wpływu na czas zakończenia obliczeń. Dla przykładu: model DFA, którego sekwencyjny czas wykonania liczony jest w tygodniach, na najniższym poziomie ziarnistości składa się z zadań o czasie ewaluacji rzędu milisekund. Jednakże próba nawet suboptymalnej alokacji tak małych zadań mija się

z celem, gdyż potencjalny zysk – na przykład w porównaniu z alokacją zadań o czasie wykonania jednej godziny – zostanie niezauważony (w praktyce jest bez większego znaczenia). Co więcej, bilans takiego działania może mieć ujemne rezultaty, gdyż czas zużyty na działanie algorytmu alokacji może pochłonąć znacznie więcej zasobów obliczeniowych.

Ponadto, zmiana ziarnistości problemu może odbywać się dynamicznie, w trakcie działania procesu ewaluacji. Przykładem jest uzależnienie poziomu ziarnistości od ilości pracy pozostałej do wykonania. Gdy pozostała praca liczona jest w tygodniach, ziarnistość komponentów pozostaje na poziomie dni, następnie: pozostała praca – dni / komponenty – godziny; pozostała praca – godziny / komponenty – minuty. Formalnie: gdy spełniony jest warunek (6.18), bazujący na parametrze PC (Processor Consumption) wierzchołków etykietowanych T (klasy zadań). Dynamiczny podział gwarantuje „nieprzeciążenie" algorytmu optymalizacji alokacji, który przy nadmiernym rozdrobnieniu problemu sam może stać się głównym celem obliczeń i zarazem wąskim gardłem całego systemu.

$$(\mathrm{PC_{AV}} > \mathrm{k_{PC}} * \mathrm{PC_{SUM}}) \wedge (\mathrm{PC_{SUM}} > \mathrm{min_{PC}}) \tag{6.18}$$

dla penych, ustalonych $\mathrm{k_{PC}}$ oraz $\mathrm{min_{PC}}$

gdzie dla $\{\mathrm{T_{1..n}}\} = \mathrm{V(G)}|_{\mathrm{v.label=T}}$:

$$\mathrm{PC_{AV}} = \frac{\sum_{i=1}^{n} \mathrm{T_i}.\mathrm{PC}}{n}$$

$$\mathrm{PC_{SUM}} = \sum_{i=1}^{n} \mathrm{T_i}.\mathrm{PC}$$

Podział modelu statycznego DFA na komponenty może zostać dokonany na dwa sposoby: manualnie lub automatycznie. Sposób manualny polega na wymuszeniu podziału przez projektanta, który kieruje się wiedzą aprioryczną, doświadczeniem oraz dążeniem do osiągnięcia założonych celów. Na kryterium podziału mają między innymi wpływ takie czynniki jak:

- założenie wielokrotnej ewaluacji pewnego fragmentu modelu,
- przewidywana (niekoniecznie trafnie) wielkości danych wejściowych oraz wyjściowych,
- brak zainteresowania danymi pośrednimi oraz wymuszenie zakończenia obliczeń na pewnym etapie przed przejściem do ewaluacji innej gałęzi grafu. Dzięki temu warunkowi możliwy jest szybszy podgląd części wyników, przed zakończeniem wykonania całego modelu.

Podział automatyczny może zostać dokonany na podstawie:

- rozmiaru danych wyjściowych w rozpatrywanym podzbiorze wierzchołków,
- częstości użycia danych wyjściowych pewnego podzbioru wierzchołków przez zadania spoza tego podzbioru.

Algorytm 6.3. Szkic algorytmu podziału wierzchołków grafu modelu DFA-Static na komponenty.

```
1) Wygeneruj zbiór O_LG = { v ∈ V(G): v.label = „O” ∧ v.DS > DS_MIN }
2) Dla każdego o ∈ O_LG ∧ o.notVisited
3)     Uruchom DFS dla następników i poprzedników o zatrzymując się, gdy v.DS < DS_MIN
4)     Utwórz komponent z wierzchołków odwiedzonych przez DFS
```

Algorytm 6.3 przedstawia propozycję automatycznego podziału wierzchołków grafu G na komponenty. Zasada działania polega na połączeniu sąsiednich zadań wraz z danymi wejściowymi / wyjściowymi w przypadku, gdy rozmiar przetwarzanych przez nich danych jest większy od założonej wartości. Tego typu podział na komponenty ma na celu zapewnienie, że zadania wymieniające między sobą znaczne ilości danych zostaną zaalokowane na tym samym węźle obliczeniowym, natomiast miejsca migracji zadań zostaną ustalone tam, gdzie ilość wymienianych danych jest odpowiednio mniejsza.

W kroku 1) generowany jest zbiór wierzchołków danych wyjściowych, których rozmiar przekracza założoną wartość DS_{MIN}. Ustalenie DS_{MIN} może odbywać się na podstawie wartości średniej rozmiarów danych: $DS_{MIN} = k_{DS} * DS_{AV}$, gdzie DS_{AV} jest wartością średnią rozmiaru danych wyjściowych w grafie G, natomiast k_{DS} pewnym założonym współczynnikiem.

W kroku 3) uruchamiane jest przeszukiwanie DFS (przeszukiwanie w głąb) odwiedzające wierzchołki sąsiednie do momentu, gdy wartość v.DS (rozmiar danych) spadnie poniżej założonej wartości DS_{MIN}.

Krok 4) tworzy komponent z podzbioru wierzchołków odwiedzonych przez algorytm DFS. W rezultacie podzbiór ten składa się z zadań „intensywnie wymieniających” pomiędzy sobą dane.

Złożoność czasowa Algorytmu 6.3 jest rzędu O(#V(G) + #E(G)) pomimo, że pętla w kroku 2) jest złożoności O(#V(G)), a algorytm DFS w kroku 3) złożoności O(#V(G) + #E(G)). Finalny koszt zamortyzowany jest rezultatem faktu, że w kroku 3) DFS odwiedza wierzchołki co najwyżej jednokrotnie, dzięki warunkowi o.notVisited wspomnianej pętli. Oszacowana złożoność czasowa kwalifikuje zaproponowany algorytm jako przydatny do praktycznego wykorzystania.

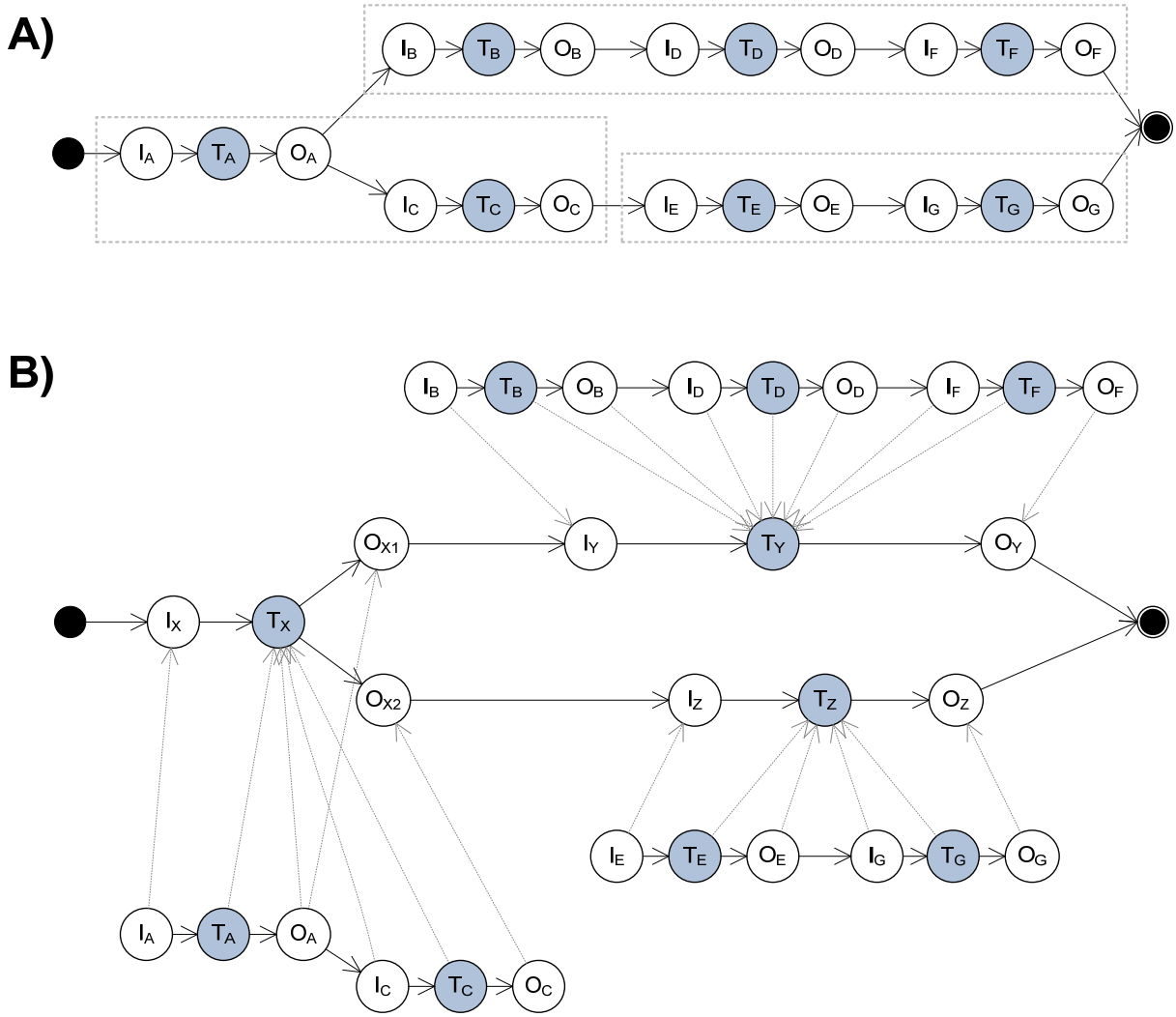
Definicja 6.16

Wierzchołek zależny, to wierzchołek pierwotny, zastąpiony w procesie komponentyzacji przez wierzchołek reprezentujący komponent. Przykładowo: na Rysunku 48 wierzchołek pierwotny T_A jest wierzchołkiem zależnym T_X, który reprezentuje komponent.

■

Rysunek 48 obrazuje przykładowy model DFA-Static (A) oraz jego podział na komponenty X, Y oraz Z, przedstawiony za pomocą grafu hierarchicznego (B), którego definicję można znaleźć w pracach [23] [24]. W przypadku alokacji zadań na najwyższym poziomie ziarnistości, pod uwagę brane są wierzchołki T, I oraz D z indeksami X, Y, Z. Alokacja któregoś z tych wierzchołków na określony procesor, pociąga za sobą alokację *wierzchołków zależnych* na ten sam procesor.

Dla przykładu: alokacja T_Y na węzeł N, powoduje w konsekwencji alokację T_B, O_B, ..., T_F na ten sam węzeł.



Rysunek 48. Zmiana ziarnistości modelu DFA: A) graf pierwotny, B) graf hierarchiczny przedstawiający podział na komponenty X, Y oraz Z.

Podsumowując, użycie komponentów, a tym samym zmniejszenie ziarnistości zadanego problemu, ma za zadanie – w przypadku optymalizacji alokacji zadań – wprowadzenie heurystyki zmniejszającej rozmiar problemu wejściowego, a także wskazanie miejsc podziału dla algorytmów typu *dziel i zwyciężaj*. Struktura komponentowa może być hierarchiczna: komponenty mogą zawierać inne komponenty, logarytmicznie redukując rozmiar problemu pierwotnego. Własności te nabierają niebagatelnego znaczenia w przypadku przetwarzania grafów reprezentujących modele DFA, o znacznej ilości wierzchołków, dla których istnieje konieczność zastosowania efektywnej alokacji na węzły środowiska obliczeniowego.

W niniejszym podrozdziale została jedynie zarysowana możliwość wykorzystania komponentów w procesie przetwarzania grafów Dynamicznych Analiz Finansowych. Użyteczność ich zastosowania została potwierdzona eksperymentalnie i planowane są dalsze prace mające na celu ich ścisłą charakterystykę w przypadku modeli DFA. Rozległość omawianego problemu wykracza niestety poza ramy niniejszej rozprawy i jest planowana jako cel przyszłych badań.

6.4. Analiza złożoności czasowej Kontrolera

W Rozdziale 5.6 przedstawiony został schemat działania *Kontrolera* (Algorytm 5.2, strona 56), którego przydatność omówiona zostanie w niniejszym rozdziale, gdyż dopiero na obecnym etapie ściśle zdefiniowany został zbiór produkcji oraz algorytmy optymalizacyjne, będące jego częścią.

Dla problemu ogólnego, ze względu na nierozwiązywalność problemu stopu, niemożliwe byłoby wyznaczenie złożoności pętli REPEAT–UNTIL – kroki 1) – 7). Jednakże analizowany jest graf G_{SUB}, który nie zawiera pętli oraz jest rozmiaru ograniczonego pewną stałą g_{MAX}. Z tego też względu wspomniana pętla może zostać wykonana co najwyżej g_{MAX} razy.

W kroku 2) odczytywany jest stan wszystkich procesów oraz węzłów obliczeniowych, których liczba jest co najwyżej n = #V(G), gdzie G jest grafem DFA-Snapshot reprezentującym środowisko wykonania; z tego względu złożoność obliczeniowa szacowana jest O(n). Podobne zachowanie cechuje kroki 6) oraz 8), których złożoność jest taka sama.

W kroku 3) uaktualnienia za pomocą *k* transformacji grafowych wymagać może *n* = #V(G) wierzchołków określonych w punkcie 2). Zbiór transformacji grafowych zawiera skończoną liczbę elementów, wielkość każdego z grafów będących lewą / prawą stroną produkcji jest ustalona i przyjmijmy, że nie większa niż *g*. Uruchomienie transformacji związane jest z kosztem co najwyżej O(n), gdyż usunięcie wierzchołka może powodować usunięcie maksymalnie *n* krawędzi. Parametry *k* oraz *g* są wartościami stałymi, z tego też względu złożoność kroku 3) jest $O(n^2)$. Zaobserwowano, że grafy reprezentujące modele DFA są grafami rzadkimi, dla których ilość krawędzi można oszacować liniowo w stosunku do liczby wierzchołków #E(G) = *ve* * #V(G), natomiast współczynnik *ve* można w przybliżeniu określić rozkładem normalnym o parametrach N(1.5, 0.25). Z tego też względu uruchomienie kroku 3) jest w praktyce efektywne.

W niniejszym podrozdziale scharakteryzowane zostało działanie algorytmów optymalizacyjnych użytych w kroku 4). Złożoność algorytmu przenoszenia gałęzi obliczeń na nieobciążone procesory oszacowano na O(#V(G) + #E(G)). Metoda wektoryzacji (która może zostać uruchomiona off-line na modelu statycznym przed uruchomieniem obliczeń właściwych) została oszacowana przez O(p * q), gdzie p = $\#V_{TEST}$ << #V(G), natomiast q = #V(G) + #E(G). Natomiast koszt podziału zbioru wierzchołków na komponenty został ograniczony przez O(#V(G) + #E(G)). Wszystkie powyższe metody zostały uznane za efektywne w praktyce.

Poniżej przeanalizowany zostanie Algorytm 5.1 (strona 54), uruchamiany w kroku 3) przez Algorytm 5.2.

W kroku 1) uruchomionych może zostać co najwyżej n = #V(G) produkcji: startując z dowolnego miejsca osiągnięty zostanie wierzchołek T i niespełniony zostanie warunek TC (Time Condition) (5.11). Każdy wierzchołek może zostać odwiedzony co najwyżej jeden raz i każda z produkcji (Rozdział 5.7), których jest skończona ilość może zostać uruchomiona dla danego wierzchołka co najwyżej jeden raz. Mapowanie produkcji sprawdzane jest na podstawie dwóch wierzchołków charakterystycznych (Tabela 7, strona 79) i ma złożoność O($\binom{n}{2}$). Złożoność czasowa całego kroku 1) jest więc O(n^3). W praktyce jednak ilość uruchamianych produkcji jest dużo mniejsza od #V(G). Podobnie mapowanie produkcji odbywa się na zbiorze $V_{SUB} \subset V(G)$, dla którego zachodzi $\#V_{SUB} << \#V(G)$, ze względu na możliwość wstępnej filtracji wierzchołków charakterystycznych w zależności od etykiety oraz dodatkowo możliwe jest utrzymywanie jedynie podzbioru $V_{FRONT} \subset V_{SUB}$, zawierającego wierzchołki określające „czoło obliczeń" (aktywne procesy oraz dane). Z tego względu krok 1) jest w praktyce efektywny.

Kroki 2) oraz 3) są złożoności czasowej O(n), gdzie n = #V(G), natomiast krok 4) jest złożoności O(1). Pętla FOR w kroku 5) jest złożoności O(n), gdyż krok 6) jest O(1).

Podsumowując, Algorytm 5.1 jest złożoności O(n^3), jednak ze względu na uwarunkowania zadania jest on w praktyce efektywny.

Wracając do analizy *Kontrolera* (Algorytm 5.2), wykazano że pętla REPEAT-UNTIL wykonywana jest co najwyżej g_{MAX} razy, natomiast kroki 2) – 6) oraz 8) są złożoności co najwyżej O(n^3), gdzie n = #V(G), oraz że są rozwiązaniami efektywnymi. Algorytm 5.2 jest więc także w praktyce rozwiązaniem efektywnym.

6.5. Podsumowanie

Ze względu na rozmiar zadań Dynamicznych Analiz Finansowych, użycie jednego węzła obliczeniowego stało się niewystarczające. Wymagania pamięciowe oraz obliczeniowe przerastają moce obliczeniowe dzisiejszej stacji roboczej. Pomimo przewidywanego, dynamicznego wzrostu wydajności sprzętu, zawsze będzie istniała presja środowisk biznesowych do zwiększenia precyzji metod symulacyjnych poprzez podniesienie ilości rozpatrywanych scenariuszy. Rozwiązaniem problemu jest zrównoleglenie obliczeń, a w szczególności obliczenia rozproszone.

W niniejszym rozdziale zaproponowano trzy metody opracowane przez autora na podstawie badań przeprowadzonych nad procesami Dynamicznych Analiz Finansowych, pozwalające na skrócenie czasu ewaluacji.

W Rozdziale 6.1 przedstawiona została metoda pozwalająca na zrównoleglenie obliczeń, polegająca na przenoszeniu wybranych gałęzi zadań niezależnych na niezajęte procesory. W celu uzyskania wydajności algorytmu pozwalającej na optymalizację on-line, opracowany algorytm wykorzystuje charakterystyczne cechy modeli DFA w celu zawężenia rozmiaru podgrafu poddawanego analizie, a jego złożoność została oszacowana na poziomie O(#V(G) + #E(G)), gdzie #V(G) i #E(G) oznaczają odpowiednio ilość wierzchołków i krawędzi rozpatrywanego grafu.

W Rozdziale 6.2 zaproponowano metodę wektoryzacji, skutkującą znaczną redukcją liczby przetwarzanych wierzchołków. Metoda ta polega zastąpieniu ciągów takich samych operacji reprezentujących poszczególne scenariusze jednym ciągiem operacji ekwiwalentnych. Przeprowadzone obserwacje pozwoliły na stwierdzenie faktu, że sekwencyjne wykonanie takiej samej operacji na wektorze danych będącym sumą danych poszczególnych operacji jest znacznie szybsze. Co więcej, znaczne obniżenie liczby wierzchołków ma niebagatelny wpływ na zwiększenie wydajności procesu dystrybucji zadań. Zaproponowany algorytm wektoryzacji może zostać użyty jeszcze przed etapem obliczeń właściwych dla modelu DFA-Static lub on-line, a rząd złożoności $O(p * q)$, gdzie $p = \#V_{TEST} << \#V(G)$, natomiast $q = \#V(G) + \#E(G)$ (przy założeniu $\#E(G)$ liniowo zależnego od $\#V(G)$), gwarantuje jego praktyczną użyteczność.

Rozdział 6.3 zawiera propozycję dynamicznego zmniejszania analizowanej dla celów optymalizacji liczby wierzchołków grafu poddawanego analizie, w celu podjęciu decyzji dotyczącej rozproszenia obliczeń, uzyskanej dzięki zmianie *ziarnistości* poprzez zastosowanie *komponentyzacji*. Proces ten może być przeprowadzany wielokrotnie, tworząc kolejne poziomy *ziarnistości*. Zastąpienie grupy wierzchołków jednym elementem (powodujące enkapsulację), skutkuje zmniejszeniem ilości danych, będących wejściem algorytmu optymalizacji. Możliwość dynamicznej zmiany poziomu ziarnistości, uzależnionego od postępu procesu obliczeniowego, pozwala na jego optymalne dostosowanie, maksymalizujące wydajność algorytmu alokacji zadań.

W przypadku każdej z metod wykazano możliwy do uzyskania zysk w postaci skrócenia czasu obliczeń. Co więcej, możliwa jest dowolna kombinacja użycia przedstawionych rozwiązań, gdyż adresują one różne aspekty optymalizacji i nie powodują negatywnych w skutkach wzajemnych interferencji, a wręcz wzajemnie się uzupełniają.

Jako przedmiot przyszłych badań określono dalsze prace nad poprawą wydajności algorytmów alokacji zadań dla środowiska rozproszonego.

Ze względu na ograniczoną objętość niniejszej rozprawy, osobnej dyskusji wymaga rozszerzenie skuteczności działania algorytmu wyszukiwania złożonych łańcuchów scenariuszy, w celu wektoryzacji zadań. W procesie prac badawczych wyselekcjonowano często pojawiające się przypadki występowania pojedynczych ciągów zadań „spoza łańcucha” (Rysunek 48, wierzchołek X), a także łańcuchów z odgałęzieniami (Rysunek 48, wierzchołki E_i).

Rozszerzonych badań wymaga także zagadnienie komponentyzacji, a w szczególności opracowanie algorytmów automatycznego podziału modeli DFA, maksymalnie polepszających jakość alokacji zadań, uzyskiwaną przez algorytmy optymalizacji.

7. Metodyka badań eksperymentalnych oraz zastosowanie modelu grafowego w praktyce

Szereg zagadnień teoretycznych pojawiających się w niniejszej pracy i prowadzących do minimalizacji czasu wykonania procesów obliczeniowych, jest silnie związany z tematyką Dynamicznych Analiz Finansowych. Z tego względu celowe było przeprowadzenie doświadczeń dla rzeczywistych modeli DFA, potwierdzających praktyczne zastosowanie zaproponowanych rozwiązań.

Założony plan badań eksperymentalnych obejmował:

1) test efektywności reprezentacji złożonych modeli DFA o dużej liczbie symulacji (rzędu 10^5 – 10^6) za pomocą formalizmu grafów etykietowanych oraz potencjalne wykorzystanie komponentów,
2) badania nad funkcją Memory Requirements, w celu określenia maksymalnego, dopuszczalnego poziomu obciążenia pamięciowego węzłów obliczeniowych niepowodującego spadku wydajności ewaluacji oraz określenia funkcji spadku wydajności,
3) badanie poprawy wydajności systemu przy użyciu techniki scalania scenariuszy i zastosowaniu obliczeń wektorowych,
4) stworzenie eksperymentalnego środowiska rozproszonego, służącego do badania poprawy wydajności poprzez przenoszenie gałęzi obliczeń na inne węzły obliczeniowe.

W ramach niniejszej pracy zrealizowano eksperymenty 1) – 3), natomiast dokończenie realizacji punktu 4) jest przewidziane jako temat przyszłych badań.

7.1. Zwiększenia efektywności gospodarki pamięcią za pomocą komponentów

Grafy atrybutowane wybrane zostały jako baza modelu DFA – zarówno do przedstawienia klasy zadania (DFA-Static), jak i środowiska wykonania wraz z zaalokowanymi procesami w pewnym momencie w czasie (DFA-Snapshot).

Pierwszym krokiem badań doświadczalnych jakie zostały podjęte, było sprawdzenie efektywności pamięciowej reprezentacji dużych modeli (100K – 1M scenariuszy) za pomocą grafów.

Kod 7.1 przedstawia propozycję interfejsu definiującego wierzchołek. W celu maksymalizacji efektywności algorytmów grafowych, w wymaganiach dotyczących jego implementacji, założono dostęp do rodzica, dzieci oraz sąsiadów w czasie O(1).

Minimalne wymagania pamięciowe obiektu klasy implementującej interfejs IVertex (spełniającego wyżej wymienione założenia) liczone we wskaźnikach (ang. pointers), przedstawiają się następująco:

- this: 1
- NodeData: (minimalnie wskaźnik + struktura z Label i DataSize) 1 + 2
- Parent: 1
- Subnodes: (załóżmy strukturę płaską – null) 1

- Predecessors: (załóżmy ilość poprzedników 2) 1 + 2
- Successors: (załóżmy ilość następników 2) 1 + 2

Wielkość obiektu została oszacowana na co najmniej 12 wskaźników, co dla systemu 64-bitowego daje 96 bajtów (przyjmijmy dla uproszczenia zaokrąglenie 0.1 KB).

Kod 7.1. Interfejs wierzchołka grafu (C#).

```
public interface IVertex
{
    object Data { get; set; }  // Label, DataSize, ...

    IVertex Parent { get; set; }
    IVertex[] SubNodes { get; set; }
    IVertex[] Predecessors { get; set; }
    IVertex[] Successors { get; set; }
}
```

Wymagania pamięciowe potrzebne do reprezentacji modelu składającego się z 10^3 wierzchołków i 10^5 scenariuszy, czyli 10^8 wierzchołków zostały oszacowane na: 10^8 * 0.1 KB ≅ 9.5 GB. Dla modelu ze scenariuszami w ilości 10^6 będzie to 95 GB. Jest to wielkość w praktyce nie do przyjęcia.

Z tego względu użyty został komponent (Rozdział 3.3, 6.3) do jednokrotnego przedstawiania zawartości scenariusza, a następnie wielokrotna jego referencja (Rysunek 49). Zabieg ten jest kolejnym, praktycznym rozszerzeniem definicji modelu DFA-Static, gdyż w Rozdziale 3.3 założono, iż podstruktura może posiadać wyłącznie jeden nadrzędny wierzchołek T.

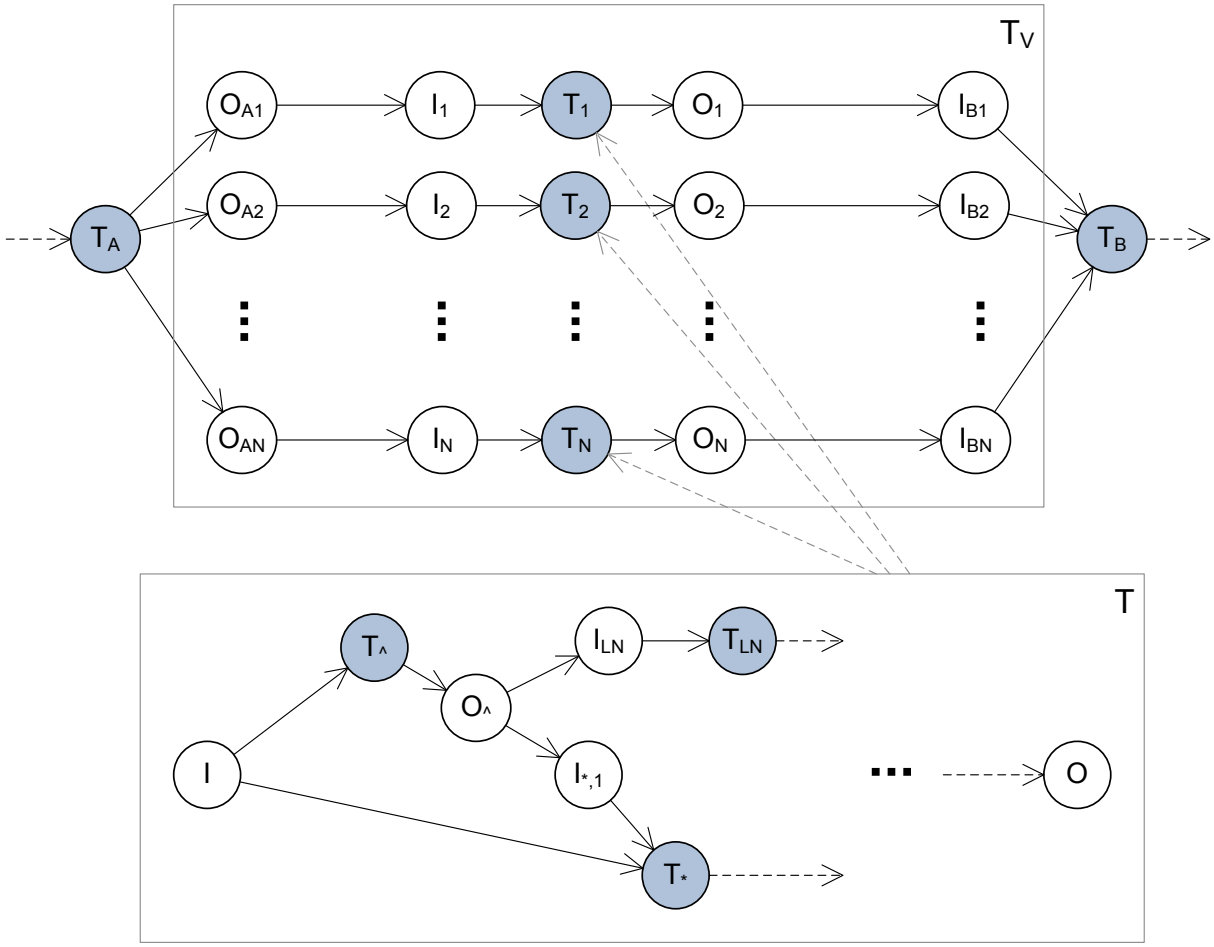
Użycie pamięci dla modelu o 10^5 scenariuszach zmalało do wielkości: [10^3 (wierzchołki modelu) + 5 * 10^5 (referencje do scenariuszy)] * 0.1 KB ≅ 50 MB, natomiast dla modelu 10^6 scenariuszy ≅ 500 MB, przy założeniu, że istnieje tylko jedno odwołanie do całej struktury scenariusza z każdego wierzchołka $T_1 – T_N$ (Rysunek 49).

Jest to rozwiązanie akceptowalne, aczkolwiek istnieje dalsza możliwość poprawy wydajności poprzez stworzenie wierzchołka T_V (Rysunek 49), posiadającego „wirtualną” strukturę zawierającą $T_1 – T_N$, wraz z wierzchołkami reprezentującymi dane wejściowe oraz wyjściowe. Przy takim rozwiązaniu wielkość potrzebnej pamięci dla dowolnej liczby scenariuszy można oszacować na: 10^3 (wierzchołki modelu) * 0.1 KB ≅ 100 KB.

Przeprowadzone doświadczenia nad reprezentacją modelu za pomocą struktury grafowej pozwoliły zidentyfikować problemy związane z dużą ilością przetwarzanych scenariuszy, wymagające znacznej ilości pamięci, uniemożliwiającej reprezentację modelu w praktyce. Zaproponowano rozwiązanie wykorzystujące ideę komponentów (Rozdział 3.3, 6.3) w nieznacznie zmodyfikowanej formie, skutkujące wysoce zadawalającym obciążeniem pamięciowym.

Podsumowując, wykonane eksperymenty wykazały przydatność zastosowania modelu grafowego w praktyce, jednocześnie nakreślając plan przyszłych badań teoretycznych, mających

na celu modyfikację modeli DFA-Static oraz DFA-Snapshot, pozwalającą na wielokrotną referencję komponentu przez wierzchołki T oraz definiującą „wirtualne" wierzchołki scenariuszy (T_V).

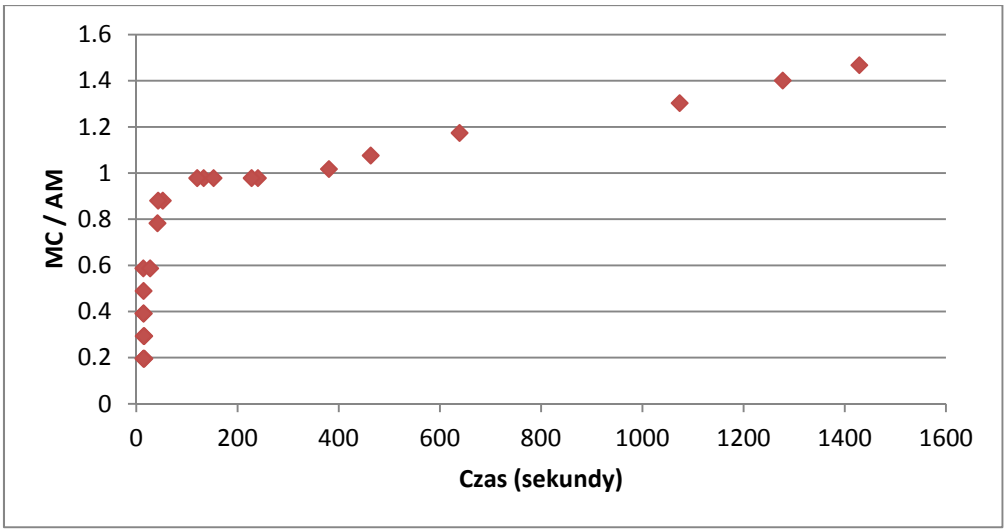


Rysunek 49. Wielokrotna referencja scenariusza T przez wierzchołki T_1 – T_N oraz obszar wierzchołka wirtualnego T_V.

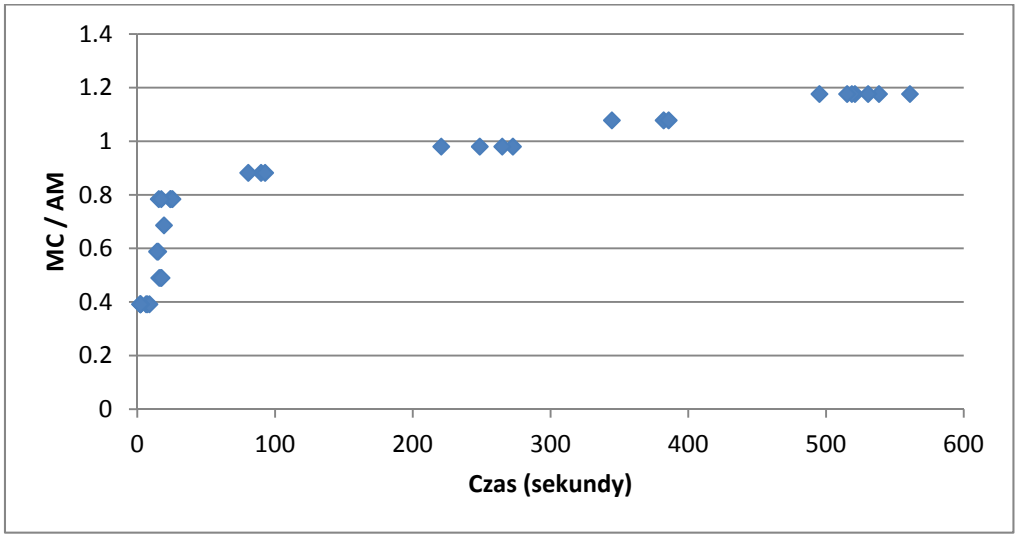
7.2. Wyznaczanie funkcji Memory Efficiency

Jednym z kluczowych elementów, jaki został zaobserwowany, powodującym znaczne opóźnienie wykonywania zadań DFA, jest wykorzystanie pamięci pomocniczej (dyskowej) przez proces obliczeniowy. Z tego powodu w Rozdziale 3 dla wierzchołków T wprowadzony został parametr ME (Memory Efficiency), pozwalający określić stopień spowolnienia obliczeń związany z przekroczeniem limitu dostępnej pamięci operacyjnej. Znajomość tego parametru według przewidywań może mieć znaczny wpływ na wybór wierzchołka obliczeniowego, na który przeniesione mają być zadania, gdyż węzły z szybszymi procesorami, lecz o mniejszej ilości pamięci mogą być docelowo znacznie mniej wydajne.

W niniejszym rozdziale przedstawione zostaną wyniki doświadczeń prowadzących do wyznaczenia w sposób empiryczny parametru ME, używanego do szacowania czasu obliczeń (wzór (5.14), strona 59).



Rysunek 50. Czas wykonania testowego procesu DFA w zależności od wartości współczynnika MC/AM dla maszyny z pamięcią operacyjną 4GB.



Rysunek 51. Czas wykonania testowego procesu DFA w zależności od wartości współczynnika MC/AM dla maszyny z pamięcią operacyjną 2GB.

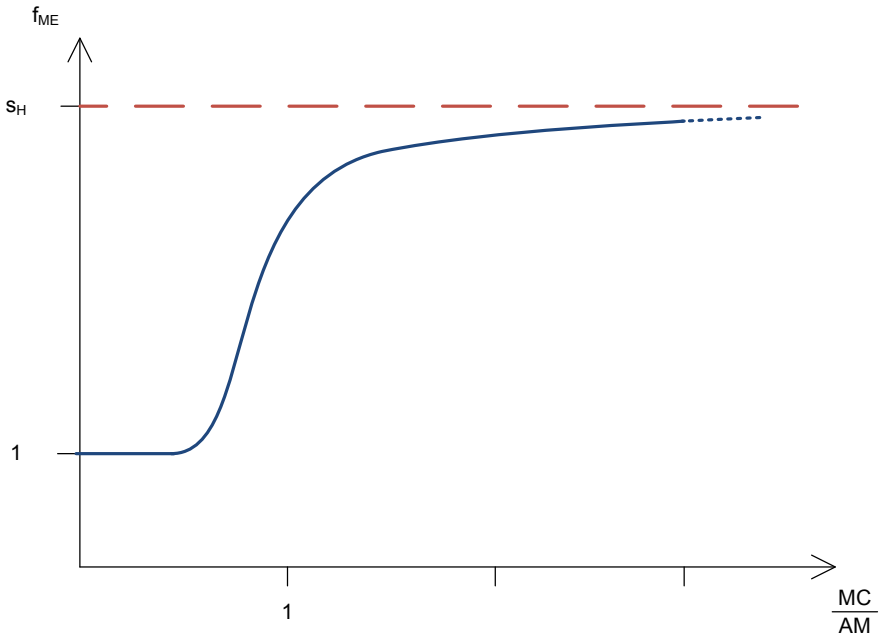
Metodyka badań została ustalona w następujący sposób: skonstruowano model wykonujący ciąg takich samych operacji OP_X, który następnie uruchomiono zwiększając liczbę scenariuszy wykonywanych wektorowo i powodujących wzrost wymagań pamięciowych, notując czas potrzebny na ewaluację.

Na Rysunku 50 oraz 51 przedstawiono wyniki doświadczeń w postaci wykresów określających czas wykonania operacji dodawania dwóch liczb (+) w zależności od wartości MC/AM (Memory Consumption / Available Memory), gdzie AM było ilością pamięci RAM testowanej maszyny (wykorzystując maszynę wirtualną możliwe było ograniczenie dostępnej pamięci środowiska *guest* przy niezmiennej architekturze procesora oraz systemu operacyjnego).

Wyniki doświadczeń pozwoliły na stwierdzenie w przybliżeniu liniowego (o współczynniku c_H) przyrostu czasu wykonania w zależności od wielkości użytej pamięci poniżej pewnej wartości MC/AM, a następnie także liniowego przyrostu o współczynniku c_L. Stwierdzono istnienie warunku $c_H > c_L$ oraz określono wartości c_H i c_L. Stwierdzono różnice współczynników c_L dla systemów o różnej wielkości pamięci RAM z trendem $c_L(ram_1) > c_L(ram_2)$, gdy $ram_1 < ram_2$.

Doświadczenie zostało powtórzone dla innych operacji: ^, *, ln, sin, dla których stwierdzono podobne zachowanie.

Na podstawie przeprowadzonych eksperymentów wyznaczono rodzinę funkcji f_{ME} dla systemów o różnej wielkości pamięci operacyjnej RAM. Wyznaczone w sposób empiryczny funkcje mogą zostać wykorzystane w dalszej kolejności w rozważaniach teoretycznych, dotyczących szybkości obliczeń (wzór (5.14), strona 59), co było celem przeprowadzonych eksperymentów. Przykładowy wykres funkcji f_{ME} przedstawia Rysunek 52.



Rysunek 52. Przykładowy wykres funkcji Memory Efficiency $f_{ME}(x)$ w zależności od stopnia wykorzystania pamięci operacyjnej MC/AM.

7.3. Badanie poprawy wydajności poprzez scalanie scenariuszy

W Rozdziale 6.2 opisany został mechanizm scalania scenariuszy, polegający na zastąpieniu ciągów takich samych operacji jednym ciągiem, wykonującym te same operacje w sposób sekwencyjny na wektorze danych.

Przykład scalania scenariuszy omówiony będzie przy użyciu uproszczonego fragmentu modelu, przedstawionego na Rysunku 53 oraz algorytmu DFS (ang. Depth-First Search, przeszukiwanie w głąb) uruchomionego na jednym procesorze. Dla modelu A przyjęto następującą kolejność odwiedzonych wierzchołków:

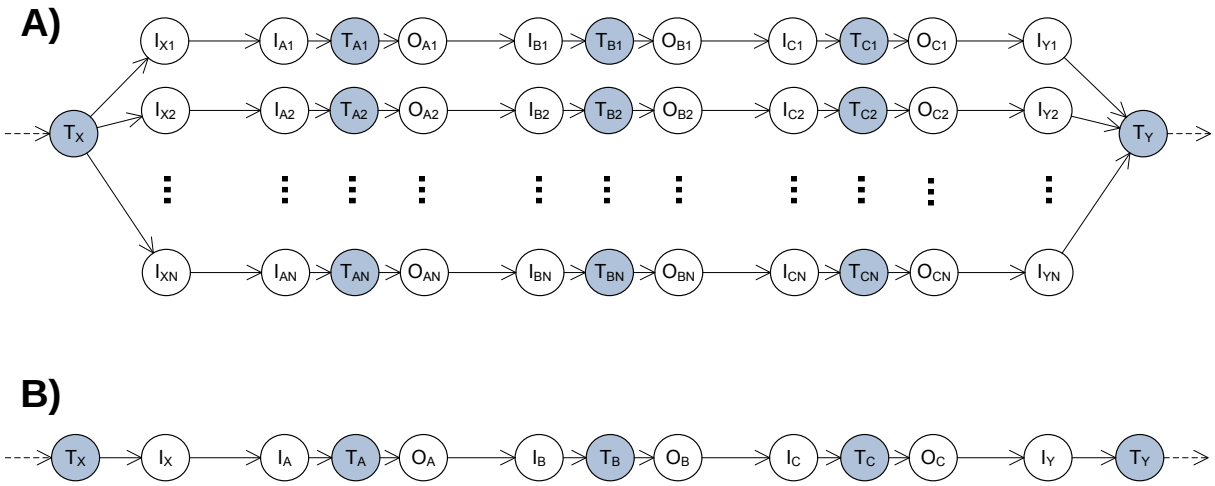
$$T_X, I_{X1}, I_{A1}, T_{A1}, ..., I_{Y1}, I_{X2}, I_{A2}, ...I_{YN}, T_Y \quad (7.1)$$

natomiast dla modelu B:

$$T_X, I_X, I_A, T_A, ..., I_Y, T_Y \quad (7.2)$$

Założono także, że dane I_{X1}, ..., I_{XN} są tego samego rozmiaru (odpowiednio dalej dane I oraz O w tych samych kolumnach) oraz operacje T_{A1}, ..., T_{AN} są takie same (odpowiednio operacje T_{Bk} i T_{Ck}).

Powyższe założenia umożliwiły scalenie poszczególnych danych w wektor danych, na przykład I_{X1}, ..., I_{XN} w wektor I_X oraz poszczególnych zadań w „zadanie wektorowe", czyli sekwencyjne uruchomienie tej samej operacji na kolejnych elementach wektora.

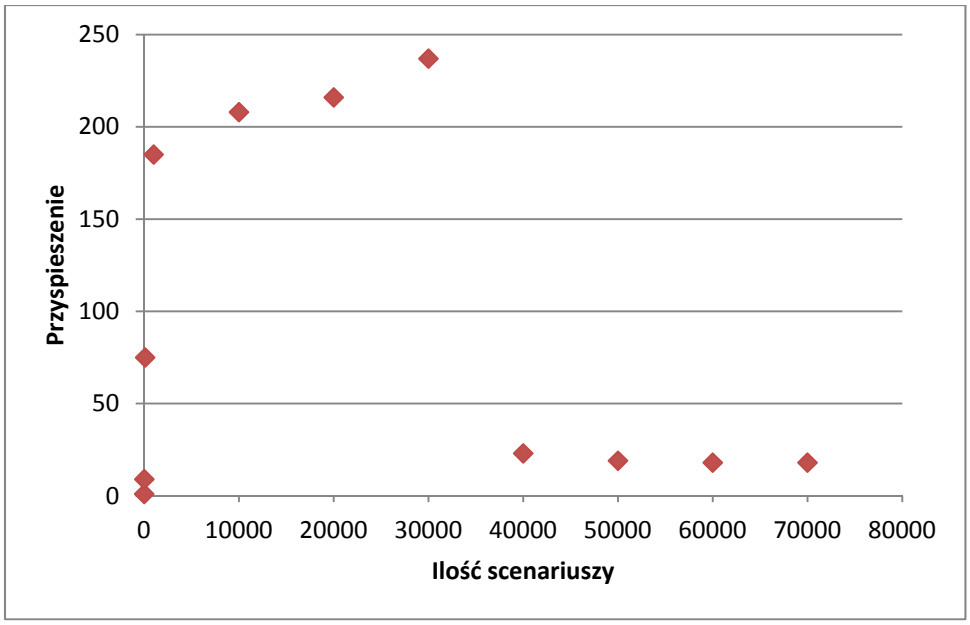


Rysunek 53. A) Przykładowy model zawierający ciągi scenariuszy oraz B) model równoważny wykonujący operacje wektorowe.

Na potrzeby eksperymentu stworzono środowisko obliczeniowe mające postać interpretera zadanego modelu analizującego poszczególne formuły oraz wykonującego kolejno operacje. Stworzono także szereg modeli nawiązujących do najczęściej pojawiających się problemów biznesowych. Doświadczenia przeprowadzono stopniowo zwiększając liczbę scenariuszy N, porównując czas wykonania modelu oryginalnego (Rysunek 53A) oraz jego wersji zmodyfikowanej poprzez scalenie scenariuszy (Rysunek 53B).

Wyniki przeprowadzonych doświadczeń wykazywały podobny schemat, który zostanie przedstawiony na wybranym projekcie. Rysunek 54 przedstawia wykres przyspieszenia działania modelu realizującego operacje wektorowe w stosunku do modelu oryginalnego, w zależności od ilości przetwarzanych scenariuszy.

Badania wykazały, że przy pewnej liczbie scenariuszy możliwe jest uzyskanie około 250-krotnego przyspieszenia obliczeń. Jednak powyżej pewnej wartości granicznej ilości scenariuszy, z powodu konieczności użycia dużo wolniejszej dyskowej pamięci pomocniczej, wydajność systemu drastycznie spada, pozostając na poziomie od 15 do 25 razy, w zależności od modelu oraz konfiguracji sprzętowej.



Rysunek 54. Przyspieszenie działania modelu realizującego operacje wektorowe w stosunku do modelu oryginalnego, w zależności od ilości scenariuszy.

Podsumowując, przeprowadzone badania empiryczne wykazały wysoką skuteczność użycia zaproponowanej w Rozdziale 6.2 metody scalania scenariuszy, pozwalające na podniesienie wydajności nawet o dwa rzędy wielkości. Jednakże rozwiązania biznesowe są zazwyczaj projektami przetwarzającymi znaczne ilości danych, dla których zachodzi konieczność użycia pamięci pomocniczej, co znacznie obniża wydajność obliczeń. Z tego względu dla projektów komercyjnych nie należy spodziewać się zwiększenia szybkości powyżej jednego rzędu wielkości.

Za cel przyszłych badań przyjęto problem wyznaczenia optymalnego podziału zadań o dużej liczbie scenariuszy na wektory w taki sposób, aby zminimalizować użycie pamięci pomocniczej, przy jednoczesnym zachowaniu jak największej liczby elementów wektora. Oszacowanie tej wartości nie jest zadaniem trywialnym i wiąże się między innymi ze znajomością wysokości drzewa obliczeń dla zadanego problemu.

Zaproponowany algorytm DFS wykonywania zadań nie jest optymalny ze względu na użycie pamięci potrzebnej do przeprowadzenia obliczeń. Celem przyszłych doświadczeń jest analiza modelu i wyznaczenie kolejności zadań w sposób minimalizujący wysokość drzewa obliczeń, pociągający za sobą zmniejszenie ilości danych składowanych na stosie. Rozważania na pokrewne tematy zostały przeprowadzone między innymi w pracach [79] oraz [80].

7.4. Podsumowanie

W niniejszym rozdziale przedstawione zostały wyniki badań eksperymentalnych. Potwierdzona została skuteczność użycia grafów jako reprezentacji problemów DFA o znacznych rozmiarach. Przedstawienie modeli o liczbie scenariuszy sięgającej 10^6 osiągnięte zostało z pomocą użycia idei komponentu, do którego możliwa jest wielokrotna referencja.

Wyznaczona została empirycznie funkcja Memory Efficiency, będąca parametrem wierzchołków T, pozwalająca na oszacowanie spowolnienia obliczeń związanego z koniecznością użycia pomocniczej pamięci dyskowej.

Wykazana została znaczna poprawa wydajności obliczeń poprzez zastosowanie metody scalania scenariuszy opisanej teoretycznie w Rozdziale 6.2.

Jako główny cel przyszłych badań eksperymentalnych wyznaczono stworzenie rozproszonego środowiska obliczeniowego, pozwalającego na badanie metod obliczeń równoległych dedykowanych dla procesów Dynamicznych Analiz Finansowych. Zostały dotychczas poczynione przez autora badania zarówno teoretyczne [81] [64] [82] [83] [70], jak i praktyczne [81], adresujące wybrane aspekty problemu, dążące do realizacji założonego celu.

8. Podsumowanie

8.1. Wnioski końcowe

Celem niniejszej pracy było skonstruowanie formalizmu opisującego zadania Dynamicznych Analiz Finansowych w sposób umożliwiający automatyczną optymalizację alokacji procesów w środowisku rozproszonym. Przegląd literatury wykazał, że temat ten nie był dotychczas rozpatrywany. Z tego względu konieczne było przeprowadzenie badań zarówno natury teoretycznej, jak i praktycznej.

Poniżej podsumowany zostanie przebieg prac badawczych, dzięki którym zrealizowany został założony cel. Pozwoli to także na potwierdzenie postawionej w niniejszej rozprawie tezy.

W Rozdziale 2 przedstawiono genezę oraz zarys metod symulacyjnych. Opisano skomplikowanie problemu, związane między innymi z poluzowaniem regulacji oraz coraz powszechniejszym użyciem instrumentów pochodnych. Prace badawcze przeprowadzone przez autora pozwoliły na zidentyfikowanie źródeł problemów implementacji systemów Dynamicznych Analiz Finansowych. Głównym z nich był brak jednolitego, ścisłego formalizmu, pozwalającego na zapisanie zadania DFA bez użycia opisu w języku naturalnym, a tym samym automatyzacja procesu alokacji zadań bez udziału operatora. Posiłkując się diagramami języka UML, autor przedstawił najczęściej występujące schematy zachowań DFA, zaobserwowane na etapie prac badawczych, mające wpływ na wydajność obliczeń. Umożliwiło to postawienie wymagań dotyczących modelu Dynamicznych Analiz Finansowych.

Wnikliwa analiza DFA umożliwiła podjęcie decyzji o przyjęciu formalizmu grafów atrybutowanych, jako bazy opisu zadnia Dynamicznych Analiz Finansowych, będącego klasą problemu. W Rozdziale 3 wprowadzony został model statyczny, przyjęto założenia i zdefiniowano parametry charakteryzujące poszczególne elementy. Spójny oraz formalny aparat matematyczny stworzył możliwość automatyzacji analizy zadań.

W Rozdziale 4 w ścisły sposób, za pomocą wprowadzonego modelu DFA-Static, scharakteryzowano problemy zasygnalizowane w Rozdziale 2. Aparat matematyczny posłużył do opisu zagadnień, które wymagały wcześniej objaśnienia w języku naturalnym, wspomaganym diagramami UML. Zaprezentowane przykłady pozwoliły na omówienie zastosowania wprowadzonych parametrów modelu. Użycie formalizmu grafowego pozwoliło na łatwą oraz klarowną analizę problemów, wręcz automatycznie nasuwającą rozwiązania. Omówienie zagadnień na bazie DFA-Static, pozwoliło także na analizę wymagań dotyczących modelu środowiska wykonania.

Rozdział 5 zawiera definicję dynamicznego modelu DFA, pozwalającego na kontrolę alokacji zadań. W pierwszej kolejności zdefiniowano wymagania, jakie powinien spełniać takowy model, oparte na badaniach przeprowadzonych przez autora. Zarysowany został także szkic działania systemu. Wprowadzony wcześniej model statyczny rozszerzono o elementy środowiska wykonania, reprezentując stan systemu w pewnym momencie trwania obliczeń. Dynamika zmian – zarówno środowiska, jak i procesów – opisana została za pomocą transformacji grafowych, do których autor wprowadził czynnik czasowy, charakteryzujący opóźnienia związane z przetwarzaniem kolejnych

zadań. Przedstawiono algorytm sterowania procesem obliczeniowym, wprowadzający automatyzację obliczeń zadań DFA, a tym samym eliminację udziału operatora.

W pracy zaprezentowana została lista transformacji grafowych, rozszerzonych o dodatkowe parametry charakterystyczne dla modelu DFA. Na bazie przedstawionego zbioru produkcji dokonana została wnikliwa analiza przebiegu wykonania przykładowego modelu DFA.

W Rozdziale 6 zaprezentowano autorskie algorytmy poprawy czasu wykonania, dedykowane dla procesów DFA i oparte na przeprowadzonych przez autora badaniach nad prezentowanym zagadnieniem. Opisany został algorytm optymalizujący dystrybucję zadań na węzły środowiska obliczeniowego w celu minimalizacji czasu wykonania, o złożoności O(#V(G) + #E(G)), gdzie #V(G) i #E(G) oznaczają odpowiednio ilość wierzchołków i krawędzi rozpatrywanego grafu. Oszacowana, niska złożoność obliczeniowa algorytmu kwalifikuje przedstawione rozwiązanie jako efektywne w praktyce, szczególnie, że wartość #E(G) w modelach DFA wykazuje liniową zależność od #V(G).

Zaproponowano metodę wektoryzacji operacji, skutkującą znaczną redukcją liczby przetwarzanych wierzchołków, polegającą na scalaniu ciągów identycznych operacji każdego ze scenariuszy w jeden ciąg operacji równoważnych. Przeprowadzone obserwacje pozwoliły na stwierdzenie faktu, że sekwencyjne wykonanie takiej samej operacji na wektorze danych będącym sumą danych poszczególnych operacji jest znacznie szybsze niż osobne obliczenia wykonywane dla każdego ze scenariuszy z osobna. Dodatkowo, uzyskana redukcja liczby wierzchołków, sięgająca rzędu 10^6, związana z liczbą symulacji, pozwoliła znacznie zwiększyć wydajność procesu optymalizującego dystrybucję zadań. Algorytm wektoryzacji może zostać użyty dla modelu statycznego przed fazą obliczeń właściwych lub on-line, a jego stosunkowo niska złożoność O(p * q), gdzie $p = \#V_{TEST} << \#V(G)$, natomiast q = #V(G) + #E(G), umożliwia jego praktyczne zastosowanie.

Przedstawiono także propozycję dynamicznego zmniejszenia liczby wierzchołków grafu, polegającą na zmianie ziarnistości poprzez zastosowanie komponentyzacji. Zastąpienie zbioru wierzchołków jednym elementem w celu enkapsulacji podzadań, powoduje zmniejszenie ilości danych analizowanych przez algorytmy optymalizujące alokację zadań, a proces ten może zostać uruchomiony wielokrotnie, tworząc kolejne poziomy ziarnistości. Dynamiczna zmiana poziomu pozwala na jego optymalne dostosowanie, w zależności od zaawansowania ewaluacji, maksymalizując wydajność algorytmu alokacji zadań.

Dla każdej z wyżej wymienionych metod wykazano możliwy do uzyskania zysk w postaci skrócenia czasu obliczeń. Możliwe jest także połączenie wszystkich zaproponowanych rozwiązań, gdyż nie kolidują one wzajemnie, a wręcz się uzupełniają, adresując odmienne obszary działania.

W Rozdziale 1 przedstawiono rezultaty eksperymentów. Wykazano skuteczność formalizmu grafowego jako metody zapisu modeli o pokaźnych rozmiarach. Empirycznie wyznaczono parametry niezbędne do szacowania opóźnień obliczeń, związanych z użyciem pamięci pomocniczej. Potwierdzono także skuteczność wektoryzacji przetwarzania zadań, uzyskując znaczne przyspieszenie momentu zakończenia ewaluacji. Za główny cel przyszłych badań postawiono stworzenie rozproszonego środowiska obliczeniowego, służącego do analizy wydajności metod zrównoleglenia obliczeń.

Na podstawie opisanych powyżej wyników badań można stwierdzić, że cel badawczy rozprawy, postawiony na wstępie, został osiągnięty. Założona w niniejszej pracy teza została potwierdzona:

Procesy Dynamicznych Analiz Finansowych można formalnie zaspecyfikować za pomocą transformacji grafowych w sposób, który pozwoli na optymalizację alokacji zadań w środowisku rozproszonym.

8.2. Kierunki dalszych prac

W niniejszej rozprawie poruszony został szereg zagadnień, które mogą zostać poddane dalszej, szczegółowej analizie.

Celowe wydaje się opracowanie charakterystyk modelujących wzajemne interferencje zadań dla różnych środowisk sprzętowych. Modyfikacje funkcji *loadFactor* (Algorytm 5.1, strona 54) powinny umożliwić między innymi symulacje priorytetów procesów oraz modelowanie różnych topologii środowisk sieciowych.

Przedstawiona w Rozdziale 5 lista transformacji grafowych modelujących dynamikę zmian, powinna zostać uzupełniona o produkcje modyfikujące środowisko sprzętowe: dodawanie / usuwanie węzłów obliczeniowych oraz dodawanie / usuwanie połączeń pomiędzy węzłami.

Modelowanie częściowych awarii sprzętowych oraz analiza możliwości kontynuacji procesu obliczeniowego w takich przypadkach byłaby cennym rozwiązaniem, entuzjastycznie przyjętym przez środowisko biznesowe. W niniejszej rozprawie na wielu etapach rozważań założono przyjęcie takiej opcji, między innymi poprzez zapamiętanie historii obliczeń (5.8) oraz pozostawienie referencji do wykorzystanych danych w postaci wierzchołków etykietowanych „AD” (Archived Data).

Dalszych badań wymaga także analiza możliwości kontynuowania obliczeń w przypadku czasowego rozłączenia podsieci będących środowiskiem obliczeniowym, a tym samym zwiększenie wydajności obliczeń. Konieczna byłaby decentralizacja podejmowania decyzji oraz wprowadzenie metod agregacji wyników częściowych. Opcja ta została poddana wstępnej analizie: w pracach (Kotulski) [22] [61] przedstawiono rozwiązania umożliwiające równoległą modyfikację środowisk grafowych przy zachowaniu globalnej spójności.

Dalszej analizie mogą zostać poddane także algorytmy alokacji zadań w celu zwiększenia ich wydajności.

Ze względu na ograniczoną objętość niniejszej rozprawy, zaznaczony jedynie został problem wyszukiwania złożonych łańcuchów scenariuszy, w celu wydajniejszej wektoryzacji zadań (Rozdział 6.2). W procesie prac badawczych wyselekcjonowano często pojawiające się przypadki występowania pojedynczych ciągów zadań spoza łańcucha (Rysunek 48, wierzchołek X), a także łańcuchów z odgałęzieniami (Rysunek 48, wierzchołki E_i).

Potencjalna poprawa wydajności procesów optymalizacyjnych może zostać osiągnięta także poprzez polepszenie algorytmów komponentyzacji odpowiedzialnych za automatyczny podział wierzchołków modelu na pozdbiory, zastępowane pojedynczymi elementami w procesie zmiany ziarnistości.

Spis rysunków

Bibliografia

[1] C. Smithson, C. Smith i D. Sykes Wilford, Zarządzanie ryzykiem finansowym, Kraków: Dom Wydawniczy ABC, 2000.

[2] Bank for International Settlements, „Semiannual OTC derivatives statistics at end-June 2011," Listopad 2011. [Online]. Available: http://www.bis.org/statistics/derstats.htm.

[3] J. Davies, S. Sandstrom, A. Shorrocks i E. Wolff, „The World Distribution of Household Wealth," Mapping Global Inequalities, Center for Global, International and Regional Studies, UC Santa Cruz, 2007.

[4] „Solvency I Directive 73/239/EEC," 1973. [Online]. Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31973L0239:EN:HTML.

[5] „Solvency II Directive 2009/138/EC," 2009. [Online]. Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2009:335:0001:01:EN:HTML.

[6] M. Crouhy, D. Galai i R. Mark, Risk Management, New York: McGraw-Hill, 2001.

[7] R. Kaufmann, A. Gadmer i R. Klett, „Introduction To Dynamic Financial Analysis," *ASTIN Bulletin,* tom 31, nr 1, pp. 213-249, 2001.

[8] P. Blum i M. Dacorogna, „DFA - Dynamic Financial Analysis," *Teugels, J. and Sundt, B. (eds.): The Encyclopedia of Actuarial Science,* tom 1, pp. 505-519, 2004.

[9] B. Nicholls i J. Skinner, „Making use of Dynamic Financial Analysis," *Presented to the Institute of Actuaries of Australia,* 2007.

[10] Beusekom-Bastiaans, „Dynamic financial analysis: introduction to nonlife insurance decisions," *Paper of Business Mathematics and Informatics,* pp. 1-33, 2005.

[11] M. Eling, T. Parnitzke i H. Schmeiser, „Management Strategies and Dynamic Financial Analysis," *Variance,* tom 2, nr 1, pp. 54-66, 21 September 2006.

[12] M. Kaut i S. W. Wallace, „Evaluation of scenario generation methods for stochastic programming," *Pacific Journal of Optimization,* tom 3, nr 2, p. 257–271, 2007.

[13] M. Cumberworth, A. Hitchcox, W. McConnell i A. Smith, „Corporate Decisions in General Insurance: Beyond the Frontier," *British Actuarial Journal,* tom 6, nr 2, pp. 259-296, 2000.

[14] P. Bratley, B. L. Fox i L. E. Schrage, A Guide to Simulation, New York: Springer-Verlag, 1987.

[15] V. Chavez-Demoulin, S. A. Jarvis, R. Perera, A. S. A. Roehrl, S. W. Schmiedl i M. P. Sondergaard, „Extreme Datamining," *Proceedings of the 26th Annual Conference of the Gesellschaft Für Klassifikation E.V,* pp. 387-394, 2003.

[16] OMG, „Unified Modeling Language: Infrastructure, Version 2.2, formal/2009-02-04," 2009.

[17] L. Kotulski, Model wspomagania generacji oprogramowania w środowisku rozproszonym za pomocą gramatyk grafowych, Kraków: Wydawnictwo Uniwersytetu Jagiellońskiego, 2000.

[18] G. Rozenberg, Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations, River Edge, NJ: World Scientific, 1997.

[19] R. Henkel, H. Ehrig, G. Engels i G. Taentzer, „A view based approach to system modeling based on open graph transformation system," *In Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Language and Tools,* 1999.

[20] H. Ehrig, J. Kreowski, U. Montanari i G. Rozenberg, In Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallelism and Distribution, World Scientific, 1999.

[21] G. Taentzer, „Distributed Graphs and Graph Transformation," *Applied Categorical Structures 7(4),* pp. 431-462, 1999.

[22] L. Kotulski, „Graph representation of the nested software structure," *ICCS 2005, LNCS 3516,* pp. 1008-1011, 2005.

[23] Szpyrka, Matyasik, Mrówka, Kotulski i Balicki, „Formal Introduction to Alvis Modelling Language," *International Journal of Applied Mathematics and Computer Science,* 2011.

[24] L. Kotulski i M. Szpyrka, „Graph representation of the Hierarchical Alvis structure," *WORLDCOMP'11 Conference, Foundations of Computer Science,* pp. 95-101, 2011.

[25] E. Briys i F. de Varenne, Insurance: From Underwriting to Derivatives, Chichester: John Wiley & Sons, 2001.

[26] R. Cheng, „Validating and comparing simulation models using resampling," *Journal of Simulation,* tom 1, nr 1, pp. 53-63, December 2006.

[27] P. J. R. Pinheiro, J. M. Andrade e Silva i M. de Lourdes Centeno, „Bootstrap Methodology in Claim Reserving," *Journal of Risk & Insurance,* tom 70, nr 4, pp. 701-714, 2003.

[28] M. Kaut i S. W. Wallace, „Shape-based scenario generation using copulas," *Stochastic Programming E-Print Series,* nr 19, pp. 1-17, 2006.

[29] S. Yung-Ming, „Dynamic Financial Analysis in Insurance," *The American Risk and Insurance Association Annual Meeting,* pp. 6-9, August 2006.

[30] P. Artzner, F. Delbaen, J. Eber i D. Heath, „Thinking coherently," *Risk,* tom 10, pp. 68-71, November 1997.

[31] P. Artzner, F. Delbaen, J. Eber i D. Heath, „Coherent measures of risk," *Mathematical Finance,* tom 9, nr 3, p. 203–228, 1999.

[32] S. P. Lowe i J. N. Stanard, „An integrated dynamic financial analysis and decision support system for a property catastrophe reinsurer," *CAS Seminar on Dynamic Financial Analysis, Spring CAS Forum,* 2006.

[33] H. Markowitz, Portfolio Selection: Efficient Diversification of Investments, New York: John Wiley & Sons, 1959.

[34] E. Elton i M. Gruber, Nowoczesna teoria portfelowa i analiza papierów wartościowych, Warszawa: WIG-Press, 1998, pp. 97-98.

[35] S. Mitra, „Scenario generation for stochastic programming," *White paper, Optirisk Systems, UK,* p. 1–34, 2006.

[36] P. Embrechts, C. Kluppelberg i T. Mikosch, Modelling Extremal Events, Berlin et al: Springer, 2003.

[37] C. D. Daykin, T. Pentikainen i M. Pesonen, Practical Risk Theory for Actuaries, London: Chapman & Hall, 1994.

[38] S. W. Philbrick i R. A. Painter, „Dynamic Financial Analysis: DFA Insurance Company Case Study Part II: Capital Adequacy and Capital Allocation," *Casualty Actuarial Society Forum,* p. 99–152, 2001.

[39] N. Domenica, G. Birbilis, G. Mitra i P. Valente, „Stochastic programming and scenario generation within a simulation framework: an information systems perspective," *Decision Support Systems,* tom 42, nr 4, p. 2197–2218, 2007.

[40] H. Huang i T. R. Willemain, „Input modelling for financial simulations using the bootstrap," *Journal of Simulation,* tom 1, nr 1, pp. 39-52, December 2006.

[41] G. Albeanu, M. Ghica i F. Popentiu-Vladicescu, „On Using Bootstrap Scenario-Generation for Multi-Period Stochastic Programming Applications," *nternational Journal of Computers Communications and Control,* tom 3, pp. 282-286, 2008.

[42] M. R. Chernick, Bootstrap Methods. A Practitioner's Guide, New York: John Wiley & Sons, 1999.

[43] A. C. Davison i D. V. Hinkley, Bootstrap Methods and their Application, Cambridge: Cambridge University Press, 1997.

[44] O. F. Demirel i T. R. Willemain, „Generation of simulation input scenarios using bootstrap

methods," *Journal of the Operational Research Society,* tom 53, nr 1, pp. 69-78, 2002.

[45] B. Efron i R. J. Tibshirani, An Introduction to the Bootstrap, New York: Chapman and Hall, 1993.

[46] B. Efron, „Bootstrap Methods: Another Look at the Jackknife," *Annals of Statistics,* tom 7, nr 1, pp. 1-26, 1979.

[47] J. Fox, An R and S-PLUS Companion to Applied Regression. Appendix: Bootstrapping Regression Models, Thousand Oaks, CA: Sage Publications, 2002.

[48] T. R. Willemain, R. A. Bress i L. S. Halleck, „Enhanced simulation inference using bootstraps of historical inputs," *IIE Transactions,* tom 53, nr 9, p. 851–862, 2003.

[49] D. Diers, „Stochastic modelling of catastrophe risks in internal models," *GRIR,* tom 5, pp. 1-27, 2009.

[50] G. Woo, The mathematics of natural catastrophes, Imperial College, 1999.

[51] D. Whitaker, „Catastrophe Modelling," w *Rational Reinsurance Buying*, London, RISK Books, 2002, pp. 103 - 122.

[52] D. Toplek i M. Eling, „Modeling and Management of Nonlinear Dependencies: Copulas in Dynamic Financial Analysis," *Working Papers on Risk Management and Insurance,* tom 34, 2007.

[53] P. Embrechts, A. McNeil i D. Straumann, „Correlation and Dependency in Risk Management: Properties And Pitfalls," *Risk Management: Value at Risk and Beyond,* p. 176–224, 2002.

[54] H. Joe, Multivariate Models and Dependence Concepts, London: Chapman & Hall, 1997.

[55] F. C. Pereira, Bayesian Markov chain Monte Carlo methods in general insurance, PhD thesis, London: City University, 2000.

[56] L. Kotulski, „Preliminary Demands for the Support of Implementation Distributed Software System," *Elektrotechnika -Scientific Quarterly of Academy of Mining and Metallurgy in Cracow,* tom vol. 7, 1988.

[57] L. Kotulski i M. Flasiński, „Constructing Allocators for Distributed Environment with the Help of Graph Rewriting Systems," *Relcomex 89, Ossolineum,* pp. 115-120, 1989.

[58] M. Flasinski i L. Kotulski, „On the Use of Graph Grammars for the Control of a Distributed Software Allocation," *The Computer Journal,* tom 35, pp. A167-A175, 1992.

[59] H.-J. Kreowski i S. Kuske, „Graph Multiset Transformation as a Framework for Massively Parallel Computation," *ICGT 2008,* pp. 351-365, 2008.

[60] M. Flasiński, „On the parsing of deterministic graph languages for syntactic pattern recognition," *Pattern Recognition,* tom 26, nr I, 1993.

[61] L. Kotulski, Distributed Software Allocation with help node Label Controlled Graph Grammars, Jagiellonian University, Institute of Computer Science, 2003.

[62] J. W. Herrmann, C.-Y. Lee i J. L. Snowdon, „A classification of static scheduling problems," *Complexity in Numerical Optimization,* pp. 203-253, 1993.

[63] J. A. Stankovic, M. Spuri, M. Di Natale i G. Buttazzo, „Implications of Classical Scheduling Results For Real-Time Systems," *IEEE Computer,* tom 28, pp. 16-25, 1994.

[64] M. Tusiewicz, „Graph-based Model of Multi-agent System Environment," w *Międzynarodowe Warsztaty Doktoranckie, OWD*, Wisła, 2004.

[65] W. Schindler, „Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators," Certification body of the BSI, Section II, 1999.

[66] J. Lagarias, „Pseudorandom Number Generators in Cryptology and Number Theory," *Proceedings of Symposia in Applied Mathematics,* tom 42, pp. 115-143, 1990.

[67] L. van Zijl, „Random Number Generation with +-NFAs," *Lecture Notes In Computer Science,* nr 2494, pp. 263-273, 2001.

[68] W. Litwin, R. Moussa i T. Schwarz, „LH*RS - A Highly-Available Scalable Distributed Data Structure," *Transactions on Database Systems,* 2005.

[69] R. Moussa, W. Litwin i T. Schwarz, „LH*RS, A highly available distributed data storage system," *Proc. of the 30th VLDB Conference,* 2004.

[70] L. Kotulski, M. Tusiewicz i D. Dymek, „On the efficient dynamical financial analysis computation supported by UML(VR)," w *Business intelligence in economic forecasting: technologies and techniques*, J. Wang i S. Wang, Redaktorzy, New York, Hershey, 2010, p. 211–233.

[71] S. P. D'Arcy i R. Gorvett, „The Use of Dynamic Financial Analysis to Determine Whether an Optimal Growth Rate Exists for a Property-Liability Insurer," *Journal of Risk and Insurance,* tom 71, nr 4, p. 583–615, 2004.

[72] S. P. D'Arcy, R. W. Gorvett, J. A. Herbers, T. E. Hettinger, S. G. Lehmann i M. J. Miller, „Building a Public Access PC-Based DFA Model," *Casualty Actuarial Society Forum,* p. 1–40, Summer 1997.

[73] S. P. D'Arcy, R. W. Gorvett, T. E. Hettinger i R. J. Walling III, „Using the Public Access Dynamic Financial Analysis Model: A Case Study," *CAS Dynamic Financial Analysis Call Paper Program,* p. 53–118, 1998.

[74] V. Alwayn, Advanced MPLS Design and Implementation: PLS, VNPs, WAN Switching, Traffic Engineering and Quality of Service, Cisco Press, 2001.

[75] M. Lewis, Comparing, designing, and deploying VPNs, Adobe Press, 2006.

[76] D. Janssens, G. Rozenberg i R. Verraedt, „On sequential and parallel node-rewriting graph grammars," *Computer Vision, Graphics, and Image Processing, Volume 23, Issue 3,* pp. 295-312 , 1983.

[77] A. Piórkowski i J. Werewka, „Minimization of the total completion time for asynchronous transmission in a packet data-transmission system," *Applied Mathematics and Computer Science,* tom 20, nr 2, pp. 391-400, 2010.

[78] T. Cormen, C. Leiserson, R. Rivest i C. Stein, Wprowadzenie do algorytmów, Wydawnictwa Naukowo-Techniczne, 2007.

[79] C.-c. Lam, D. Cociorva, G. Baumgartner i P. Sadayappan, „Memory-optimal evaluation of expression trees involving large objects," *Proceedings of the 6th International Conference on High Performance Computing,* pp. 103-110, 1999.

[80] A. W. Appel i K. J. Supowit, „Generalization of the Sethi-Ullman algorithm for register allocation," *Software—Practice & Experience,* tom 17, pp. 417 - 421, June 1987.

[81] M. Tusiewicz, System wieloagentowy: teoria, projekt, implementacja oraz przyklady zastosowań, Praca magisterska, Kraków: Department of Mathematics, Physics and Computer Science, Jaggielonian University, 2003.

[82] M. Tusiewicz, „Distributed complete search with branch-and-bound for NP-hard problems," *Automatyka : półrocznik Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie,* tom 9, p. 245–253, 2005.

[83] M. Tusiewicz, „Multiagent framework with internal document flow," *Tools of information technology : proceedings of the 2nd conference,* p. 39–44, 2007.