AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa magisterska

Porównanie złożoności i efektywności operacji wykonywanych na relacyjnych bazach danych oraz sztucznych systemach skojarzeniowych.

Comparison of complexity and efficiency of operations performed on relational data bases and artificial associative systems.

Autor: *Michał Grygierzec*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr hab. Adrian Horzyk*

Kraków, 2014

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję Promotorowi, Recenzentowi oraz Rodzinie, a w szczególności mojej cudownej Żonie, której dedykuję tę pracę. Deo gratias!

Spis treści

1. Wprowadzenie 7

1.1. Cele pracy 8

1.1.1. Cel główny 8

1.1.2. Cele poboczne 8

1.2. Zawartość pracy 8

2. Obecny stan wiedzy 9

2.1. Relacyjne bazy danych 9

2.1.1. Opis składowych systemów DBMS 9

2.1.2. Zależności funkcyjne i normalizacja 10

2.2. Sieci neuronowe 13

2.2.1. Zasada działania 13

2.2.2. Model neuronu McCullocha-Pittsa 14

2.2.3. Metoda uczenia regułą Hebba 15

2.2.4. Rodzaje sieci neuronowych 16

2.3. Sztuczne systemy skojarzeniowe AAS 21

2.3.1. Opis systemów AAS 22

2.3.2. Składowe systemów AAS 23

2.3.3. Asocjacyjne grafowe struktury danych AGDS 25

3. Transformacja relacyjnych baz danych do postaci AGDS 29

3.1. Warstwa danych 29

3.1.1. Wymagania 29

3.1.2. Implementacja 30

3.2. Neurony 30

3.2.1. Motywacja 31

3.2.2. Implementacja 31

3.3. Transformacja relacyjnych BD do postaci AGDS 32

3.3.1. Opis algorytmu 33

3.3.2. Implementacja algorytmu ... 36
3.3.3. Redukcja wieloznaczności ... 40
3.4. Implementacja aplikacji webowej ... 41
3.4.1. Klient webowy ... 42
4. Porównanie efektywności ... 45
4.1. Przykłady wybranych operacji ... 45
4.1.1. Znajdowanie rekordów o zadanych wartościach atrybutów ... 45
4.1.2. Znajdowanie podobnych rekordów ... 48
4.1.3. Znajdowanie minimum i maksimum dla zadanego parametru ... 49
4.1.4. Sortowanie względem dowolnej ilości parametrów ... 50
4.1.5. Podsumowanie ... 51
5. Podsumowanie ... 53
5.1. Wnioski ... 53
5.2. Zrealizowane cele ... 53
5.3. Perspektywy ... 54

1. Wprowadzenie

Już w 1997 r. autorzy książki [UW01, str. 24-25] opisywali jak dużą rolę w poprawie szybkości przetwarzania danych odgrywa zrównoleglenie obliczeń. Przedstawili również wizję „pamięci trzeciego poziomu", w której ruchome ramiona robotów wyciągały właściwe nośniki danych (np. płyty CD) i umieszczały w czytniku. W dzisiejszych czasach nikt nie kwestionuje potrzeby przechowywania i przetwarzania danych w bazach danych, choć roboty przenoszące dyski optyczne (zwłaszcza przestarzałe CD-ROMy) raczej należą do rzadkości. Jednak coraz częściej kwestionowany jest sposób przechowywania i przetwarzania danych - popularność zyskują bazy nierelacyjne (tzw. noSQL). Zapewne jeszcze przez długi czas bazy relacyjne będą „królować", ale ich dominacja w ciągu nadchodzących lat staje pod znakiem zapytania.

Ilość składowanych informacji w skali świata jest niewyobrażalna i stale rośnie. W czasie pisania tej pracy na międzynarodowej konferencji „MongoDB World" odbywającej się w Nowym Jorku padły słowa, iż „90% danych z całego świata jeszcze 2 lata temu w ogóle nie istniało"[1]. Coraz więcej wiemy na temat otaczającej nas rzeczywistości, coraz więcej danych jesteśmy w stanie uzyskać poprzez obserwacje pewnych procesów, czy prowadzone badania. Cały czas człowiek stara się przetwarzać dane coraz efektywniej i szybciej. Jednakże ludzkość staje przed kolejnym niebagatelnym wyzwaniem - maszynowym, inteligentnym przetwarzaniem i interpretacją informacji - budowaniem wiedzy.

Ludzie poprzez skojarzenia potrafią być kreatywni, potrafią wiązać ze sobą pewne informacje i budować wiedzę. Jednakże nasza pamięć bywa zawodna - umysł nie jest w stanie konkurować z bazami danych w kwestii przechowywania danych, natomiast maszyny wciąż nie są inteligentne tak, jak ludzie, którzy błyskawicznie potrafią dokonywać skojarzeń. Gdyby mechanizm asocjacji przenieść na grunt baz danych, przetwarzanie i interpretacja danych byłyby efektywniejsze. Gdyby ponadto zrównoleglić te mechanizmy, wydajność takich operacji znacznie by wzrosła. Ta praca dyplomowa jest skromną próbą przedstawienia danych zgromadzonych w relacyjnej bazie danych w formie grafu i przeprowadzeniu na nim pewnych operacji.

[1] W wolnym tłumaczeniu z jęz. ang.

1.1. Cele pracy

1.1.1. Cel główny

Głównym celem pracy było zaimplementowanie asocjacyjnych grafowych struktur danych oraz porównanie ich efektywności w stosunku do relacyjnych baz danych. Z realizacją tego celu wiązało się opracowanie algorytmu transformującego potencjalnie dowolną bazę danych do postaci AGDS[2].

1.1.2. Cele poboczne

Ponadto wyznaczono również dodatkowe cele:

- zaimplementowanie aplikacji webowej, dzięki której można by zwizualizować otrzymany graf;
- poszerzenie wiedzy i umiejętności z zakresu inżynierii oprogramowania;
- zdobycie wiedzy z zakresu asocjacyjnej sztucznej inteligencji;
- przeprowadzenie eksperymentów w ramach pracy naukowej.

1.2. Zawartość pracy

W **rozdziale 2** przedstawiono aktualny stan wiedzy z dziedzin dotyczących tematu pracy - relacyjnych baz danych, sieci neuronowych i wywodzących się z nich sztucznych systemów skojarzeniowych.

W **rozdziale 3** zaprezentowano rozwiązanie postawionego problemu - zaprojektowane algorytmy, implementacje warstwy danych, neuronów, grafów AGDS oraz aplikacji webowej.

W **rozdziale 4** omówiono wybrane przykłady operacji dla relacyjnych baz danych i sztucznych systemów skojarzeniowych oraz przedstawiono wyniki analizy porównawczej.

Rozdział 5 podsumowuje wyprowadzone wnioski, zrealizowane cele oraz przedstawia perspektywy kontynuacji badań naukowych.

[2]Termin szczegółowo omówiony w rozdziale 2.3.3

2. Obecny stan wiedzy

Rozdział ten opisuje zagadnienia teoretyczne najistotniejsze z punktu widzenia realizowanego tematu: najpierw relacyjne bazy danych, poprzez sieci neuronowe, na sztucznych systemach skojarzeniowych skończywszy.

2.1. Relacyjne bazy danych

2.1.1. Opis składowych systemów DBMS

Relacyjne bazy danych są określane skrótowo terminem **DBMS** (z jęz. ang. ***D**ata**B**ase **M**anagement **S**ystem*). Jak czytamy w [UW01, rozdz. 1], „pierwsze profesjonalne systemy zarządzania bazami danych pojawiły się na rynku pod koniec lat sześćdziesiątych". Były wykorzystywane m.in. w systemach rezerwacji miejsc lotniczych, w systemach bankowych, czy też po prostu w dokumentowaniu działania przedsiębiorstw. W latach siedemdziesiątych pojawiła się koncepcja tworzenia w bazach danych relacji. Poniżej zostaną przedstawione kluczowe elementy baz danych, które zostaną wykorzystane w kolejnych rozdziałach. Ich szczegółowy opis można znaleźć w pozycji [GMUW06, str. 64-66].

Atrybut można zdefiniować jako właściwość *encji*, czyli abstrakcyjnego obiektu, któremu odpowiada relacja w bazie danych. Atrybut to po prostu kolumna danej relacji. Atrybuty mogą być różnych typów, m.in: liczby całkowite, zmiennoprzecinkowe, łańcuchy znaków, daty, wartości logiczne (*prawda* lub *fałsz*), itp. Typy te stanowią **dziedzinę** danego atrybutu. Atrybuty, które opisują daną relację, stanowią jej pierwszy wiersz. Wraz z nazwą relacji tworzą **schemat** danej relacji. W kolejnych wierszach znajdują się tzw. **krotki**, czyli rekordy danych. Przechowują wartości poszczególnych atrybutów. Zbiór krotek danej relacji nazywany jest **instancją relacji**. Instancja relacji często ulega zmianie w bazie danych ze względu na dodawanie, uaktualnianie i usuwanie krotek. Sam schemat relacji bardzo rzadko ulega zmianie, szczególnie jeśli w bazie znajdują się miliony rekordów, ponieważ wiązałoby się to z dodawaniem, usuwaniem atrybutów dla każdego z nich, co byłoby bardzo kosztowną operacją. Atrybuty, których wartości jednoznacznie identyfikują krotkę w instancji relacji nazywane są **kluczami**. Natomiast związki między relacjami są również reprezentowane poprzez relacje. W relacjach tych znajdują się klucze łączonych relacji i opcjonalnie dodatkowe atrybuty (tab. 2.3). Jeśli w schemacie relacji reprezentującej związek pomiędzy dwoma relacjami jedna z tych relacji występuje wielokrotnie, to konieczne jest przemianowanie atrybutów tak, by nazwy nie powtarzały się. Jest to bardzo ważna uwaga, która znajdzie

swoje odzwierciedlenie w omawianym później algorytmie transformacji relacyjnych baz danych do grafów AGDS.

Tablica 2.1: Relacja *Muzyk*

Id	Imię	Instrument	Data urodzenia
1	Miles Davis	trąbka	1944-05-26
2	Eric Clapton	gitara	1945-03-30

Tablica 2.2: Relacja *Zespół*

Id	Nazwa	Rodzaj muzyki	Data założenia
1	Jazz band	jazz	1946-04-23
2	Blues band	blues	1948-06-27

Tablica 2.3: Relacja reprezentująca związek wiele do wielu: Muzyk-Zespół

Muzyk	Zespół	Staż
1	1	2 lata
2	1	1 rok
2	2	3 lata

2.1.2. Zależności funkcyjne i normalizacja

W pozycji [GMUW06, str. 85] znaleźć można definicje zgodności atrybutów oraz zależności funkcyjnej.

Definicja 1 ***Zgodność atrybutów*** $A_1, A_2, ..., A_n$ *oznacza, że obie krotki mają takie same wartości składowych dla wymienionych atrybutów.*

Definicja 2 ***Zależność funkcyjna (FD)*** *w relacji R występuje, jeśli z tego, że dwie krotki tej relacji R są zgodne dla atrybutów* $A_1, A_2, ..., A_n$*, wynika zgodność owych krotek w pewnym innym atrybucie* B*.*

Można wyszczególnić następujące rodzaje zależności funkcyjnych [GMUW06, str. 94-95]:

trywialna – „jeśli zbiór złożony z atrybutów typu B jest podzbiorem atrybutów typu A";

nietrywialna – „jeśli co najmniej jeden z atrybutów typu B znajduje się pośród atrybutów typu A";

całkowicie nietrywialna – „jeśli żaden z atrybutów typu B nie znajduje się pośród atrybutów typu A";

Kolejnym ważnym zagadnieniem są klucze relacji (patrz [GMUW06, str. 87]):

Definicja 3 ***Klucz relacji*** *jest tworzony przez atrybut lub zbiór atrybutów* $\{A_1, A_2, ..., A_n\}$*, jeśli:*

1. *„Wszystkie pozostałe atrybuty relacji są funkcyjnie zależne od tych atrybutów. A zatem nie może się zdarzyć, aby dwie różne krotki relacji R były zgodne dla wszystkich atrybutów* $A_1, A_2, ..., A_n$
2. *„Nie istnieje taki podzbiór właściwy zbioru* $\{A_1, A_2, ..., A_n\}$*, od którego pozostałe atrybuty relacji R są zależne funkcyjnie, tzn. klucz musi być* minimalny.

Pojęcie **nadklucza** w uproszczeniu można zdefiniować jako klucz relacji, który nie spełnia warunku minimalności.

W dalszych rozważaniach zostanie również wykorzystane pojęcie **anomalii**. Zgodnie z opisem [GMUW06, str. 105-106] można wyszczególnić następujące ich rodzaje:

Redundancja – „Występuje, gdy w dwóch lub więcej krotkach dane są zduplikowane."

Anomalie modyfikacji – „dochodzi do nich wtedy, gdy dane zostają zmodyfikowane tylko w jednej z krotek, w których występują - są powielone. Za sprawą modyfikacji dane stają się niespójne. Można uniknąć takiej sytuacji dokonując dekompozycji i usuwając zbędną redundancję poprzez modyfikację schematów relacji."

Anomalie usunięć – „może wystąpić utrata części danych po zmianie wartości pewnego atrybutu na wartość pustą."

W jaki sposób można uniknąć powyższych anomalii? W pozycji [GMUW06, str. 108] czytamy: „(...) istnieje prosty warunek, którego spełnienie zapewnia, że w schemacie nie występują anomalie omówione powyżej. Warunek ten nazywa się postacią normalną *Boyce'a-Codda* lub w skrócie **BCNF** (*Boy-Codd normal form*). "

Definicja 4 *„Relacja R jest w postaci* ***BCNF*** *wtedy i tylko wtedy, gdy dla każdej nietrywialnej zależności* $A_1, A_2, ..., A_n \rightarrow B$ *zbiór* $\{A_1, A_2, ..., A_n\}$ *jest nadkluczem R."*

Innymi słowy warunek BCNF jest spełniony, wtedy i tylko wtedy, gdy lewa strona każdej funkcyjnej zależności nietrywialnej zawiera klucz.

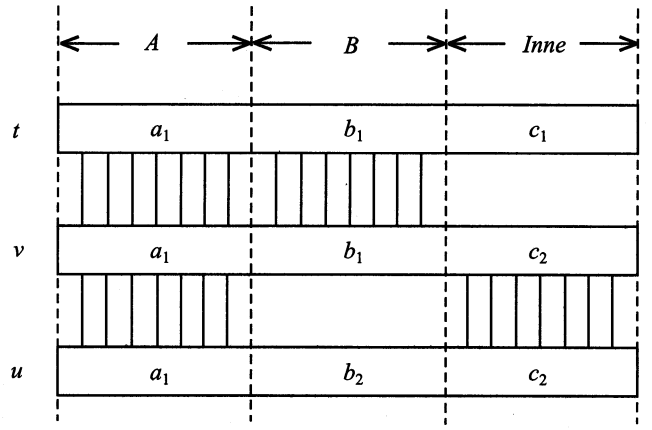
Kolejną rozważaną postacią normalną jest tzw. *trzecia postać normalna (3NF)*, która zachodzi dla relacji R, gdy spełniona jest implikacja [GMUW06, str. 119]:

„*Jeżeli* $A_1A_2 \cdots A_n \rightarrow B$ *jest zależnością nietrywialną, to albo* $\{A_1, A_2, ..., A_n\}$ *jest nadkluczem, albo* B *jest elementem pewnego klucza.*"

By móc przedstawić ostatnią ważną postać normalną, zostanie zdefiniowana zależność wielowartościowa ([GMUW06, str. 121, 123]). Ogólnie rzecz biorąc zależność wielowartościowa sprowadza się do niezależności dwóch lub większej ilości atrybutów. Formalnie można ją opisać następująco:

„Dla każdej pary krotek *t* i *u* z relacji *R*, które mają takie same wartości atrybutów typu A, można znaleźć w R taką krotkę *v*, której składowe mają wartości równe:

1. Wartościom atrybutów typu A w krotkach *t* oraz *u*.
2. Wartościom atrybutów typu B krotki *t*.
3. Wartościom tych składowych krotki *u*, które nie są ani typu A, ani typu B."



Rysunek 2.1: Zależność wielowartościowa. Źródło: [GMUW06, str. 123]

Zależność wielowartościową w relacji R można określić jako nietrywialną, jeśli spełnia dwa warunki ([GMUW06, str. 126]):

1. „Żaden atrybut typu B nie jest typu A".
2. „Każdy atrybut R jest albo typu A, albo typu B".

Na koniec przedstawiona zostanie czwarta postać normalna (4NF) [GMUW06, str. 126].

Definicja 5 *Relacja R jest w **czwartej postaci normalnej (4NF)** zawsze wtedy, kiedy wielowartościowa zależność $A_1A_2\cdots A_n \to\to B_1B_2\cdots B_n$ jest zależnością nietrywialną, a $\{A_1, A_2, ..., A_n\}$ jest nadkluczem w R.*

Dzięki tej postaci normalnej można uniknąć niepotrzebnej redundancji związanej z występowaniem zależności funkcyjnych lub zależności wielowartościowych. Jednakże dekompozycja relacji ma swoją cenę – ma miejsce rozdrobnienie danych i w momencie, gdy użytkownik bazy danych chce pobrać dane wykorzystując skomplikowane kwerendy zawierające wielokrotnie słowo kluczowe *JOIN* języka

SQL, czyli dokonuje tzw. *złączeń*, to wiąże się to z dodatkowym narzutem obliczeniowym. Poszukiwanie swego rodzaju „złotego środka" między denormalizacją, a dekompozycją zostało trafnie opisane w pozycji [EB05, str. 339]: Autorzy piszą, że w praktyce najczęściej nie normalizuje się baz danych do poziomów wyższych niż 3NF, BCNF lub 4NF. Co więcej, często z powodów wydajnościowych relacje pozostawia się w niższej postaci normalnej. Można stąd wysnuć wniosek, że w relacyjnych bazach danych właściwy schemat relacji będzie kompromisem między wydajnością, a brakiem redundancji.

2.2. Sieci neuronowe

W niniejszej pracy znajduje się również odwołanie do sieci neuronowych, ponieważ to z nich właśnie wywodzą się omawiane w następnej kolejności sztuczne systemy skojarzeniowe. W swojej książce, prof. Ryszard Tadeusiewicz – niekwestionowany autorytet w dziedzinie sieci neuronowych, zarówno w Polsce jak i na świecie – zwraca uwagę na bardzo istotną kwestię ([Tad93, str. 8]): „(...) najcenniejszą własnością sieci neuronowych jest ich zdolność do przetwarzania informacji w sposób **równoległy**, całkowicie odmienny od szeregowej pracy tradycyjnego komputera. Bardzo szybko stwierdzono także, że zasadniczym atutem sieci może być proces ich uczenia, zastępujący tradycyjne programowanie."

Warto zwrócić uwagę na dwie kluczowe kwestie, do których odniesienie nastąpi w rozdziale dotyczącym sztucznych systemów skojarzeniowych w kontekście struktur bazodanowych:

- **zdolność uczenia**, a więc brak potrzeby programowania takich sieci. Mają one umiejętność dostosowania się, adaptacji w miarę napływających do nich bodźców, czy też danych.
- **równoległość** - sposób przetwarzania danych, który znacznie poprawia efektywność i szybkość działania systemu.

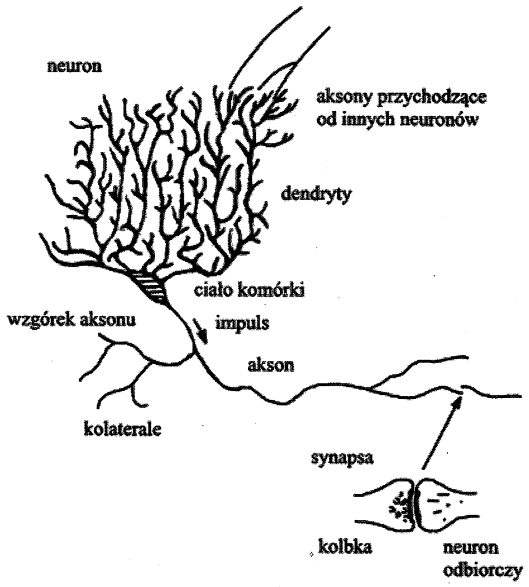
Teoria sieci neuronowych ewoluowała na przestrzeni lat, pojawiło się wiele kombinacji i wariantów takich sieci, a do najbardziej znanych należą m. in.: pojedyncze neurony liniowo przetwarzające sygnały, wielowarstwowe, nieliniowe sieci neuronowe, sieci CP przesyłające żetony, sieci rezonansowe, sieci Hopfielda, sieci pamięci skojarzeniowej, czy też sieci samoorganizujące się. Poniżej zostaną przedstawione najważniejsze kwestie dotyczące zasady działania neuronów oraz krótkie opisy działania różnych rodzajów sztucznych sieci neuronowych oraz metod ich uczenia.

2.2.1. Zasada działania

Praktycznie wszystkie modele neuronów miały swój pierwowzór w strukturach biologicznych mózgu – zasadniczo różniły się tylko głębokością i wiernością tego odwzorowania oraz wprowadzaniem pewnych zmian, czy rozszerzeń. Dlatego też analiza działania sztucznych sieci neuronowych zostanie poprzedzona krótką notką dotyczącą budowy i działania komórki nerwowej.

Neuron w uproszczeniu składa się z **dendrytów**, które dochodzą do ciała komórki, tzw. **somy**, z której wychodzi **akson**. Poprzez akson rozchodzi się impuls elektryczny aż do **synapsy**, w której zachodzą

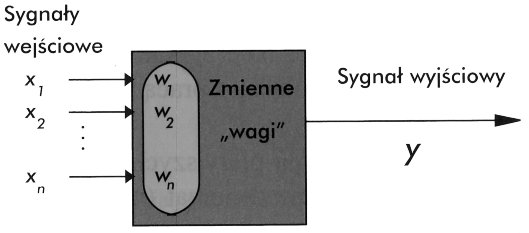
bardzo skomplikowane procesy biochemiczne i sygnał przekazywany jest do kolejnej komórki nerwowej. Sygnał elektryczny, który dochodzi do synapsy może spowodować uwolnienie się większej lub mniejszej ilości *neurotransmiterów*, czyli substancji przekazujących sygnał biochemiczny. Zmianie sygnału i ilości przekazywanych neurotransmiterów towarzyszą zmiany w metabolizmie komórki nerwowej. Można powiedzieć, że sygnał w pewnym sensie jest ważony, a komórka adaptuje się do zmieniającego się środowiska poprzez zmianę swojego rozmiaru i sposobów oraz intensywności reakcji, a także wielu innych parametrów, co przekłada się na sposób „ważenia" sygnałów. W pozycji [TGBL07, str. 35] autor zwraca szczególną uwagę na ten mechanizm – taki sposób działania neuronu umożliwia proces uczenia się. W jaki sposób biologiczny model przekłada się na teorię sztucznych sieci neuronowych? W następnym podrozdziale zostanie przedstawiona pierwsza formalna definicja z lat '40 ubiegłego wieku.



Rysunek 2.2: Budowa komórki nerwowej. Źródło: [TGBL07, str. 33]

2.2.2. Model neuronu McCullocha-Pittsa

Ogólnie rzecz biorąc sztuczny neuron McCullocha-Pittsa składa się z wejść (wektor **x**), wektora wag **w** oraz sygnału wyjściowego **y**, przy czym każdemu wejściu odpowiada jedna waga.



Rysunek 2.3: Sztuczny neuron - rozszerzenie modelu McCullocha-Pittsa. Źródło: [TGBL07, str. 32]

W modelu neuronu McCullocha-Pittsa sygnały wejściowe wchodzące w skład wektora x można ponumerować od 1 do n, wtedy: $x_i, i = 1, 2, ..., n$. Wartości wejściowe są binarne - mogą przyjmować wartości 0 lub 1, co oznacza obecność impulsu lub jego brak na danym wejściu w danej chwili. Reguła aktywacji neuronu wygląda następująco [uBJ96, str. 36]:

$$y^{k+1} = \begin{cases} 1, \text{gdy } \sum_{i=1}^{n} w_i x_i^k \geq T, \\ 0, \text{gdy } \sum_{i=1}^{n} w_i x_i^k < T, \end{cases} \tag{2.1}$$

k oznacza kolejne momenty czasu. w_i odpowiada i-tej wartości wagi, która odpowiada i-temu wejściu x_i. W modelu tym dla synaps pobudzających przyjęto $w_i = 1$, a dla hamujących $w_i = -1$. T oznacza próg aktywacji, po osiągnięciu którego neuron zadziała i na wyjściu y pojawi się odpowiedni sygnał równy 1 dla tego modelu. Mimo tak znacznego uproszczenia modelu neuronu i prostoty jego działania możliwe jest zrealizowanie za jego pomocą funkcji logicznych NOT, OR, AND, NOR lub NAND. Z algebry Boole'a wiadomo, że „(...) dowolnie złożony układ kombinacyjny można zbudować posługując się tylko funktorami NOR lub NAND", jak podają autorzy w pozycji [uBJ96, str. 36]. Niestety model ten dokonuje szeregu drastycznych uproszczeń. Po pierwsze wagi i progi aktywacyjne były raz ustawiane i niezmienne w czasie symulacji. Po drugie model nie zakładał istnienia interakcji pomiędzy elementami sieci - pozwalał tylko na przepływ sygnałów. Dlatego też z czasem zaczęły się pojawiać koncepcje, dzięki którym za pomocą sztucznych neuronów można było budować złożone i funkcjonalne sieci neuronowe, które potrafiły rozwiązywać szereg ciekawych zagadnień, m.in.: umożliwiono uczenie się sieci poprzez interakcję między jej elementami, która przejawiała się modyfikowaniem wag, ich adaptacją do zmieniającego się środowiska, a więc nowych danych napływających do niej.

Gdy prezentowany był biologiczny model neuronu, stwierdzono, że dla procesu uczenia kluczowa jest zmiana ilości neurotransmiterów, która odpowiada wagom synaptycznym w sztucznym neuronie. Jeśli wejścia i wyjścia neuronu mogą przyjmować różne wartości liczbowe (nie tylko 0 i 1), wprowadzone zostaną nieliniowe funkcje aktywacji (których dziedziną jest pobudzenie neuronu), to wachlarz możliwości znacznie się poszerzy. Jako że temat pracy dotyczy systemów skojarzeniowych, a więc innymi słowy systemów zdolnych do dokonywania asocjacji, poniżej zostanie przedstawiona koncepcja sieci neuronowych w charakterze pamięci asocjacyjnych, które były jednymi z pierwszych sieci, które próbowały modelować mechanizm dokonywania skojarzeń.

2.2.3. Metoda uczenia regułą Hebba

W analizie różnych rodzajów sieci asocjacyjnych pojawią się odwołania do reguły Hebba, która odegrała ogromną rolę w dziedzinie uczenia sieci neuronowych i doczekała się wielu modyfikacji. Reguła ta ma swoje źródło w neurobiologii [uBJ96, str. 60]:

„Jeżeli akson komórki A bierze systematycznie udział w pobudzaniu komórki B powodującym jej aktywację, to wywołuje to zmianę metaboliczną w jednej lub obu komórkach, prowadzącą do skuteczności pobudzania B przez A."

Jest to metoda uczenia bez nauczyciela, w której sygnałem uczącym jest sygnał wyjściowy. Wyjście neuronu y można opisać równaniem:

$$y_i = f(w_i^t x), \tag{2.2}$$

gdzie x to wejście neuronu, a w_i^t, to odpowiadająca mu waga, natomiast funkcja f jest nieliniowym odwzorowaniem pobudzenia neuronu. Wtedy przyrost Δw_i wektora wag wynosi:

$$\Delta w_i = \eta y_i x = f(w_i^t x)x \tag{2.3}$$

Przy czym najczęściej przyjmuje się początkowe wartości wag na wartości przypadkowe bliskie 0. W ten sposób podczas uczenia sieci neuronowej dla pewnych bodźców wzmacniane są te połączenia synaptyczne, które spowodowały również wysokie pobudzenie na wyjściu neuronu.

2.2.4. Rodzaje sieci neuronowych

Poniżej przedstawione zostaną przykłady sieci neuronowych, które zachowują się jak asocjacyjne magazyny pamięci - mają zdolność do „kojarzenia", „przypominania" sobie nauczonych wcześniej wzorców na podstawie wzorców pobudzających, które mogą być zniekształconymi wersjami obiektów z ciągu uczącego. Wyróżnia się dwa rodzaje sieci asocjacyjnych: sieci *autoasocjacyjne*, czyli odwzorowujące zbiór **X** w siebie, lub sieci *heteroasocjacyjne*, czyli odwzorowujące zbiór **X**, w inny zbiór **Y**.

Sieci rezonansowe ART

Określenie sposobu budowy sieci neuronowych ART pochodzi z jęz. ang. i oznacza *Adaptive Resonance Theory*. W pozycji [Tad93, str. 74] można znaleźć informację, że „konkretne metody wywodzące się z tej teorii są chronione patentem, którego posiadaczem jest Uniwersytet w Boston." Wyróżnia się dwa warianty sieci: ART1 oraz ART2, z czego pierwsza służy do analizy obrazów binarnych, natomiast druga dotyczy obrazów ciągłych, czy też innymi słowy obrazów analogowych. Poniżej zostanie tylko ogólnie przeanalizowana sieć ART1.

Sieć została zaproponowana jako ulepszona wersja sieci *Kohonena*, w której występował problem niestabilności podczas uczenia. Objawiało się to możliwością zmiany reprezentacji klasy przez neuron, który już jakiś obiekt reprezentuje, np: neuron rozpoznający literę *O*, w trakcie procesu uczenia w pewnym momencie „przestawia się" na rozpoznawanie litery *Q*. Niewątpliwie stanowi to problem w zadaniach klasyfikacji. Sieć rezonansowa wykorzystuje element rywalizacji w warstwie wyjściowej - neuron wyjściowy, który został najbardziej pobudzony wzmacnia wejścia, które doprowadziły do jego pobudzenia. Pozostałe wyjścia są zerowane tak, żeby nie zakłócać procesu uczenia - w ten sposób raz powiązany z daną rozpoznaną klasą neuron nie zmienia reprezentowanej przez siebie klasy.

Sieć składa się z dwóch warstw, w warstwie dolnej znajduje się N neuronów, a w warstwie górnej M. Liczność warstwy dolnej odpowiada ilości wejść - np. liczbie pikseli, które składają się na klasyfikowany, czy też właśnie zapamiętywany obraz. Pierwsza warstwa jest połączona z drugą warstwą na

zasadzie „każdy z każdym", natomiast wyjścia warstwy drugiej poprzez sprzężenia zwrotne są połączone również ze wszystkimi neuronami pierwszej warstwy. Połączenia dół–góra określa się mianem *pamięci długotrwałej* (wektor wag **w**), natomiast góra–dół *pamięci krótkotrwałej* (wektor wag **v**). Neurony z pierwszej warstwy są odpowiedzialne za dopasowanie wektora wejściowego **x** do wektora wag **w** standardowo poprzez obliczenie iloczynu skalarnego, natomiast w odwrotnym kierunku ma miejsce określenie stopnia podobieństwa tzw. *progu czujności (ang. vigilance treshold)* do stosunku wektora wag **v** i unormowanego wektora wejściowego.

W pozycji [uBJ96, str. 266-268] znajduje się czytelny schemat i opis algorytmu uczenia:

1. Najpierw należy przyjąć wartość progu czujności ρ w przedziale $(0, 1)$, a dla macierzy **W** oraz **V** o wymiarach $M \times N$ należy przyjąć wartości początkowe:
$$w_i j = \frac{1}{1+N}, \qquad v_{ij} = 1, \forall i, j \tag{2.4}$$

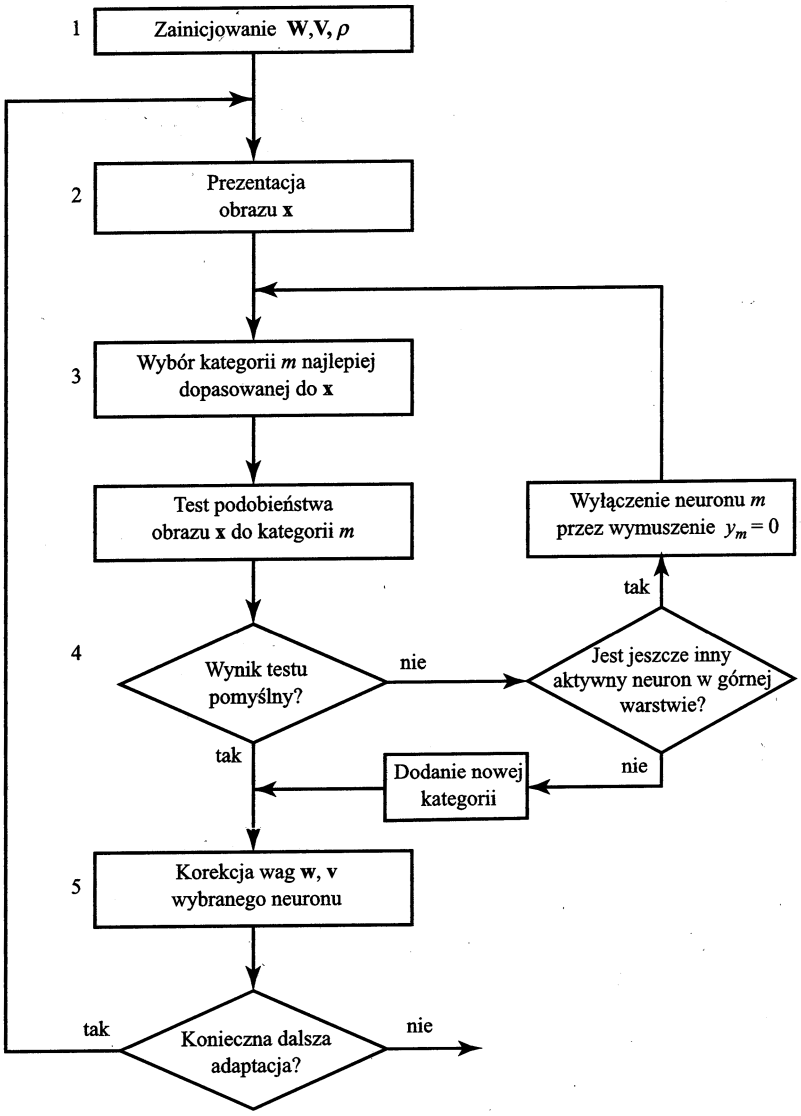
2. Następnie na wejście podawany jest obraz binarny $x_i \in 0, 1, i = 1, 2, \ldots, N$, a następnie obliczane są tzw. miary dopasowania:
$$y_j = \sum_{i=1}^{N} w_{ij} x_i, \qquad j = 1, 2, \ldots, M \tag{2.5}$$

3. W kolejnym kroku wybierana jest najlepiej dopasowana kategoria m, dla której
$$y_m = \max_{j=1,2,\ldots,M} (y_j) \tag{2.6}$$

4. Po wybraniu zwycięskiego neuronu przeprowadzany jest test podobieństwa:
$$p_m \stackrel{def}{=} \frac{\sum_{i=1}^{N} v_{im} x_i}{\sum_{i=1}^{N} x_i} > \rho \tag{2.7}$$
Jeśli test wypadł pomyślnie (czyli sieć jest w *rezonansie* z obrazem wejściowym), to obiekt zostaje zaliczony do danej klasy. Jeśli zaś wypadł niepomyślnie, a w górnej warstwie są aktywne inne neurony, to wyjście testowanego neuronu jest zerowane (neuron staje się nieaktywny) i algorytm kontynuuje działanie od kroku trzeciego. Jeśli zaś w górnej warstwie sieci nie ma już aktywnych neuronów, jest wybierany wolny neuron z indeksem m – będzie on przedstawicielem nowej klasy.

5. W ostatnim kroku korygowane są wagi v_{im} oraz w_{im} dla neuronu o indeksie m wybranego w kroku 4. Dla $i = 1, 2, \ldots, N$:
$$v_{im}(t+1) = v_{im}(t)\, x_i \tag{2.8a}$$
$$w_{im}(t+1) = \frac{v_{im}(t+1)}{0,5 + \sum_{i=1}^{N} v_{im}(t)\, x_i} \tag{2.8b}$$
Po aktualizacji wag następuje powrót do kroku 2.

Manipulując progiem czujności można osiągnąć większy stopień generalizacji, czyli mniejszą liczbę klas (ρ bliższe 0) lub bardziej szczegółowe rozróżnienie, czyli większą ilość klas (ρ bliższe 1).



Rysunek 2.4: Schemat blokowy algorytmu uczenia sieci rezonansowej ART1: [uBJ96, str. 267]

Sieci Hopfielda

Bardzo ciekawe właściwości, mimo stosunkowo prostej budowy, ma sieć Hopfielda ze względu na obecne w niej sprzężenia zwrotne (patrz rys. 2.5). Jest to sieć autoasocjacyjna, która pozwala na odtworzenie wyuczonych wzorców poprzez „skojarzenie" nieznanych kształtów z wcześniej przyswojonymi. Zastosowania tej sieci były różne od rozpoznawania obrazów, przez konstruowanie z nich przetworników analogowo-cyfrowych, po próbę rozwiązania problemu komiwojażera (TSP).

Sieć jest zbudowana tylko z jednej warstwy n neuronów, z progami T_i. Jeśli przyjęte zostaną następujące oznaczenia: **x** - wektor wejściowy, **T** - wektor progowy, a **net** - wektor pobudzeń neuronów, to ogólny wzór na pobudzenie neuronów takiej sieci w wersji macierzowej wygląda następująco:

$$\mathbf{net} = \mathbf{W}\mathbf{y} + \mathbf{x} - \mathbf{T}, \tag{2.9}$$

gdzie:

$$\mathbf{W} = \begin{bmatrix} \mathbf{w_1^T} \\ \mathbf{w_2^T} \\ \vdots \\ \mathbf{w_1^T} \end{bmatrix} = \begin{bmatrix} 0 & w_{12} & \dots & w_{1n} \\ w_{21} & 0 & \dots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & 0 \end{bmatrix} \tag{2.10}$$

Dzięki założeniu o symetryczności macierzy wag i wyzerowaniu jej diagonalnych elementów (zlikwidowanie autoasocjacyjności neuronów względem samych siebie), sieć Hopfielda zachowuje stabilność, jako układ dynamiczny. Będzie się to wiązało ze zmianą funkcji „energii" sieci, która będzie malała aż do osiągnięcia lokalnego minimum. Jeśli zaś chodzi o kolejne przejścia stanów w sieci ze sprzężeniami, używana jest rekurencyjna zależność:

$$y_i^{k+1} = \operatorname{sgn}\left(\mathbf{w}_i^T \mathbf{y}^k + x_i - T_i\right), \quad k = 1, 2, \dots \tag{2.11}$$

przy czym aktualizacja składowych wektora wyjść dokonuje się *asynchronicznie*, a więc w danym momencie tylko jedna składowa spośród n składowych ulega zmianie. Kolejne wyjścia są wybierane losowo. Można też zapisać powyższe równanie w postaci macierzowej:

$$y^{k+1} = \mathbf{\Gamma}\left[\mathbf{W}\mathbf{y}^k + \mathbf{x} - \mathbf{T}\right], \quad k = 1, 2, \dots \tag{2.12}$$

W rozważaniach dotyczących stabilności sieci wykorzystana zostanie tzw. *funkcja energetyczna*, *funkcja stanu*, lub też *funkcja Lapunowa*, jeśli zmiany jej wartości w trakcie działania algorytmu są niedodatnie.

$$E \stackrel{def}{=} -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{x}^T\mathbf{y} + \mathbf{T}^T\mathbf{y} \tag{2.13}$$

Gradient energii względem wektora wyjściowego prezentuje się następująco:

$$\nabla E = -\frac{1}{2}\left(\mathbf{W}^T + \mathbf{W}\right)\mathbf{y} - \mathbf{x} + \mathbf{T} \tag{2.14}$$

Jako że macierz wag jest symetryczna, można dokonać redukcji:

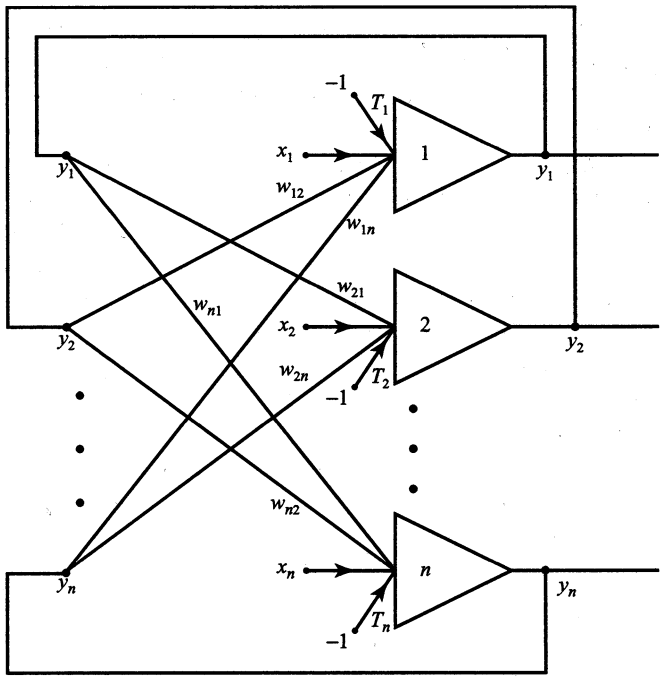
$$\nabla E = -\mathbf{W}\mathbf{y} - \mathbf{x} + \mathbf{T} \tag{2.15}$$

Następnie można wyprowadzić wzór na przyrost energii:

$$\Delta E = (\nabla E)^T \Delta\mathbf{y} = \left(-\mathbf{w}_i^T\mathbf{y} - \mathbf{x}_i + \mathbf{T}_i\right)\Delta\mathbf{y}_i = -net_i\Delta\mathbf{y}_i \tag{2.16}$$

Jako że przestrzeń wyjść jest ograniczona do wierzchołków n-wymiarowego sześcianu, można stwierdzić, że stan przejściowy w takim wypadku zawsze doprowadzi do punktu stabilnego, który nazywany jest rozwiązaniem granicznym równania 2.12.

$$y^* = \mathbf{\Gamma}\left(\mathbf{W}\mathbf{y}^* + \mathbf{x} - \mathbf{T}\right) \tag{2.17}$$



Rysunek 2.5: Schemat sieci Hopfielda. [uBJ96, str. 163]

Sieci neuronowe BAM

BAM (z jęz. ang. *Bidirectional Associative Memory*, czyli dwukierunkowa pamięć skojarzeniowa. W przeciwieństwie do poprzedniej sieci jest siecią heterogeniczną - jest zdolna do zapamiętywania i odtwarzania p wzorców **a** i odpowiadających im wzorców **b** w obu kierunkach: z a do b i na odwrót. Przy czym wektory są postaci [uBJ96, str. 222-224]:

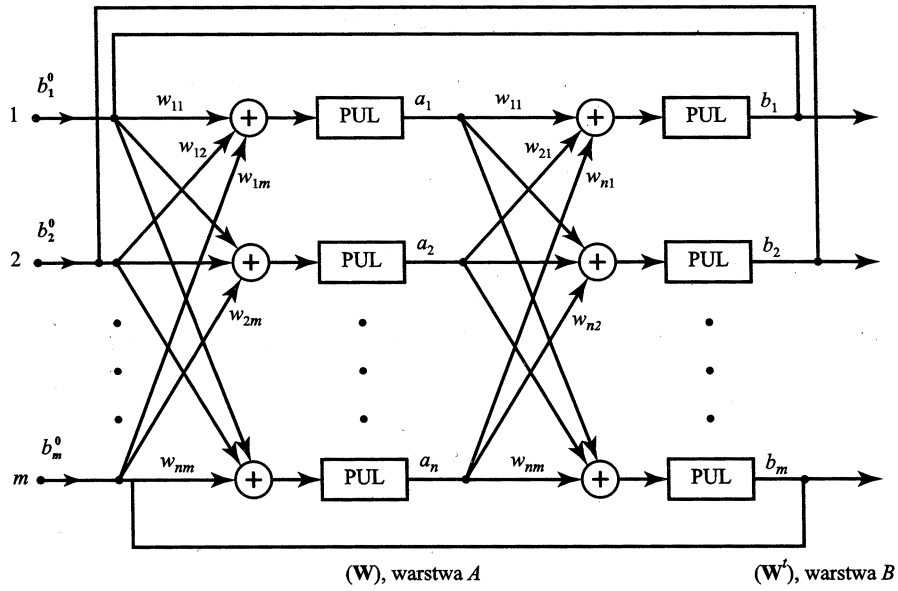
$$\left\{\left(a^{(1)}, b^{(1)}\right), \left(a^{(2)}, b^{(2)}\right), \ldots, \left(a^{(p)}, b^{(p)}\right)\right\}, \tag{2.18}$$

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, \quad a_i \pm 1, \quad i = 1, 2, \ldots, n, \quad \text{oraz} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \quad b_i \pm 1, \quad i = 1, 2, \ldots, m$$

Jeśli na wejście sieci zostanie podany wektor początkowy **b**, to na wyjściu warstwy A pojawi się wektor:

$$\mathbf{a}' = \mathbf{\Gamma}[\mathbf{W}\mathbf{b}] \tag{2.19}$$

gdzie Γ jest operatorem nieliniowym, który reprezentuje bipolarne funkcje aktywacji. Bardzo ważna jest zasada, która głosi, iż „zerowe pobudzenie łączne nie zmienia sygnału wyjściowego neuronu". Otrzymany wektor $\mathbf{a}'$ podawany jest na wejście warstwy B, na wyjściu której uzyskany zostanie wynik:



Rysunek 2.6: Schemat sieci BAM. [uBJ96, str. 223]

$$\mathbf{b}' = \mathbf{\Gamma}[\mathbf{W}^T\mathbf{a}'] \tag{2.20}$$

Z powyższych wzorów widać, że aby otrzymać wektor **a**, należy przemnożyć wektor **b** przez macierz **W**, natomiast aby otrzymać wektor **b**, trzeba przemnożyć wektor **a** przez transpozycję macierzy **W**. Modyfikacja wag sieci, czyli zapamiętywanie przez nią poszczególnych, skorelowanych ze sobą par odbywa się wg wzmiankowanej wcześniej reguły Hebba:

$$\mathbf{W} = \sum_{m=1}^{p} \mathbf{a}^m \mathbf{b}^{mT} \tag{2.21}$$

Dla sieci tego typu może się zdarzyć, że pewne stany nie będą stabilne. Co ciekawe, jeśli macierz **W** będzie macierzą kwadratową i symetryczną, to sieć zachowuje się jak poprzednio omawiana sieć autoasocjacyjna.

Podsumowanie

Powyższe rozważania dotyczące asocjacji i sieci asocjacyjnych nie wyczerpują tematyki z tym związanej, lecz stanowią wprowadzenie do opisu sztucznych systemów skojarzeniowych, gdzie asocjacje zachodzą nie tylko pomiędzy dwoma obiektami, lecz mogą być stosowane do wielu obiektów oraz ich sekwencji.

2.3. Sztuczne systemy skojarzeniowe AAS

Dotychczas przedstawiono różne rodzaje sieci neuronowych, które w pewien sposób próbują odtwarzać mechanizmy asocjacyjne, jednakże za każdym razem są to mechanizmy, które stanowią pewne

uproszczenie – nie potrafią np. dokonywać skojarzeń w kontekście czasu, czy też odwzorowywać złożonych sekwencji skojarzeń. W monografii [Hor13b] opisano dokładnie możliwości niespotykanych dotąd neuronowych, asocjacyjnych struktur. Wzmiankowana pozycja bibliograficzna stanowi doskonałe kompendium na temat asocjacyjnych sztucznych systemów skojarzeniowych, które otwierają nowy rozdział w dziedzinie badań nad sztuczną inteligencją.

2.3.1. Opis systemów AAS

Skrót AAS oznacza *artificial associative systems* ([Hor13b, str. 44-47]), czyli sztuczne systemy skojarzeniowe. Systemy te są wzorowane na swoich biologicznych odpowiednikach - czyli na ludzkim mózgu i zachodzących w nim procesach. Wprawdzie wzmiankowane wcześniej sieci neuronowe również były inspirowane naturą, to jednak różnią się fundamentalnie od AAS-ów. Ich wspólny mianownik stanowią oczywiście neurony, które są podstawą budowy systemów nerwowych, ale współczesne sieci neuronowe, takie jak opisane wcześniej sieci asocjacyjne korzystają z biologicznych mechanizmów w ogromnym uproszczeniu - pomijają wiele cech i funkcji, które mają bardzo ważne znaczenie dla funkcjonowania biologicznych układów nerwowych. Jednym z pominiętych aspektów jest *czas*. Mogłoby się wydawać, że działanie układu w czasie nic nie wnosi do ogólnego funkcjonowania modelu, a wręcz przeszkadza. Jednak, gdy przeanalizuje się sposób działania współczesnych komputerów i ludzkiego mózgu, można dojść do bardzo ciekawych wniosków, zaprzeczających temu.

Współczesne komputery wielokrotnie przewyższają mocą obliczeniową swoich „przodków" z ubiegłego wieku i potrafią dokonywać obliczeń nieosiągalnych w żaden sposób dla ludzkiego umysłu. Mimo to wciąż w wielu sytuacjach człowiek przewyższa maszynę, której wciąż, niezmiennie brakuje inteligencji na miarę ludzkiego umysłu. Chociażby weźmy pod uwagę fakt, że człowiek potrafi błyskawicznie rozpoznawać twarze członków rodziny, znajomych, czy innych ludzi, którzy w jakiś sposób są mu znani. Ludzie są w stanie w mgnieniu oka dokonywać skojarzeń, być kreatywni. Systemy AAS są właśnie próbą zbliżenia maszyn do sposobu myślenia i działania ludzkiego umysłu. W jaki sposób dokonuje się to w systemach AAS? Właśnie poprzez niedoceniany „czas", uwzględnianie go w modelu obliczeniowym sztucznych systemów skojarzeniowych. Biologiczne neurony mogą być aktywowane maksymalnie do 100 razy na sekundę - to jest nic nie znacząca liczba w porównaniu do możliwości współczesnych procesorów, a jednak umysł ludzki świetnie sobie radzi z wyzwaniami, jakie stawia przed nim świat. Neurony po aktywacji przez pewien czas nie mogą zostać ponownie aktywowane, będąc kolejno w stanie *refrakcji bezwzględnej*, a potem względnej. Okazuje się, że ten mechanizm ma kluczowe znaczenie dla powstawania alternatyw skojarzeniowych.

Możliwość dokonywania skojarzeń i budowania wiedzy w systemie poprzez skojarzenia daje ogromne możliwości szybkiego przetwarzania informacji, ale z pewnością nie zastąpi zupełnie tradycyjnych baz danych, w których dane są przechowywane pasywnie w sposób pewny i bezpieczny. Natura systemów skojarzeniowych jest nieco inna i stara się osiągnąć inne cele. Umysł ludzki wbrew pozorom nie jest magazynem pamięci - pamięć ludzka jest zawodna w przeciwieństwie do poprawnie działających (w sensie braku awarii) baz danych, które mogą przechowywać dane przez długie lata. Systemy

AAS budują swoją wiedzę i gromadzą informacje na podstawie napływających do nich danych w pewnym kontekście. Kontekst jest tu kluczowy i może być czasem bardzo złożony. Oprócz tego niebagatelne znaczenie ma kolejność, czy też częstotliwość prezentacji pewnych bodźców (danych).

W klasycznych algorytmach często budowane są różne struktury danych: drzewa, listy, kolejki, które są następnie iteracyjnie przeszukiwane w pętlach - często zagnieżdżonych. Dla pewnych zastosowań wiąże się to z niewygodną i niepraktyczną złożonością obliczeniową (problemy NP-zupełne [CLR01, str. 1022]). Jednym z popularnych problemów NP-zupełnych jest tzw. *problem komiwojażera (TSP)*. Jak się okazało jest on stosunkowo wydajnie rozwiązywany przez omawianą wcześniej sieć neuronową Hopfielda, która konkuruje z innymi aproksymacyjnymi algorytmami i jest w stanie wyznaczyć suboptymalne rozwiązanie ([Tad93, str. 126]). Jest to niezbity dowód na to, że wiele problemów może być znacznie wydajniej rozwiązywanych metodami inteligencji obliczeniowej takimi jak sieci neuronowe, czy właśnie przełomowe sztuczne systemy skojarzeniowe.

2.3.2. Składowe systemów AAS

Podobnie jak w sztucznych sieciach neuronowych, w systemach AAS również obecne są neurony zwane ASNami, czyli asocjacyjnymi neuronami (z jęz. ang. *associative neuron* [Hor13b, str. 52]. Reprezentują one asocjacyjne połączenia, czy też relacje pomiędzy elementami systemu AAS. Zostały wyszczególnione następujące asocjacyjne relacje:

- asocjacyjne podobieństwo (**ASIM** – *associative similarity*),
- asocjacyjne następstwo (**ASEQ** – *associative sequence*),
- asocjacyjny kontekst (**ACON** – *associative context*),
- asocjacyjne definiowanie (**ADEF** – *associative defining*),
- asocjacyjne tłumienie (**ASUP** – *associative suppression*).

Asocjacyjne podobieństwo ASIM

Są to relacje łączące podobne dane – ich układy, czy też kombinacje. Połączenia tego typu dotyczą danych, które są w sensie jakiejś metryki bliskie: np. sąsiadujące ze sobą liczby całkowite. Przy okazji omawiania połączeń warto zwrócić uwagę na ważną kwestię. W sztucznych systemach połączenia mogą być zarówno **jednostronne**, jak również **dwustronne**, wzajemne - te ostatnie mogą również modelować dwa lub wiele połączeń. Może to uprościć implementację i operowanie na takich konstrukcjach, a nie wpływa na pogorszenie działania sztucznych systemów – wręcz przeciwnie, redukuje ilość widocznych połączeń, więc przy okazji poprawia czytelność wizualizacji takiego systemu. Podobnie jak w modelach zwykłych neuronów z asocjacyjnymi połączeniami wiążą się wagi, które mogę być jednakowe w obu kierunkach (połączenia **homogeniczne**) lub różne (połączenia **heterogeniczne**).

Asocjacyjne następstwo ASEQ

Jeśli dane następują po sobie chronologicznie, lub występują po sobie w przestrzeni, to są łączone połączeniami asocjacyjnego następstwa ASEQ. Jeśli w trakcie uczenia, budowania wiedzy przez sieć pewne dane często po sobie występują, albo są zlokalizowane w bliskim sąsiedztwie w przestrzeni, to połączenia tego typu pomiędzy neuronami, które je reprezentują są wzmacniane. Dlatego odgrywają tak ważną rolę w procesie uczenia, ale także późniejszego rozumowania, czy wnioskowania.

Asocjacyjny kontekst ACON

Powiązania tego typu wydają mi się szczególnie ciekawe, ponieważ pozwalają na aktywację pewnych neuronów, pod wpływem wcześniejszej aktywacji innych neuronów, które stanowią kontekst dla tych właśnie aktywowanych. Można by zapytać, czym różnią się one od połączeń ASEQ, które również aktywują neurony, które w różnych przestrzeniach występują po sobie? Otóż połączenia ACON aktywują inne neurony nie bezpośrednio, ale w kolejnej aktywacji występującej w późniejszej chwili czasowej. Jako że wszystkie procesy w sieci asocjacyjnej rozgrywają się w czasie, umożliwia to zamodelowanie zjawisk dotyczących skojarzeń kontekstowych.

Asocjacyjne definiowanie ADEF

Są to połączenia występujące bezpośrednio między neuronami receptorycznymi lub innymi neuronami, które oddziałują równocześnie na definiowany neuron. Definiowane neurony mogą oddziaływać na inne neurony i w ten sposób powstaje złożona sieć zależności, w której coraz bardziej skomplikowane struktury definiują inne struktury - możliwości takich sieci są ogromne, a ich zastosowania bardzo ciekawe.

Asocjacyjne tłumienie ASUP

Ostatnim, niemniej bardzo istotnym rodzajem połączeń asocjacyjnych jest tłumienie. Służą one wyostrzeniu, uwydatnieniu jednych neuronów poprzez dyskryminowanie i tłumienie innych. Pozwala to na „automatyczne pokrycie przestrzeni danych reprezentowanymi kombinacjami lub układami danych" [Hor13b, str. 59]. Dzięki temu mechanizmowi możliwym staje się uniknięcie zbędnej redundancji reprezentacji pewnych klas, czy układów danych. Podobny problem występował w „tradycyjnych" sieciach neuronowych, gdzie w trakcie uczenia niektóre neurony wyjściowe mogły zmieniać typ rozpoznawanego przez siebie obiektu, lub dochodziło do sytuacji, w których kilka neuronów wyjściowych rozpoznawało tę samą klasę obiektu. Problem ten rozwiązano w sieciach rezonansowych ART, używając podobnej koncepcji, w której tylko jeden neuron mógł być „zwycięski", a pozostałe neurony wyjściowe były zerowane, czyli tłumione właśnie.

Powyższe rozważania przybliżają pokrótce wycinek obecnego stanu wiedzy dotyczącego elementów składowych asocjacyjnych sztucznych systemów skojarzeniowych. Systemy te są dokładnie i szczegółowo opisane w pozycji [Hor13b].

2.3.3. Asocjacyjne grafowe struktury danych AGDS

Na koniec zostanie przedstawiona kluczowa dla realizowanej pracy struktura – asocjacyjna grafowa struktura danych AGDS [Hor13b, rozdz. 5].

Grafowa asocjacyjna struktura danych AGDS (*associative graph data structure* - to graf, który przechowuje dane, ich kombinacje i relacje zachodzące pomiędzy nimi. Jest strukturą pasywną i statyczną podobnie jak bazy danych. Dlatego też w strukturze tej nie uwzględnia się w ogóle pojęcia czasu. Jest on wprowadzany w rozszerzeniach modelu AGDS, m. in. w aktywnych asocjacyjnych grafach neuronowych (AANG - *active associative neural graphs*), które wykraczają poza ramy niniejszej pracy. Mimo braku wykorzystania wszystkich (tj. bardziej zaawansowanych) asocjacyjnych mechanizmów, czy też połączeń takich jak ASUP, albo ACON, który sprowadza się w przypadku grafu AGDS do połączeń ASEQ, będących w rzeczywistości połączeniami ACON pierwszego stopnia (czyli występujące w bezpośrednim następstwie). Dlatego też w grafach AGDS występują połączenia ASIM, ADEF i ASEQ w miarę potrzeb. Dane przechowywane są jako węzły, natomiast asocjacyjne zależności występujące między nimi jako krawędzie. Węzły mogą być etykietowane i w ten sposób można nadawać im różne znaczenie. Na samym grafie można wykonywać operacje przeszukiwania wedle klasycznych metod, lub też można rozszerzyć graf AGDS do postaci AANG i skorzystać z wszystkich dobrodziejstw związanych z występowaniem czasu i innych zaawansowanych koncepcji asocjacyjnych sztucznych systemów skojarzeniowych. Do niewątpliwych zalet takich struktur należy praktycznie brak redundancji - wszystkie dane są unikalne i uporządkowane, przez co struktury te są oszczędne. Jednocześnie dzięki obecności połączeń między danymi, można dokonywać błyskawicznych operacji wyszukiwania danych skorelowanych, czy też wyłapywania różnic, i innych podobnych operacji. Struktury te wydają się być pewnego rodzaju złotym środkiem dla problemów dotyczących redundancji, czy wydajności omówionych w rozdziale 2.1.2. Jednakże grafów AGDS nie stosuje się w pewnych specyficznych zastosowaniach. Mianowicie nie wykorzystuje się ich do przechowywania zupełnie niepowiązanych ze sobą danych, dla których nie występują żadne interesujące związki. W takim wypadku pozostaje jedynie wykorzystanie tradycyjnych pasywnych baz danych (relacyjnych, dokumentowych, grafowych, czy jakichkolwiek innych).

Formalnie graf AGDS można przedstawić jako uporządkowaną siódemkę [Hor13b, str. 109]:

$$AGDS = (VV, VR, VS, VC, ESIM, ESEQ, EDEF) \tag{2.22}$$

gdzie $VV, VR, VS\ i\ VC$ są zbiorami wierzchołków, a $ESIM, ESEQ\ i\ EDEF$ to zbiory krawędzi definiowane następująco:

VV – zbiór wierzchołków reprezentujących pojedynczą wartość (*value vertex*);

VR – zbiór wierzchołków reprezentujących przedział wartości (*range vertex*);

VS – zbiór wierzchołków reprezentujących podzbiór wartości (*subset vertex*);

VC – zbiór wierzchołków reprezentujących kombinację wartości (*combination vertex*);

ESIM – zbiór krawędzi nieskierowanych (*undirected edge*) łączących asocjacyjnie podobne wierzchołki (*similarity edge*);

ESEQ – zbiór krawędzi skierowanych (*directed edges*) łączących asocjacyjnie następne wierzchołki (*sequence edge*);

EDEF – zbiór krawędzi dwustronnie skierowanych (*bidirected edges*) łączących wierzchołek definiujący z wierzchołkiem definiowanym (*defining edge*) w taki sposób, że inna etykieta (waga) określa przejście z od jednego do drugiego wierzchołka;

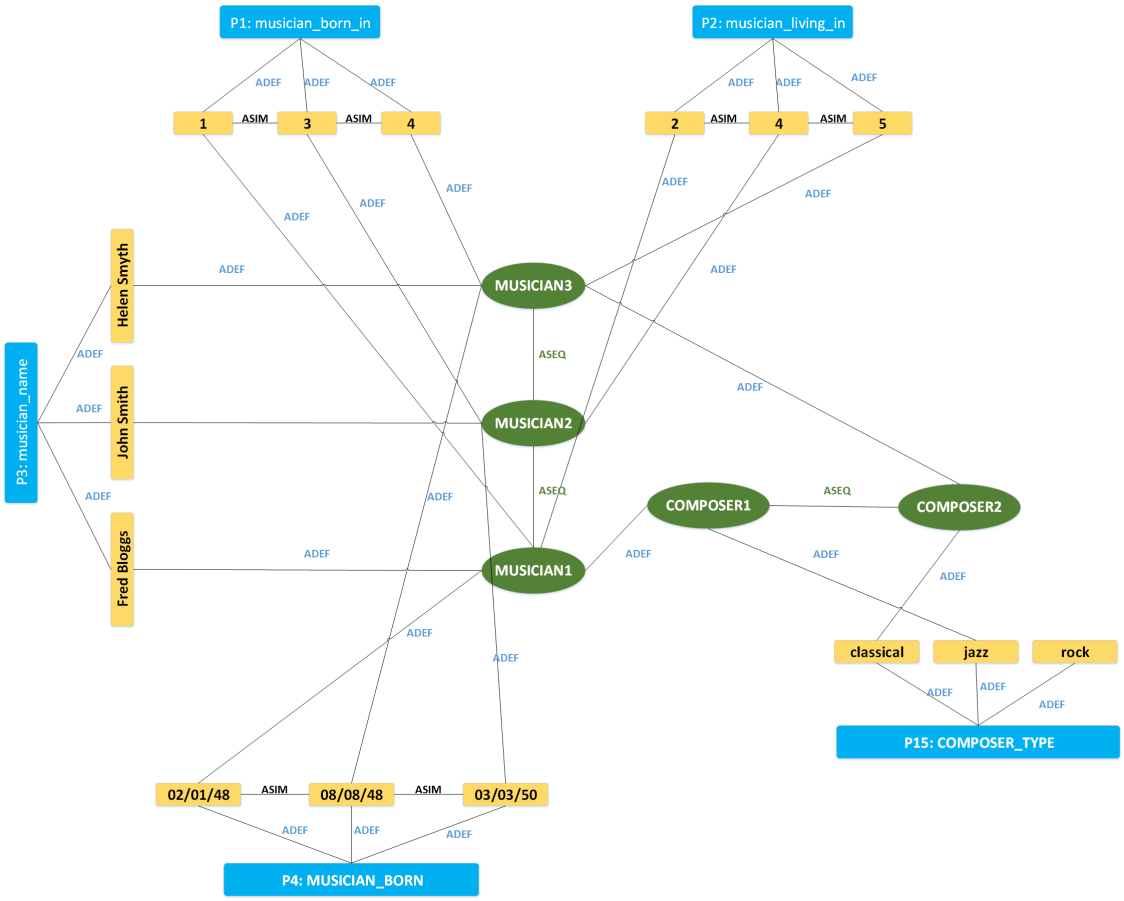
Poniżej zostaną również zdefiniowane elementy składowe grafów AANG, ponieważ wykorzystywana w nich nomenklatura zostanie użyta również do opisu grafów AGDS. Znacznie uprości to w przyszłości planowaną transformację grafów AGDS do postaci AANG.

SENSIN – wejście sensoryczne (*sensory input*)

VN – receptoryczny neuron wartości (*value neuron*) - reprezentuje wartość powiązaną z danym SENSINem;

SN – asocjacyjny neuron wzorca (*sample neuron*) - reprezentuje dane definiowane przez VNy i inne SNy

Na rysunku 2.7 przedstawiona jest przykładowa struktura grafu AGDS. Na zewnątrz grafu niebieskie prostokąty to wejścia sensoryczne (SENSINy). Elementy te definiują bezpośrednio neurony wartości (VNy), które są z nimi połączone asocjacyjnymi relacjami typu ADEF. Jeśli neurony te są odpowiedniego typu, to są połączone między sobą asocjacyjnymi połączeniami typu ASIM. Następnie neurony VN definiują asocjacyjne neurony wzorców SN. Niektóre SNy mogą być również definiowane przez inne SNy, co również zostało pokazane na załączonej ilustracji.



Rysunek 2.7: Przykładowy graf AGDS.

3. Transformacja relacyjnych baz danych do postaci AGDS

Rozdział ten opisuje w jaki sposób został zrealizowany temat pracy, a także przedstawia najważniejsze algorytmy i szczegóły implementacyjne.

Dla celów badawczych i realizacji założeń tej pracy, zostało opracowane oprogramowanie, które oparte zostało o platformę .NET w języku C# - zarówno tzw. back-end aplikacji webowej jak i logika biznesowa. Po stronie klienta został wykorzystany szereg frameworków i bibliotek pomocniczych opisanych w dalszej części rozdziału.

3.1. Warstwa danych

Aby dokonać transformacji potencjalnie dowolnej bazy danych do postaci grafowej niezbędna była implementacja odpowiedniej warstwy danych, która komunikowałaby się z konkretnym serwerem bazodanowym. Udostępnione w ten sposób dane byłyby wczytywane i przetwarzane przez kolejne moduły aplikacji.

3.1.1. Wymagania

Aby dobrze zaimplementować warstwę danych musiałem opracować właściwy projekt i architekturę. Podczas projektowania optymalnego rozwiązania kierowałem się zasadą, by uzależnić klasę od abstrakcyjnego interfejsu. Ponadto przyjąłem, że klasa ta powinna realizować wzorzec projektowy *Table Data Gateway* znany również pod nazwą *Data Access Object* [MM08, 634-646]. Wzorzec ten używa specjalnej formy *Fasady*. Należało również zdefiniować dla jakich obiektów DAO będzie Fasadą. Dlatego też opracowano następujący interfejs:

- `GetDBsNames` – metoda zwraca listę nazw baz danych dostępnych na serwerze, do którego została podłączona klasa.
- `GetTablesNamesForDB(string dbName)` – metoda zwraca listę nazw tabel dla danej bazy danych.
- `GetColumnsNamesForDB(string dbName)` – metoda zwraca słownik zawierający unikalne nazwy tabel i odpowiadające im listy nazw kolumn.

– `GetColumnsNamesForTable(string dbName, string tableName)` – metoda zwraca listę nazw kolumn dla danej tabeli znajdującej się w danej bazie danych.

– `GetForeignKeysForDB(string dbName)` – metoda zwraca listę instancji specjalnej struktury przechowującej dane dotyczące kluczy obcych dla danej bazy danych.

– `GetPrimaryKeysForDB(string dbName)` – metoda zwraca słownik, którego kluczami są nazwy tabel w danej bazie danych a wartościami listy nazw kolumn będących PK.

– `GetDataForTable(string dbName, string tableName)` – metoda zwraca listę wierszy, zawierających wartości odpowiadających poszczególnym kolumnom w danej tabeli w danej bazie danych

3.1.2. Implementacja

Jako że starałem się rozwijać oprogramowanie przyrostowo metodą *TDD* (*Test Driven Development*) [MM08, 73-82], rozpocząłem implementację od napisania testów jednostkowych. Dzięki takiemu podejściu mogłem w łatwy sposób kontrolować postępy prac i poprawność implementacji. Takie podejście wiązało się dodatkowo ze zwiększeniem komfortu samego procesu wytwarzania oprogramowania, ponieważ dawało większe poczucie pewności, że pisany kod będzie działał prawidłowo.

W obecnej wersji programu zaimplementowano jeden obiekt DAO dla bazy danych typu Sql Server. Dzięki zastosowaniu wzorca projektowego DAO i obecności abstrakcyjnego interfejsu z łatwością można było dodać kolejne implementacje dla innych silników bazodanowych.

Z implementacją tego modułu wiązało się też stworzenie specjalnej, dedykowanej struktury danych, która miała przechowywać informacje dotyczące kluczy obcych obecnych w bazie danych. Rzeczony obiekt został nazwany *TableMetadata*. Zawiera on pole, w którym przechowywana jest nazwa tabeli, której dotyczy oraz lista obiektów *ForeignKeyRelationship*, które dziedziczą z abstrakcyjnego interfejsu. Obiekty te zawierają informacje dotyczące jednej zależności typu FK (*Foreign Key*). Pierwsze pole nazwane zostało *ReferencingColumn* i oznacza nazwę kolumny, która odwołuje się do innej kolumny z innej tabeli. *ReferencedTable* to nazwa tabeli, do której odwołujemy się poprzez zależność FK, natomiast *ReferencedColumn* to nazwa kolumny, do której odnosi się klucz obcy. Zwrócenie tych informacji jest niezbędne do prawidłowego działania algorytmu omawianego w sekcji 3.3.

W ramach zainstalowanego oprogramowania *Sql Server* firmy *Microsoft* może istnieć wiele instancji silnika bazodanowego. Aby móc spośród nich wybierać bez konieczności rekompilacji źródeł, dodano plik konfiguracyjny *app.config*, w którym można zdefiniować tzw. *Connection String*, który zawiera informacje jak należy łączyć się z bazą danych.

3.2. Neurony

Kolejnym ważnym modułem był projekt odpowiedzialny za implementację neuronów.

3.2.1. Motywacja

Wprawdzie realizacja tematu wymagała zbudowania struktury AGDS, w której neurony nie występują, jednak w celu umożliwienia łatwego przejścia, czy też przekształcenia struktur AGDS do postaci AANG w przyszłości, zdecydowałem się nazwać je w taki sposób już na tym etapie. W ten sposób nazewnictwo sugeruje już pewne podobieństwo do struktur biologicznych, choć z implementacji nie wynika to wprost (np. brak metod aktywacji neuronów, itp.).

Ze względu na planowane w przyszłości rozszerzenia klas neuronów, postanowiłem wykorzystać wzorzec projektowy *Factory*. Wzorzec ten pozwala uniknąć zależności od konkretnych klas oraz wprowadza zależności względem klas abstrakcyjnych, co jest zgodne z zasadą odwracania zależności (DIP *Dependency Inversion Principle*), która głosi, że szczegółowe rozwiązania powinny zależeć od abstrakcji - nigdy na odwrót. Klasa realizująca wzorzec Factory pozwoliła na uniknięcie w kodzie wyrażeń zawierających słowo kluczowe **new**, a tym samym uzależnianie od instancji innych klas. Zamiast tego były zwracane interfejsy, czy też klasy abstrakcyjne.

3.2.2. Implementacja

Klasy *SN* (sample neuron), *VN* (value neuron) dziedziczą ze swoich interfejsów oraz z abstrakcyjnej bazowej klasy *Neuron*. Klasa *SENSIN* również dziedziczy ze swojego interfejsu, podobnie jak klasa *Synapsis* reprezentująca połączenia między neuronami. Zaimplementowano również wzorzec projektowy *Null Object* dla klasy *VN*.

Klasa abstrakcyjna Neuron

Klasa ta poza tym, że stanowi klasę bazową dla neuronów SN oraz VN zawiera też property *Cardinality*, czyli liczność, co zostanie szerzej omówione przy okazji prezentacji algorytmu transformującego relacyjne bazy danych do postaci AGDS. Oprócz tego w klasie tej jest obecna właściwość *Conenctions*, ponieważ każdy neuron jest połączony z innym obiektem. Jednym z wymagań dotyczących asocjacyjnych sztucznych systemów skojarzeniowych jest, aby każdy neuron, czy sensor brał czynny udział w procesach zachodzących w sieci - w systemach AAS nie mogą występować elementy „nieprzydatne".

Klasa VN

Klasa ta reprezentuje neuron wartości. Głównym zadaniem neuronów tego typu w przypadku zrealizowanej aplikacji jest przechowywanie wartości pewnych danych sensorycznych. Odbywa się to poprzez słowo kluczowe **dynamic**, które umożliwia programowanie dynamiczne w języku C#. Jednym z wymagań postawionych realizowanej aplikacji dyplomowej była transformacja „potencjalnie dowolnej bazy danych", dlatego też należało założyć, że nigdy nie będzie wiadomo, jakie typy danych trzeba będzie obsłużyć. Oprócz tego każdy neuron VN przechowuje informację do jakiego wejścia sensorycznego (czyli do SENSINa) należy.

Klasa SN

Neurony tej klasy biorą udział w definiowaniu kombinacji i układów danych. Najczęściej stanowią reprezentację pewnych rzeczywistych obiektów - np. rekordów tabeli w bazie danych. Posiadają dwie właściwości:

– Name – przechowuje nazwę neuronu SN na potrzeby struktury AGDS

– DisplayName – przechowuje „ładną" nazwę na potrzeby wizualizacji. Jest to nazwa potencjalnie krótsza i łatwiejsza w percepcji dla człowieka

Klasa SENSIN

Jest to ciekawa klasa o sporych możliwościach. Jej zadanie polega na przyjmowaniu bodźców z zewnątrz i tworzeniu odpowiadających im neuronów wartości VN. SENSIN może być kontenerem dla neuronów wartości jednego, ściśle określonego typu. Oznacza to, że jeśli do danego SENSINa przekazywane są liczby całkowite, to nie można w pewnym momencie podać mu łańcucha znaków, bo nie będzie „wiedział", jak go zinterpretować. W relacyjnych bazach danych w danej kolumnie zawsze muszą znajdować się dane tego samego typu, więc to założenie jest zawsze spełnione. Ponadto SENSIN przechowuje wszystkie VNy w posortowanej kolejności. Dlatego w każdym momencie jest w stanie wskazać neuron MINVN - o najmniejszej wartości oraz MAXVN - o największej wartości.

Klasa Synapsis

Klasa ta reprezentuje synapsę, czyli połączenie między neuronami. Jest to uogólnienie różnych typów połączeń występujących w grafie, omówionych w rozdziale 2.3.3. Klasa ta przechowuje informacje dotyczące tego jakie neurony łączy - jeden z nich jest określany mianem źródła (`Source`), natomiast drugi określany jest jako cel (`Target`). Oprócz tego w każdą stronę połączenie może charakteryzować się różnymi wagami synaptycznymi, oznaczanymi jako `WeightSourceTarget` w kierunku źródło - cel, oraz `WeightTargetSource` w kierunku cel - źródło. Jako że połączenia mogą być zarówno jednokierunkowe jak i dwukierunkowe, należało w pewien sposób to zaznaczyć. Zdecydowałem się w takiej sytuacji ustawić stosowną wartość wagi synaptycznej w danym kierunku na wartość `null`. Oznacza to, że jest to połączenie jednokierunkowe. W przeciwnym wypadku obie wagi mają przypisane wartości, które w szczególnym przypadku mogą być równe. Oprócz tego klasa *Synapsis* posiada jeszcze jedno pole - `ConnectionType`, które jest ustawiane na wartość „ADEF" lub „ASIM" w zależności od rodzaju połączenia.

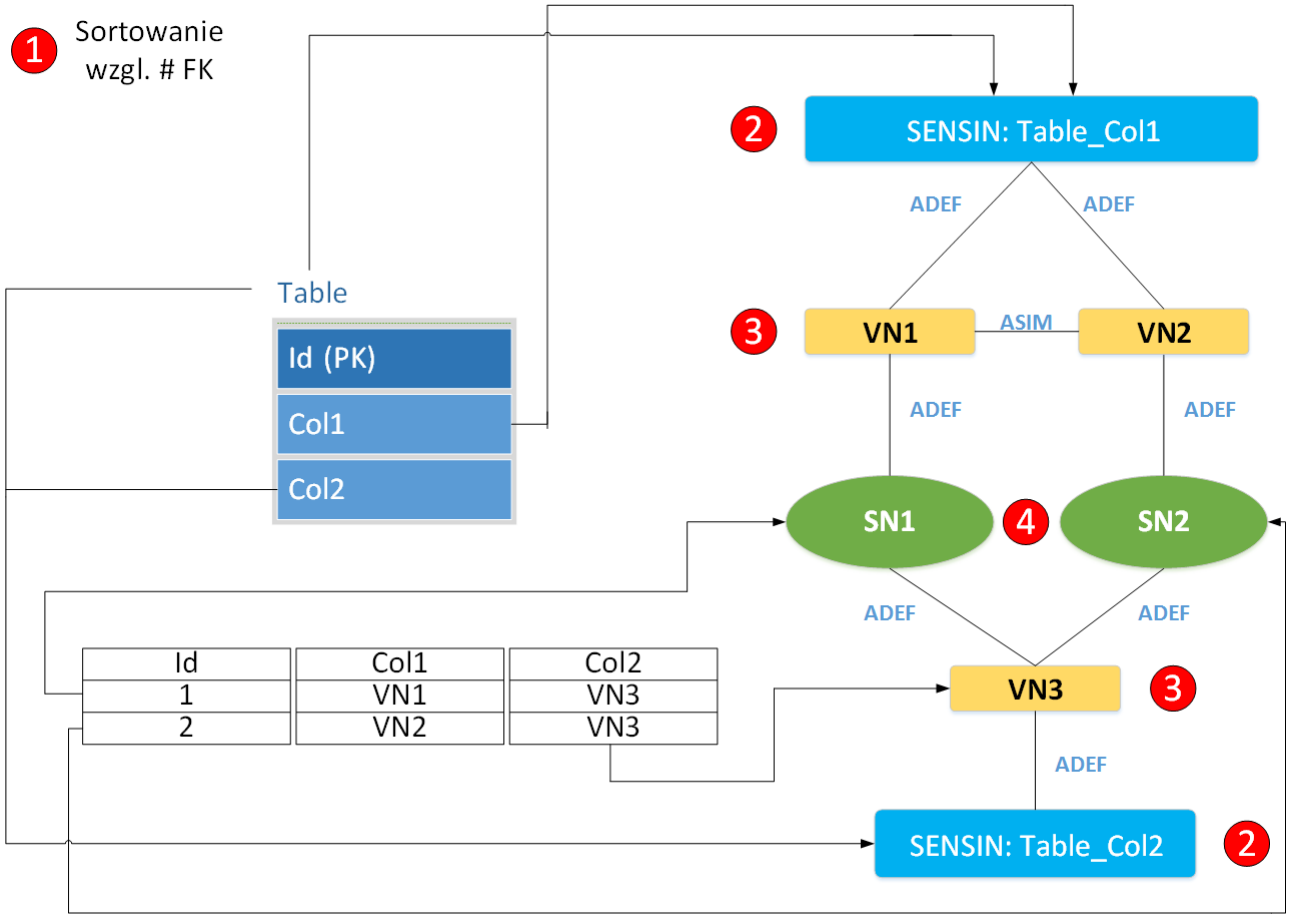
3.3. Transformacja relacyjnych BD do postaci AGDS

Po omówieniu wszystkich niezbędnych zagadnień teoretycznych oraz implementacyjnych może zostać przedstawione kluczowe zagadnienie, jakim jest transformacja potencjalnie dowolnej relacyjnej bazy danych do postaci grafu AGDS. „Potencjalna dowolność" jest związana z ograniczeniem nałożonym na typy danych, jakie mogą być przechowywane w tabelach. Na dzień dzisiejszy program obsługuje

wszystkie typy liczbowe zmienno i stało-przecinkowe, łańcuchy, znaki, wartości logiczne, a także daty. Żadne dane w formie binarnej, czy multimedialnej nie są obsługiwane.

3.3.1. Opis algorytmu

Algorytm transformacji relacyjnej bazy danych do grafowej postaci AGDS można opisać etapami. W celu wizualizacji, zaprezentowania algorytmu w formie graficznej przygotowano ilustrację (rys. 3.1).



Rysunek 3.1: Ilustracja algorytmu transformacji bazy danych na postać grafu AGDS.

1. Jako pierwsze odbywa się sortowanie tabel względem ilości występujących w nich odwołań do innych tabel, czyli obecności tzw. *kluczy obcych - FK* w porządku rosnącym. Oznacza to, że algorytm w kolejnych krokach jako pierwsze będzie analizował tabele, w których nie występują takie zależności (czyli dla których ilość zależności od innych tabel jest równa 0). Jako kolejne pojawią się tabele, w których takie zależności występują, ale nie pojawią się one w przypadkowej kolejności. Fundamentalne znaczenie ma to, żeby tabele wystąpiły w posortowanej liście zawsze po tabelach, od których zależą. Niespełnienie tego warunku doprowadziłoby do sytuacji, w której algorytm próbuje zbudować neurony, które zależą od nieistniejących neuronów. Wtedy należałoby

zaznaczyć taki brak, przejść do kolejnych tabel i gdy pojawi się brakująca tabela, zbudować dla niej neurony, a następnie wrócić do niespełnionych wcześniej zależności i uwzględnić je poprzez zbudowanie stosownych połączeń synaptycznych między neuronami. Takie podejście niepotrzebnie skomplikowałoby cały algorytm, dlatego tabele są sortowane. Szczegółowy opis zasad na jakich tabele są porównywane został przedstawiony w podrozdziale 3.3.2 w punkcie 1.

2. Dla każdej tabeli z posortowanego ciągu są następnie budowane SENSINy reprezentowane na rys. 3.1 przez niebieskie podłużne prostokąty. Przy czym jednej tabeli odpowiada tyle SENSINów, ile znajduje się w niej kolumn, nie będących kluczami głównymi (PK), ani obcymi (FK). Na rys. 3.1 widać niebieską encję *Table*, w której znajduje się jedna kolumna będąca kluczem głównym (*Id*) oraz dwie „zwykłe" kolumny (*Col1* oraz *Col2*). Dlatego też algorytm zbudował tylko 2 SENSINy – pominął kolumnę będącą kluczem głównym. Takie zachowanie jest podyktowane tym, że dane zgromadzone w kolumnach będących kluczami głównymi są unikalne – oznacza to, że nie mogą się w tabeli powtarzać krotki, dla których wartości z tej kolumny byłyby równe. Jeśli każda wartość w kolumnie jest inna, to nie może być mowy o żadnej agregacji, czy uogólnianiu danych z tej kolumny. W kontekście asocjacji dane te nie mają większej wartości, dlatego są wykorzystywane w inny sposób, co zostanie opisane przy okazji konstruowaniu neuronów wzorcowych SN. Nazwy SENSINów również są konstruowane w pewien ściśle określony sposób, wg następującej konwencji:

   Table – nazwa tabeli, dla której budowane są SENSINy.

   Col – nazwa kolumny, dla której budowany jest SENSIN.

   $$SENSIN\_NAME = Table + Col, \tag{3.1}$$

   przy czym nazwy tabeli i kolumny są oddzielone separatorem: „_". Dlatego też nazwa SENSINa odpowiadającego kolumnie *Col1* w tabeli *Table* została wyznaczona jako: *Table_Col1*. Dla drugiej kolumny analogicznie wyznaczona nazwa to: *Table_Col2*.

3. W kolejnym kroku algorytm iteruje po danych znajdujących się w tabeli i buduje neurony wartości (VNy) reprezentowane na rys. 3.1 przez żółte prostokąty. Elementy te są przyporządkowane do jednego SENSINa – zawierają się w nim. Jak było wzmiankowane w punkcie 2. SENSIN odpowiada jednej kolumnie w tabeli i właśnie wartości z tej kolumny są reprezentowane przez VNy. W tym momencie następuje całkowite usunięcie redundancji, ponieważ tworzone w ramach SENSINa VNy nie powtarzają się. Każdej wartości w kolumnie odpowiada VN, a jeśli wartości się powtarzają, to inkrementowany jest specjalny licznik w danym VNie, który reprezentuje ile razy dana wartość wystąpiła w danej kolumnie w danej tabeli. Na rysunku widać, że do SENSINa *Table_Col1* przyporządkowane są dwa VNy: *VN1* oraz *VN2*. Do SENSINa *Table_Col2* przyporządkowany jest tylko jeden VN – *VN3*, reprezentujący wartość *VN3*, która powtórzyła się w kolumnie *Col2* dwa razy. Wartości VNów muszą być tego samego typu w ramach jednego SENSINa, tak jak w relacyjnej bazie danych typ kolumny w schemacie tabeli jest ściśle określony i obowiązuje

dla wszystkich wartości w tej kolumnie. W tym przypadku wszystkie są łańcuchami znaków, ale mogłyby być równie dobrze liczbami, czy datami. Jeśli da się określić i w jakiś sposób zmierzyć podobieństwo między neuronami wartości, to łączone są one połączeniem *ASIM*. Jest to homogeniczne dwustronne połączenie, którego waga obliczana jest wg wzoru:

$$w_{VN,\tilde{VN}} = 1 - \frac{|v - \tilde{v}|}{R}, \tag{3.2}$$

gdzie

v – to wartość VNa VN

$\tilde{vn}$ – to wartość VNa $\tilde{VN}$

R – to przedział liczony dla całego SENSINa, definiowany jako różnica między największym neuronem VN – VNMAX i najmniejszym neuronem VN – VNMIN w danym SENSINie

Bardzo ważne jest, aby aktualizować wagi połączeń ASIM dla wszystkich neuronów wartości VN występujących w SENSINie, gdy zmieniają się skrajne neurony VN (MIN albo MAX), ponieważ zmienia się wtedy zakres dla całego SENSINa.

4. Ostatni etap polega na tworzeniu neuronu wzorca SN, który reprezentuje wiersz w tabeli i połączeniu go z definiującymi go neuronami wartości VN, które reprezentują wartości w kolumnach tabeli. Neurony SN są reprezentowane na rys. 3.1 przez zielone elipsy. W omawianym przykładzie neuron *SN1* jest definiowany przez neurony wartości *VN1* oraz *VN3*, natomiast neuron *SN2* określają neurony *VN2* oraz *VN3*. Pomiędzy poszczególnymi neuronami tworzone są specjalne połączenia synaptyczne typu „ADEF" o określonych wagach (wzór i mechanizm jest szczegółowo opisany w sekcji 3.3.2 w punkcie 7). Jeśli w tabeli występują kolumny typu PK, to ich wartości są wykorzystane do nazywania SNów (mechanizm również szczegółowo opisany w sekcji 3.3.2 w punkcie 4). W omawianym przykładzie właściwość *DisplayName* dla wierzchołka *SN1* przyjęłaby wartość: *Table_Id_1*, a dla *SN2*: *Table_Id_2*. Natomiast właściwość *Name* dla neuronu *SN1* przyjęłaby dłuższą, bardziej szczegółową wartość: *Table_Id_1_Col1_VN1_Col2_VN3*. Ta sama właściwość dla neuronu *SN2*: *Table_Id_2_Col1_VN2_Col2_VN3*.

Powyższy algorytm opisuje najprostszy przypadek, gdy w tabeli nie ma kluczy obcych (FK) i nie występuje dla tych kluczy wieloznaczność. Jeśli w tabeli są klucze obce, to algorytm łącząc neurony definiujące SN, musi znaleźć neuron SN, który odpowiada wierszowi tabeli, do którego odnosi się pole w kolumnie będącej kluczem obcym. Ilustrują to rysunki 3.2 oraz 3.3, natomiast szczegółowe omówienie implementacji znajduje się w sekcji 3.3.2 w punkcie 5.

W przypadku wystąpienia wieloznaczności (co zostało zilustrowanie na rysunkach 3.4 oraz 3.5) trzeba utworzyć **neuron „obcy"**. Neuron ten (zaznaczony kolorem fioletowym) pozwala uniknąć wieloznaczności, co również zostało szczegółowo opisane w sekcji 3.3.3. Przyjęto nazwę sugerującą zależność typu *klucz **obcy** (FK)*, ponieważ tylko dla takich zależności, w specyficznych przypadkach neurony te muszą być tworzone.

3.3.2. Implementacja algorytmu

Algorytm został zaimplementowany z wykorzystaniem wielu funkcji wyszczególnionych i opisanych dokładnie w kolejnych punktach.

1. Sortowanie tabel względem zależności FKs

Pierwszym krokiem na drodze do otrzymania grafu AGDS jest zidentyfikowanie zależności reprezentowanych przez obecność kluczy obcych w bazie. Dzięki poprawnie zaimplementowanej warstwie danych, uzyskujemy specjalne struktury, które zawierają wszystkie niezbędne informacje – które tabele i które kolumny są związane ze sobą poprzez klucze obce. Dla struktur zawierających dane na temat zależności FKs zaimplementowana została specjalna klasa, która dziedziczy po interfejsie `IComparer<T>` i służy do porównywania między sobą dwóch obiektów typu `T` - w tym przypadku jest to typ `ITableMetadata`, czyli interfejs specjalnej struktury, o której mowa.

Porównywanie struktur x oraz y typu `ITableMetadata` odbywa się na następujących zasadach:

1. Wartości null są sobie równe i wartość null jest zawsze mniejsza od prawidłowej wartości (nie null).

2. Jeśli ani x ani y nie mają żadnych zależności typu FK, są sortowane leksykograficznie względem nazw tabel.

3. Jeśli x ma 1 lub więcej zależności a y nie, to x jest większe.

4. Jeśli y ma 1 lub więcej zależności a x nie, to y jest większe.

5. Sprawdzany jest warunek, czy może x zależy od y (odwołuje się do y poprzez klucz obcy FK), jeśli tak to x jest większe.

6. Sprawdzany jest warunek, czy może y zależy od x (odwołuje się do x poprzez klucz obcy FK), jeśli tak to y jest większe.

7. Jeśli zarówno x jak i y posiadają zależności, ale są one zbiorami rozłącznymi, to ten obiekt, który ma więcej zależności jest uznawany za większy.

8. Jeśli oba obiekty posiadają równoliczne, rozłączne zbiory zależności, to są sortowane leksykograficznie względem nazw tabel.

Gdy algorytm budowania grafu AGDS dysponuje już posortowaną rosnąco tabelą obiektów TableMetadata wg wyżej zaproponowanych zasad, można przystąpić do konstruowania odpowiedniej struktury danych iterując po posortowanej liście i wykonując kroki opisane szczegółowo w kolejnych podrozdziałach.

2. Budowanie SENSINów dla danej tabeli

Na podstawie informacji dotyczących danej tabeli o kluczach obcych, liście kolumn, oraz liście kluczy głównych PK konstruowany jest SENSIN w ten sposób, że program iteruje po wszystkich kolumnach danej tabeli i podejmuje działania wg następujących reguł:

1. Jeśli dana kolumna jest kluczem głównym PK (unikalna wartość w skali tabeli) przejdź do następnej kolumny

2. Jeśli dana kolumna jest kluczem obcym FK przejdź do następnej kolumny

3. Jeśli dana kolumna nie jest ani PK ani FK zbuduj nowy SENSIN o nazwie nazwaTabeli_nazwaKolumny i dodaj do kontenera struktury AGDS, zawierającego wszystkie SENSINy, jeśli SENSIN o takiej nazwie nie pojawił się wcześniej.

W ten sposób każda kolumna zawierająca faktyczne dane, będzie posiadała swój odpowiednik w grafie AGDS w formie wejścia receptorycznego (SENSIN).

3. Pobranie danych dla danej tabeli

W kolejnym kroku program iteruje po wszystkich wierszach danej tabeli wykonując następujące działania:

4. Utworzenie neuronu SN dla wiersza tabeli

Dla każdego wiersza tabeli tworzony jest odpowiadający mu neuron wzorca SN, którego właściwości *Name* oraz *DisplayName* są początkowo ustawiane na nazwę tabeli. Oprócz tego tworzony jest licznik `countNeuronsConnectedToNewSN` neuronów połączonych z właśnie utworzonym neuronem SN i ustawiany jest na 0. Następnie dla każdej kolumny wiersza odpowiadającego właśnie utworzonemu neuronowi SN przeprowadzane są następujące operacje:

1. Do właściwości *Name* neuronu SN jest dodawany łańcuch znaków wg wzoru: „_nazwaKolumny_wartośćKolumny”.

2. Jeśli dana kolumna jest kluczem głównym (PK), to do właściwości *DisplayName* dodawany jest łańcuch znaków: „_nazwaKolumnyPK_wartośćKolumny”.

3. Jeśli dana kolumna jest kluczem obcym (FK), to inkrementowany jest licznik `countNeuronsConnectedToNewSN` oraz aktualny neuron łączony jest z neuronem go definiującym (patrz punkt 5).

4. Jeśli dana kolumna nie jest ani PK ani FK, tworzony jest neuron VN i synapsa łącząca go z budowanym właśnie SNem (patrz punkt 6). Jeśli operacja budowania VNa się powiodła, inkrementowany jest licznik `countNeuronsConnectedToNewSN`.

Jeśli po przejściu w pętli po wszystkich kolumnach okaże się, że żadna kolumna nie była PK, to property *DisplayName* neuronu SN jest ustawiane na wartość z property *Name*. Na koniec na podstawie wartości licznika `countNeuronsConnectedToNewSN` uaktualniane są wagi synaps łączących neurony definiujące nowo powstały neuron SN z nim samym (patrz punkt 7).

5. Łączenie neuronu SN z neuronem go definiującym

Jeśli do tabeli, której jeden z wierszy reprezentowany jest przez neuron definiujący nowy SN, istnieje więcej niż jedno odwołanie z tabeli, której jednemu z wierszy odpowiada nowo utworzony neuron SN, to mamy do czynienia z wieloznacznością. Sposób postępowania w danym przypadku jest objaśniony w sekcji 3.3.3. W przeciwnym przypadku tworzone jest połączenie synaptyczne między neuronem definiującym a definiowanym tak, że definiujący jest źródłem, a definiowany celem. Waga ze źródła do celu ustawiana jest na wartość *null*, natomiast w odwrotnym kierunku przyjmuje się wartość 1. Typ połączenia ustalany jest na „ADEF". Synapsa dodawana jest do kontenerów synaptycznych obu neuronów oraz do kontenera synaptycznego grafu AGDS.

6. Dodawanie nowego VNa i synapsy łączącej go z SNem

1. Wyszukiwany jest SENSIN na podstawie danej nazwy tabeli i kolumny, z której pochodzi dodawana wartość.

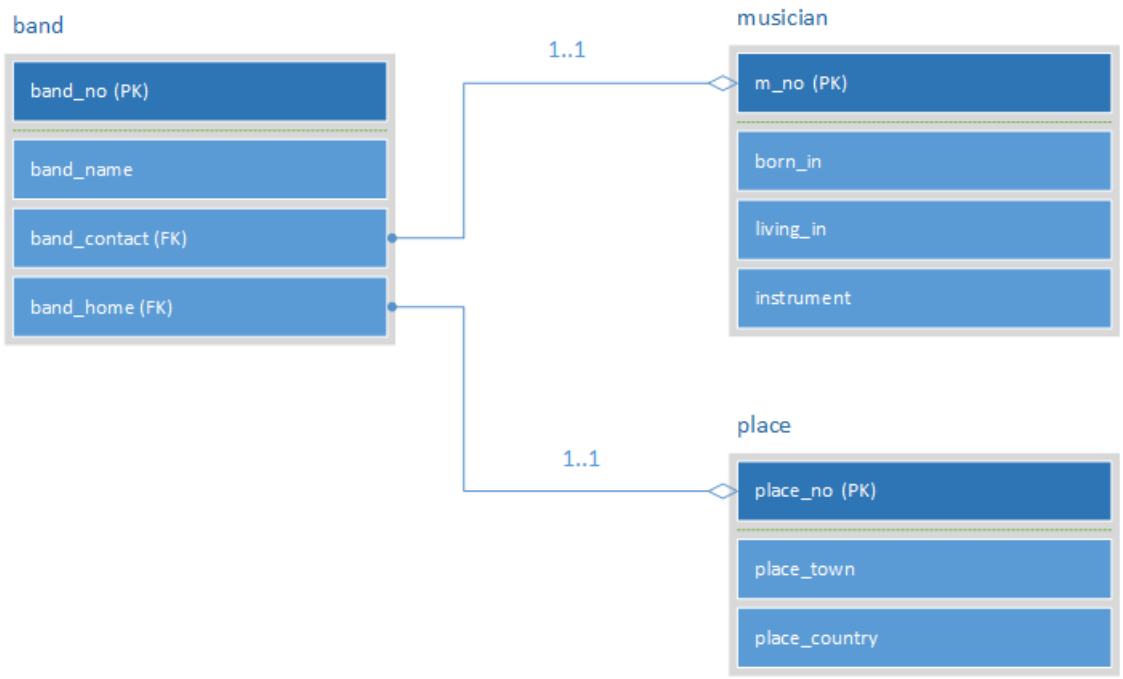
2. Do SENSINa dodawana jest nowa wartość. Jeśli wartość nie jest nullem, a istnieje już w SENSINie neuron VN reprezentujący tę wartość, to inkrementowana jest właściwość `Cardinality` tego neuronu VN i jest on zwracany przez SENSIN. Jeśli neuron VN reprezentujący daną wartość nie istniał w SENSINie wcześniej, to jest tworzony, dodawany do SENSINa i zwracany. Jeśli wartość jest błędna zwracany jest VN.NULL.

3. Jeśli zwrócony neuron VN jest poprawny (nie jest neuronem VN.NULL), to tworzone jest połączenie typu „ADEF" między VNem, który jest źródłem, a nowym SNem, który jest celem. Waga źródło-cel ustawiana jest na null (do późniejszej aktualizacji - patrz punkt 7), a w odwrotnym kierunku na wartość 1. Utworzona synapsa jest dodawana do kontenerów synaptycznych grafu AGDS, neuronu VN oraz nowego neuronu SN - zwracana jest wartość true, czyli operacja przebiegła poprawnie. Jeśli neuron VN jest neuronem VN.NULL, to zwracany jest fałsz, czyli niepowodzenie.

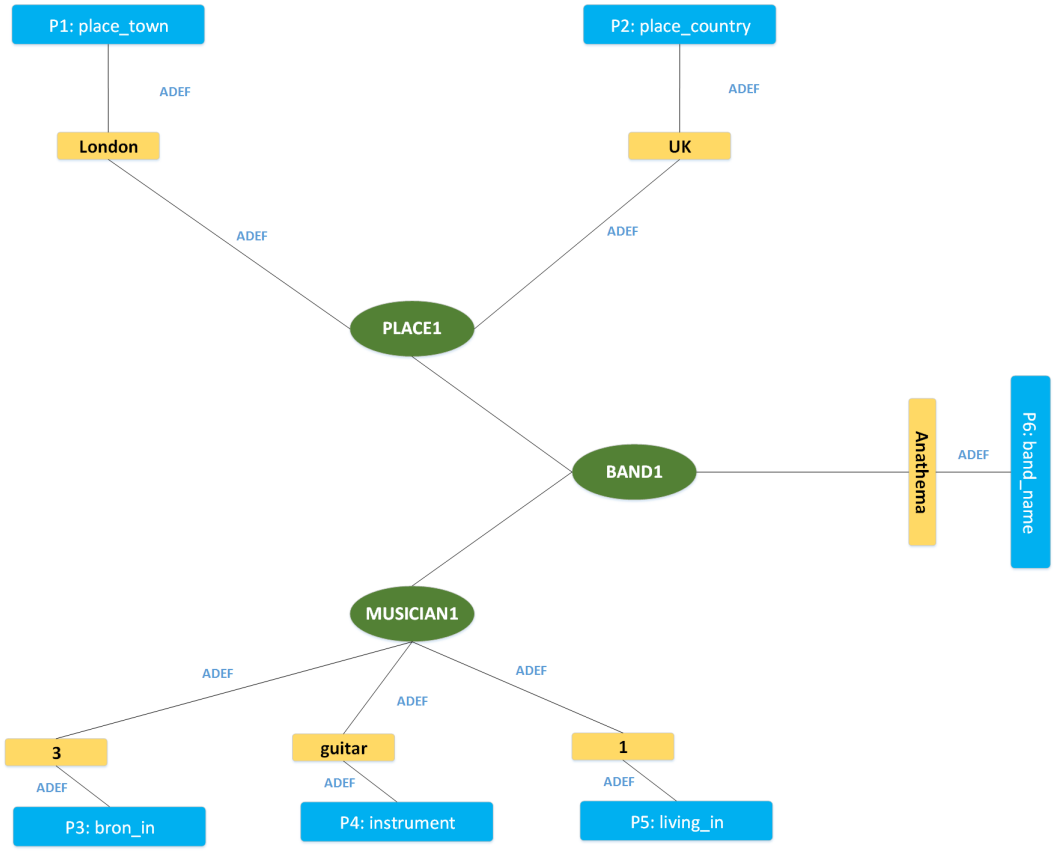
7. Uaktualnianie wag synaptycznych połączeń definiujących nowy SN

Po skończonym procesie tworzenia SNa należy zaktualizować wagi połączeń synaptycznych definiujących nowy SN. Wyszukiwane są te połączenia i ich waga, pierwotnie ustawiona na null, ustawiana jest na wartość ilorazu 3.3.

$$\frac{1}{\texttt{countNeuronsConnectedToNewSN}} \tag{3.3}$$



Rysunek 3.2: Przykładowe encje transformowane do postaci AGDS jednoznacznie.



Rysunek 3.3: Encje z rys. 3.2 po konwersji do postaci AGDS.

Wyżej zaprezentowany algorytm prezentuje sposób transformacji relacyjnej bazy danych do postaci grafu AGDS. Jednakże czasami może pojawić się wieloznaczność podczas definiowania neuronów SN przez inne neurony SN ze względu na występowanie wielokrotnych zależności typu FK, co zostanie dokładniej przeanalizowane w kolejnej sekcji 3.3.3.

3.3.3. Redukcja wieloznaczności

Gdy wykryta została wieloznaczność sposób postępowania jest następujący:

1. Najpierw kontener zawierający wszystkie neurony jest przeszukiwany pod kątem istnienia tzw. neuronu „obcego", który posłużył już wcześniej do rozwiązania wieloznaczności tego samego typu. Wyszukuje się go na podstawie nazwy tabeli, nazwy kolumny odwołującej się do innej tabeli poprzez klucz obcy FK, nazwy kolumny do której się odwołuje wcześniej wzmiankowany klucz obcy FK, wartości odwołania (najczęściej wartość klucza obcego FK, czyli wartość w kolumnie PK tabeli, do której się odwołujemy).

2. Jeśli neuron „obcy" istnieje, jego property `Cardinality` jest inkrementowane, a on sam definiuje nowo utworzony SN. Analogicznie definiowałby nowo utworzony neuron SN neuron, do którego odwołuje się nowo utworzony SN, gdyby wieloznaczność nie wystąpiła.

3. Jeśli neuron „obcy" nie istnieje, to musi zostać utworzony. Otrzymuje specjalnie spreparowaną nazwę wg wzoru:

$$FSNN = RNTN + RNCN + RCN + FKV, \tag{3.4}$$

gdzie:

FSNN – ***F**oreign **S**N **N**ame*, nazwa SNa, będącego neuronem „obcym".

RNTN – ***R**eferencing **N**euron **T**able **N**ame*, nazwa tabeli neuronu SN, który będzie definiowany poprzez właśnie budowany neuron „obcy".

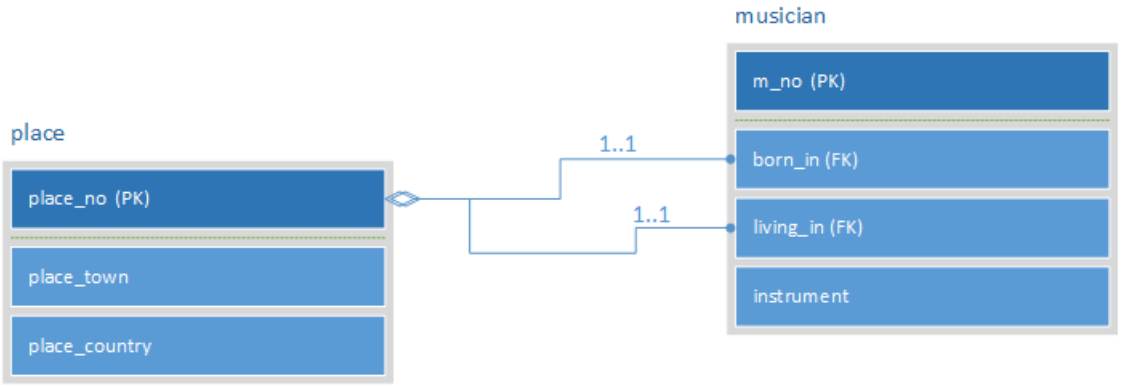
RNCN – ***R**eferencing **N**euron **C**olumn **N**ame*, nazwa kolumny z tabeli neuronu SN, który jest definiowany poprzez właśnie budowany neuron „obcy". Jedna z kolumn, dla których występuje wieloznaczność.

RCN – ***R**eferenced **C**olumn **N**ame*, nazwa kolumny z tabeli, której wiersz reprezentuje neuron, który miał definiować bieżący neuron SN, a dla którego ze względu na wystąpienie wieloznaczności budowany jest neuron „obcy".

FKV – ***F**oreign **K**ey **V**alue*, wartość w kolumnie *RCN*, równa wartości w kolumnie *RNCN* (klucz obcy FK) w bieżącej tabeli *RNTN*,

a poszczególne człony w nazwie *FSNN* są rozdzielone separatorem: „_". Następnie tworzona jest dwukierunkowa homogeniczna synapsa o wartościach wag równych 1 i typie połączenia „ADEF"

między neuronem, do którego odwołuje się nowy SN, a właśnie utworzonym „obcym" neuronem SN. Połączenie to jest dodawane do kontenera synaptycznego grafu AGDS i do kontenerów synaptycznych neuronów: neuronu SN, do którego odwołuje się nowo powstały neuron SN oraz „obcego" neuronu SN. „Obcy" neuron jest dodawany do kontenera neuronów grafu AGDS. Na koniec „obcy" neuron definiuje nowo utworzony SN, dzięki czemu unikamy wieloznaczności (rys. 3.5).



Rysunek 3.4: Przykładowe encje transformowane do postaci AGDS - występuje wieloznaczność.

3.4. Implementacja aplikacji webowej

Back-end został w całości zaimplementowany w języku C# na platformie .NET. Zarówno klasy odpowiedzialne za tzw. logikę biznesową, jak i klasy poszczególnych warstw.

Struktura warstwowa

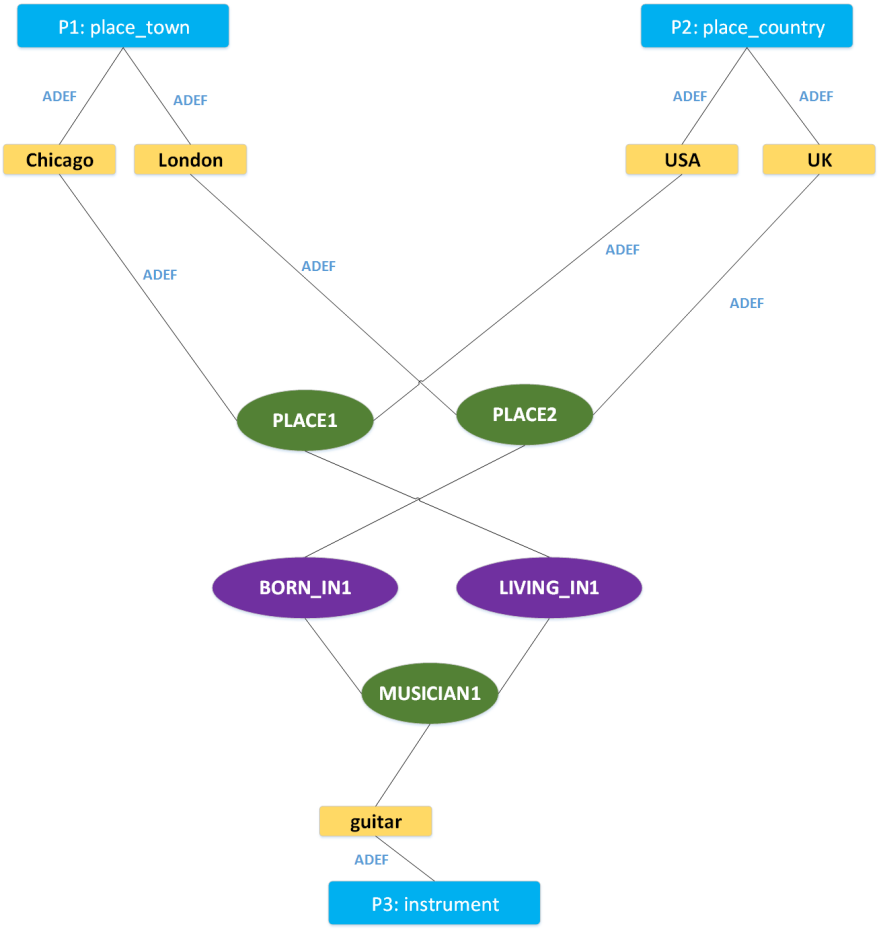
– Dolną warstwę stanowi klasa AAIData, która udostępnia interfejs do obsługi bazy danych oraz struktur grafowych AGDS.

– Kolejna warstwa to serwis, który może być podmieniony na rzeczywisty serwis WCF-owy, udostępniający usługi oferowane przez aplikację np. w chmurze obliczeniowej.

– Następna warstwa to już WebAPI, które udostępnia dane z serwisu poprzez interfejs RESTowy, dla asynchronicznych zapytań AJAXowych.

– Najwyższa warstwa to klient webowy.

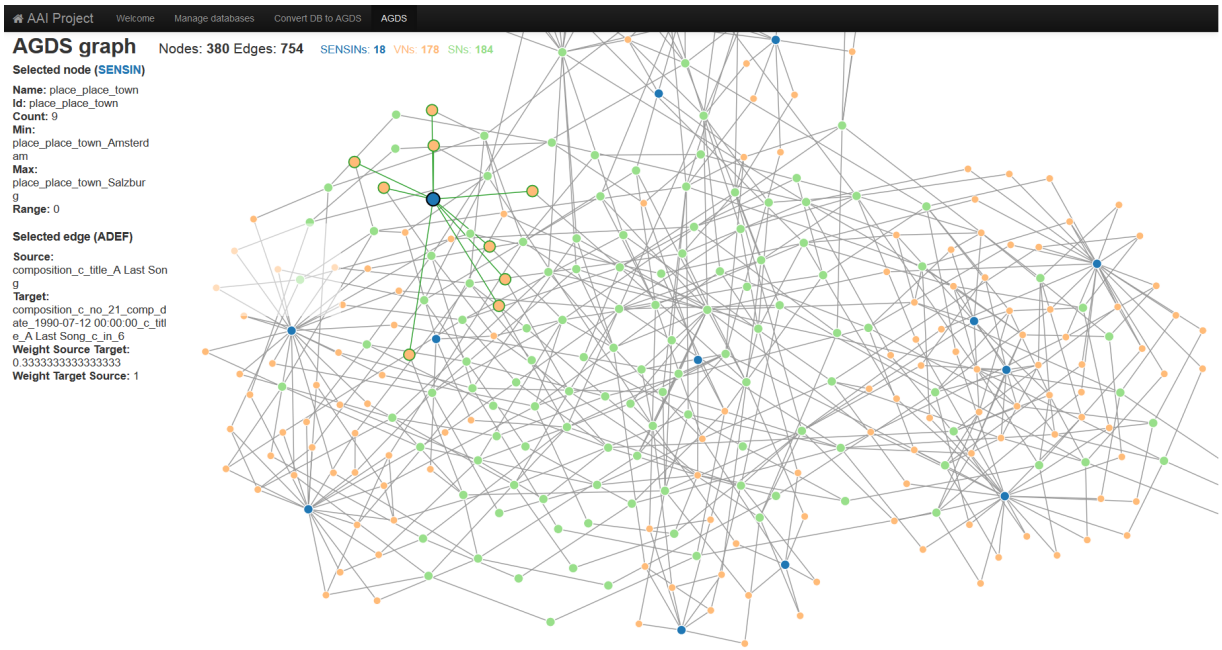
Warstwa AAIData jest odpowiedzialna za transformowanie grafu AGDS do postaci zwykłego grafu złożonego z węzłów i krawędzi - postać ta jest wykorzystywana przez bibliotekę do rysowania grafów w celu wizualizacji danych.



Rysunek 3.5: Encje z rys. 3.4 po konwersji do postaci AGDS. Widoczne fioletowe „obce” neurony SN.

3.4.1. Klient webowy

Klient webowy został zaimplementowany jako SPA, czyli (*Single Page Application*). Do realizacji tej architektury wykorzystany został framework *Durandal.js*, który pozwala na rozwijanie aplikacje wedle wzorca MVVM (*Model View ViewModel*). Framework ten wykorzystuje inną bibliotekę - *Knockout.js*. Strona internetowa została oparta o framework Bootstrap.js, natomiast do rysowania grafów wykorzystana została biblioteka *VivaGraphJS*([Kas14]).



Rysunek 3.6: Zrzut ekranu okna wizualizacji grafu AGDS dla przykładowej bazy danych.

4. Porównanie efektywności

Zarówno bazy danych jak i sztuczne systemy skojarzeniowe posiadają pewien system, mechanizm przetwarzania danych. W tym rozdziale spróbuję dokonać analizy porównawczej efektywności obu rodzajów systemów pod kątem przeprowadzanych operacji.

4.1. Przykłady wybranych operacji

Przy okazji omawiania różnych operacji, dla ustalenia uwagi analizowane będą 2 proste struktury (tabela 4.1 oraz graf 4.1), na których w uproszczeniu zostaną zwizualizowane poruszane kwestie.

Tablica 4.1: Relacja *Muzyk*.

Id	Imię	Instrument	Staż (w latach)
1	Michael Cox	flet	15
2	Eric Clapton	gitara	17
3	Darek Sojka	akordeon	25
4	Bogdan Wita	gitara	25
5	Darek Sojka	flet	17

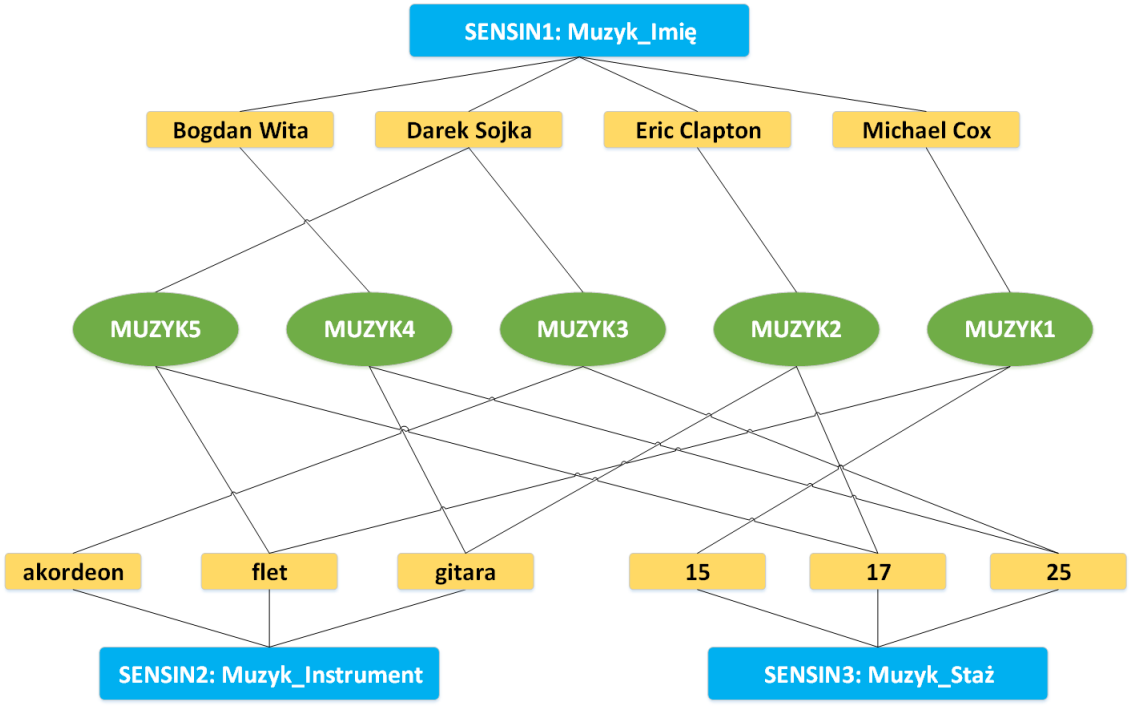
4.1.1. Znajdowanie rekordów o zadanych wartościach atrybutów

Jest to operacja polegająca na wyszukaniu danych na podstawie jednego lub kilku zadanych parametrów definiujących dany rekord. Jeśli dana jest tabela *Muzyk*, w której użytkownik chciałby wyszukać wszystkich muzyków, którzy grają na gitarze, to zapytanie w języku SQL wyglądałoby mniej więcej następująco:

Listing 4.1: Zapytanie znajdujące muzyków grających na gitarze.

```
SELECT * FROM Muzyk WHERE Instrument = 'gitara';
```

W relacyjnych bazach danych można utworzyć na danym atrybucie indeks np. w postaci tablicy haszującej, czy B-drzewa. Jeśli rozważymy operację wyszukiwania w B-drzewie, to złożoność czasowa

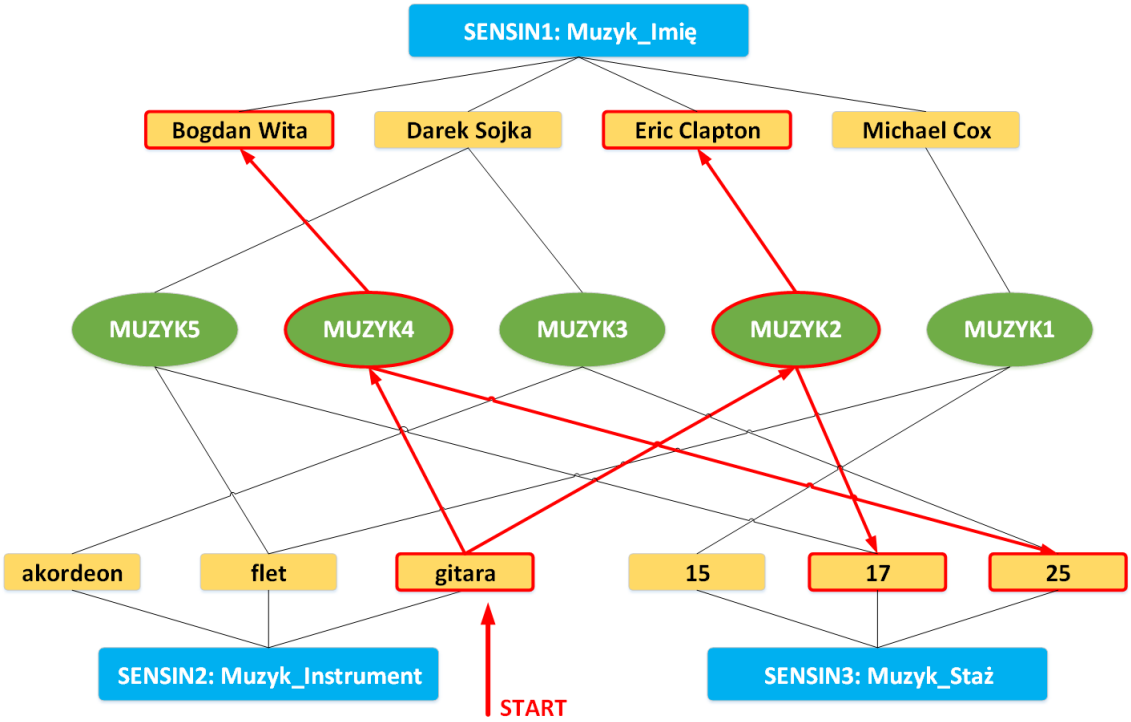


Rysunek 4.1: Grafowa struktura AGDS dla relacji Muzyk: tab. 4.1.

wynosi $O(t \log_t n)$ ([CLR01, str. 441]). Jeśli natomiast nie ma żadnych indeksów i dane nie są posortowane, to wtedy trzeba przeszukać tabelę ze złożonością liniową $O(n)$ [Hor13b, str. 110]. Dla zapytań wykorzystujących znane z algebry operatory selekcji i projekcji, czyli takich jak analizowane w tym przykładzie, przyjmuje się oszacowanie M równe 1, przy czym na potrzeby rozważań dotyczących złożoności pamięciowej M będzie oznaczało „przestrzeń potrzebną tylko na przechowywanie danych wejściowych i pośrednich wyników działań" [GMUW03, str. 295].

Gdyby tabelę przekształcić w graf AGDS, to wtedy wystarczyłoby w SENSINie *Muzyk_Instrument*, aktywować VN 'gitara' i w ten sposób można by wyszukać wszystkie neurony, połączone z tym neuronem wartości, czyli wszystkich muzyków grających na gitarze. Zarówno SENSIN, jak i znajdujący się w nim VN są dostępne w czasie stałym - bezpośrednio. Jeśli implementacja byłaby równoległa, to wtedy można by zwrócić wszystkich muzyków również w czasie stałym. W przeciwnym przypadku trzeba by zwracać ich kolejno w czasie liniowym, a następnie zwracać w czasie liniowym dane z nimi związane, by dowiedzieć się np. kim są, itd.

Co w przypadku, gdy rekord ma zostać wyszukany na podstawie wielu atrybutów na raz? Można wtedy korzystać z indeksów wielowymiarowych ([GMUW03, rozdz. 5]). W takim przypadku jednak muszą być spełnione pewne warunki, żeby dostęp był efektywny – określony musi być pierwszy atrybut tak, żeby można było za pomocą indeksu z korzenia odnaleźć kolejne podindeksy. Jeśli tak nie jest, to trzeba przeszukiwać wszystkie podindeksy, co może okazać się bardzo czasochłonne [GMUW03, str. 246]. Wg moich oszacowań pesymistyczny wariant, gdzie znany jest tylko ostatni atrybut spośród wszystkich atrybutów wchodzących w skład indeksu wielowymiarowego może wiązać się ze złożonością



Rysunek 4.2: Wyszukiwanie muzyków, którzy grają na gitarze.

czasową równą $O(t^{h-1}\log_2 t)$, gdzie h, to *wysokość* B-drzewa, a t to *minimalny stopień*, od którego zależy górne i dolne ograniczenie na ilość kluczy w węzłach.

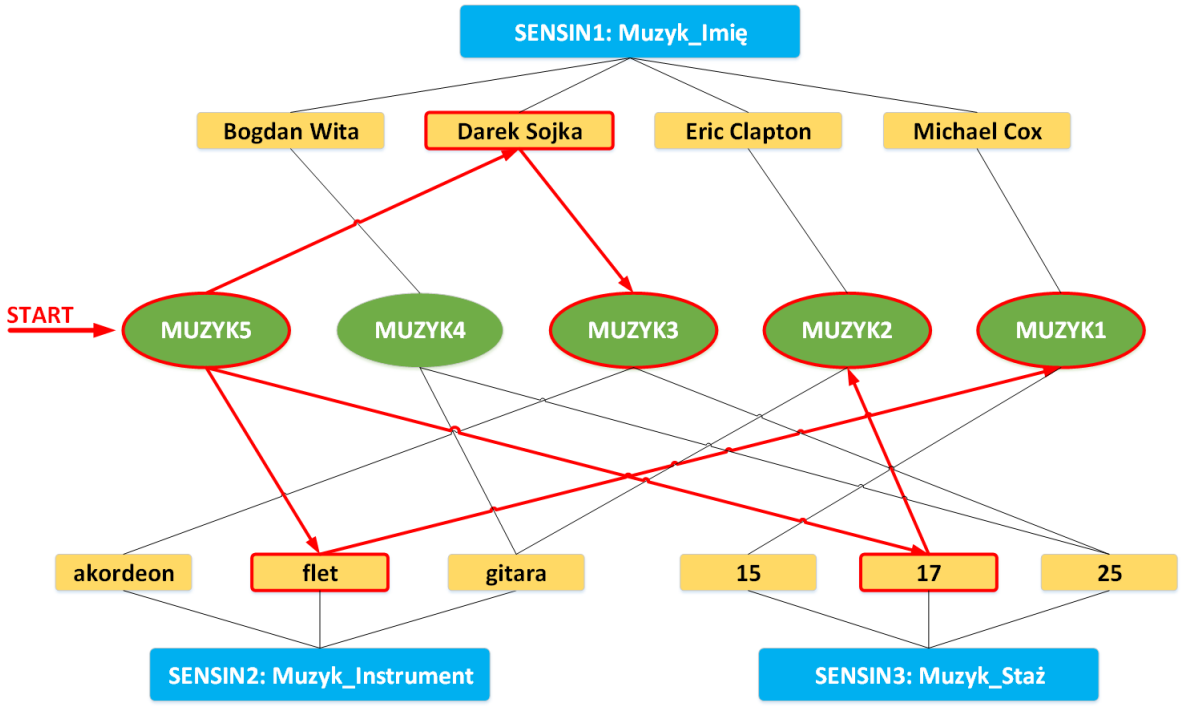
W przypadku sztucznych systemów skojarzeniowych wyszukanie odpowiednich wzorców sprowadza się do aktywowania odpowiednich neuronów wartości VN i sprawdzenia, które neurony SN zostały aktywowane. Przy implementacji równoległej wyszukanie jednej odpowiedzi/wzorca odbywa się ze stałą złożonością obliczeniową $O(1)$, a w przypadku sekwencji obiektów ze złożonością liniową [Hor13b, str. 129-130]. Na rys. 4.2 pokazano jak działa ten mechanizm w strukturach grafowych AGDS. Aktywacja oznacza po prostu odwołanie do wierzchołka, z którego rozpoczynamy przeszukiwanie grafu. Następnie, przesuwając się po krawędziach dochodzimy do wierzchołków reprezentujących muzyków. Z nich, idąc dalej po krawędziach odczytujemy wszystkie dane, jakie z konkretnymi muzykami są związane. Mechanizm ten można prześledzić na przykładzie: jako startowy wybierany jest węzeł *gitara*, który bezpośrednio definiuje muzyków, którzy grają na gitarze (*MUZYK2* i *MUZYK4*). Z węzłów reprezentujących muzyków dostępne są wszystkie dane ich dotyczące. Widzimy, że na gitarach grają: *Bogdan Wita* oraz *Eric Clapton* oraz że mają staż odpowiednio 25 i 17-letni. Na tym przykładzie zilustrowano, że dzięki połączeniom obecnym w grafie błyskawicznie można przeprowadzać wszelkie analizy i dokonywać eksploracji danych.

4.1.2. Znajdowanie podobnych rekordów

To zadanie jest podobne do poprzedniego, z tym że najpierw trzeba pobrać wartości atrybutów z rekordu i – w przypadku języka SQL – skonstruować złożone zapytanie wykorzystujące operatory logiczne, bądź matematyczne albo ich kombinacje. W tym przypadku wystarczy prosty operator logiczny:

Listing 4.2: Zapytanie znajdujące muzyków podobnych do danego.

```
SELECT * FROM Muzyk WHERE
    Instrument = 'flet' OR
    Imię = 'Darek Sojka' OR
    Staż = 17;
```



Rysunek 4.3: Wyszukiwanie muzyków, podobnych do wybranego.

Natomiast dla struktur asocjacyjnych wystarczyłoby określić próg aktywacji, który określałby satysfakcjonujący stopień podobieństwa i analizować kolejne aktywacje różnych neuronów wzorców. Na rys. 4.3 widać, że po wybraniu interesującego nas wierzchołka reprezentującego muzyka *MUZYK5*, idąc po krawędziach do neuronów VN (czyli zwykłych wirzchołków w grafie) można znaleźć 3 muzyków, którzy są podobni do pierwotnie wybranego. Pierwszy muzyk jest podobny w tym, że gra na flecie (*Michael Cox*, kolejny, że ma taki sam staż (17 lat *Eric Clapton* oraz że ma tak samo na imię – oznacza tego samego muzyka, który gra na innym instrumencie (na *flecie*). W tym przypadku widać również, że dzięki usunięciu redundancji w asocjacyjnych grafach AGDS w wydajny sposób można dokonywać wnioskowania i ciekawych analiz. Gdyby ustawić wymagany stopień zgodności na np. większy niż 50%, to wtedy żaden muzyk nie zostałby określony jako podobny do danego, ponieważ na tym konkretnym

przykładzie zgodność wynosi 33.33%, ponieważ każdy muzyk jest definiowany przez 3 neurny wartości VN - węzły grafu AGDS.

4.1.3. Znajdowanie minimum i maksimum dla zadanego parametru

Operacja polega na znalezieniu minimum lub maksimum dla zadanego parametru w tablicy. Odpowiednie query w jęz. SQL mogłoby mieć następującą postać:

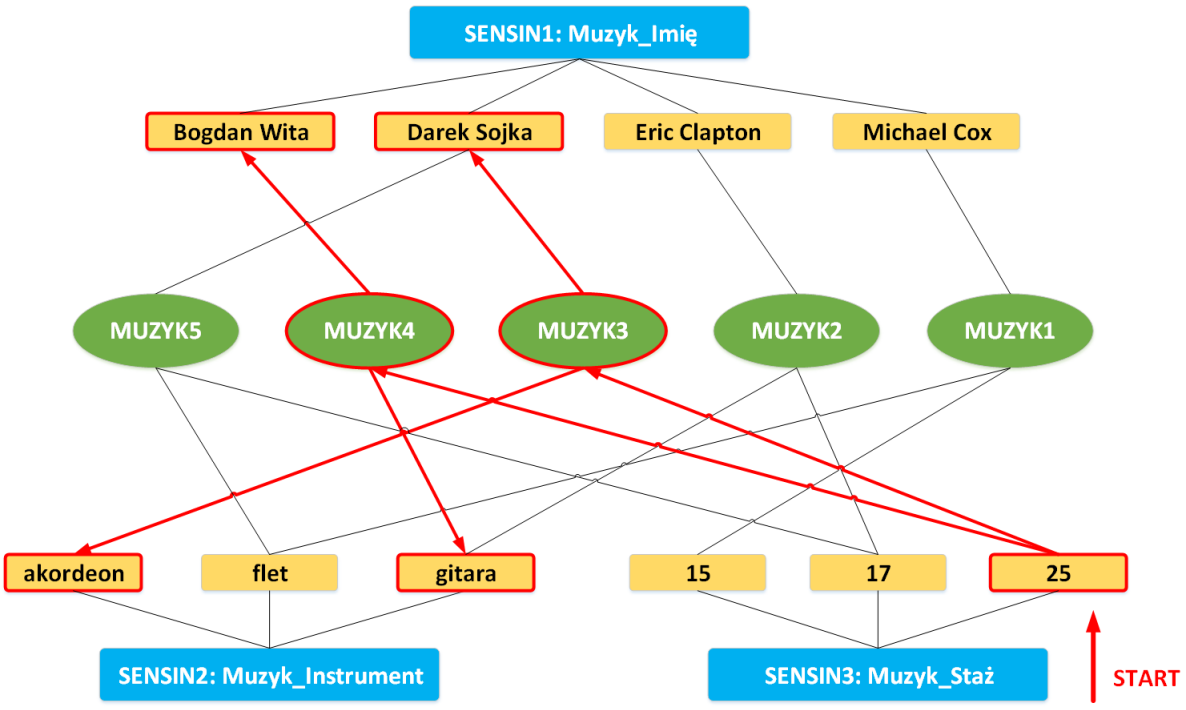
Listing 4.3: Zapytanie znajdujące muzyków z największym stażem.

```
SELECT MAX(Staż) FROM Muzyk;
```

Problem polega na tym, że nie wiadomo tak naprawdę, czy tylko jeden muzyk ma tak długi staż. Gdyby użytkownik chciał uzyskać dane dotyczące muzyków z najdłuższym stażem, trzeba by nieco zmodyfikować zapytanie:

Listing 4.4: Zapytanie znajdujące dane dotyczące muzyków, którzy mają najdłuższy staż.

```
SELECT * FROM Muzyk WHERE Staż = (SELECT MAX(Staż) FROM Muzyk);
```



Rysunek 4.4: Zidentyfikowanie muzyków o najdłuższym stażu.

Operacje znajdowania minimum i maksimum wiążą się z agregacją, dla której oszacowanie M wynosi B, które definiuje się jako „liczbę krotek potrzebnych do przechowania wszystkich bloków relacji” [GMUW03, str. 295].

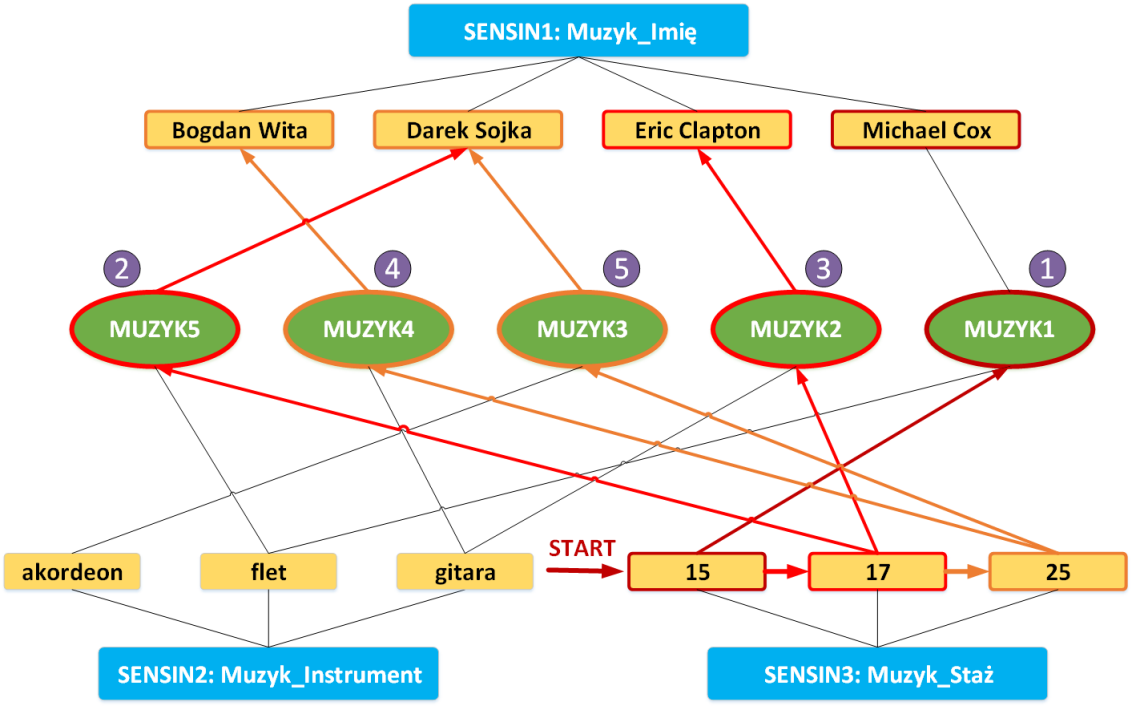
Dla asocjacyjnych sztucznych systemów skojarzeniowych wystarczyłoby aktywować ostatni VN w SENSINie *Muzyk_Staż* i wszystkie połączenia pomiędzy VNem, a neuronami SN reprezentującymi mu-

zyków, dałyby właściwą odpowiedź. Następnie należałoby tylko w miarę potrzeb „sięgnąć dalej" i pobrać dane, z którymi połączone są neurony wzorców muzyków. Schematycznie ilustruje to rys. 4.4.

Analiza grafu poprawnie wykazała, że jest dwóch muzyków z najdłuższym stażem, którzy nazywają się *Bogda Wita* oraz *Darek Sojka*. Grają 25 lat na *gitarze* oraz *akordeonie*. Muzyk *Darek Sojka* gra również na *flecie*, ale tylko 17 lat, więc ta wartość nie zostaje zwrócona. Analogicznie analizę można przeprowadzić dla wartości minimalnej.

4.1.4. Sortowanie względem dowolnej ilości parametrów

W bazach danych, które wykorzystują klasyczne algorytmy sortujące najczęściej złożoność obliczeniowa sortowania rekordów w tablicy sprowadza się do złożoności liniowo-logarytmicznej $O(n \log n)$. Aby sortować względem innej kombinacji kluczy, należy dodać kolejny indeks, co wiąże się z dodatkowym narzutem pamięciowym rzędu $O(n)$. Jeśli uwzględni się fakt, że w sztucznych systemach skojarzeniowych wszystkie wartości są zawsze posortowane dla wszystkich SENSINów, to zwrócenie ciągu neuronów reprezentujących rekordy w tabeli w posortowanym porządku względem wszystkich parametrów nie stanowi problemu i dla równoległej implementacji odbywa się w czasie liniowym $O(n)$!



Rysunek 4.5: Zwrócenie posortowanego ciągu muzyków względem wielu parametrów: Stażu i Imienia rosnąco.

W przykładzie widocznym na rys. 4.5 widać, jak przydatne jest wstępne posortowanie wszystkich możliwych VNów we wszystkich SENSINach. Zwrócenie posortowanego ciągu względem *Stażu* oraz *Imienia* sprowadziło się do iterowania najpierw po *Stażu*, zaczynając od najniższej wartości równej *15*, i zwracaniu kolejnych muzyków. W momencie, gdy wielu muzyków miało ten sam *Staż* (np. *17* oraz *25*),

trzeba było iterować po *Imionach*, by wyznaczyć właściwą kolejność. W ten sposób górne oszacowanie uzyskania posortowanego względem p parametrów ciągu obiektów w strukturze AGDS można ustalić na $O(n \cdot p \cdot q)$, gdzie q to maksymalna ilość różnych wartości VNów w ramach SENSINów, a n to liczba sortowanych rekordów.

4.1.5. Podsumowanie

W tabeli 4.1 występuje redundancja – dane się powtarzają. Natomiast w grafie AGDS wszystkie wartości są uporządkowane, nie występuje redundancja, a węzły są ze sobą powiązane, dzięki czemu mimo całkowitej dekompozycji i ich rozdrobnieniu, możliwa jest ich bardzo wydajna i efektywna analiza. Można stąd wysnuć wniosek, że w strukturach AGDS nie trzeba iść na żadne kompromisy – dekompozycja i brak redundancji idą w parze z wydajnością i efektywnością w przeciwieństwie do relacyjnych baz danych, dla których projektanci muszą poszukiwać „złoty środek”.

5. Podsumowanie

5.1. Wnioski

W wyniku zrealizowanych prac wysnuto następujące wnioski:

– Bazy relacyjne są obecnie najpopularniejszymi systemami przechowywania i operowania na danych - bezpiecznie przechowują dane w sposób pasywny.

– Ciekawą alternatywą dla relacyjnych systemów bazodanowych w przypadku eksploracji danych i wykonywania analizy danych są sztuczne systemy skojarzeniowe bazujące na strukturze AGDS, które udostępniają wiele ciekawych i wartościowych informacji takich jak korelacje, podobieństwa między obiektami, ich kolejność względem zadanych parametrów, maksima, minima, itp.

– Równoległość obliczeń odgrywa ogromną rolę w każdym systemie - szczególnie w systemach wzorowanych na układach biologicznych. Dzięki zrównolegleniu obliczeń można zaobserwować znaczny wzrost efektywności.

5.2. Zrealizowane cele

W ramach zrealizowanej pracy magisterskiej udało się osiągnąć następujące cele:

– Zaprojektowano i zaimplementowano algorytm transformacji dowolnej bazy danych (zgodnie z przyjętymi założeniami i ograniczeniami) do postaci grafu AGDS.

– Dokonano analizy efektywności działania relacyjnych baz danych oraz grafowych struktur AGDS. Ponadto:

– Dokonano implementacji webowej aplikacji, dzięki której można było zwizualizować otrzymane wyniki transformacji relacyjnych baz danych do postaci grafów AGDS.

– Zdobyto fascynującą i ciekawą wiedzę z zakresu asocjacyjnej sztucznej inteligencji.

– Poszerzono wiedzę i umiejętności dotyczące inżynierii oprogramowania, oraz technologii webowych.

– Przeprowadzono ciekawe eksperymenty naukowe.

5.3. Perspektywy

Niniejsza praca dyplomowa jest zaledwie przyczynkiem do dużo ciekawszych i głębszych badań nad asocjacyjną sztuczną inteligencją. Niemniej jednak można zaproponować szereg rozszerzeń do zrealizowanych i zaimplementowanych w tej pracy rozwiązań i zaproponowanie nowych celów, czy też obszarów badań:

- można by wzbogacić warstwę baz danych o obsługę nowych silników bazodanowych typu: PostgreSQL czy MySQL.
- można zaimplementować algorytm transformacji baz danych do postaci AGDS w wersji zrównoleglonej, wykorzystującej wiele rdzeni procesora.
- można ulepszyć mechanizm wizualizacji grafów AGDS i poprawić interfejs użytkownika.
- można wprowadzić również wizualizację 3D.
- kolejnym krokiem mogłaby być transformacja grafu AGDS do postaci AANG.

Planuję w niedalekiej przyszłości dokonać powyższych usprawnień i rozszerzeń w realizowanej pracy, a także w miarę możliwości systematycznie poszerzać wiedzę dotyczącą sztucznych systemów skojarzeniowych i asocjacyjnej sztucznej inteligencji, która jest niezwykle fascynującą dziedziną nauki.

Bibliografia

[CLR01] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Wprowadzenie do algorytmów.* Wydawnictwa Naukowo-Techniczne, Warszawa, 2001.

[EB05] R. Elmasri and Navathe S. B. *Wprowadzenie do systemów baz danych.* Wydawnictwa HELION, Gliwice, 2005.

[GMUW03] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Implementacja systemów baz danych.* Wydawnictwa Naukowo-Techniczne, Warszawa, 2003.

[GMUW06] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Systemy baz danych. Pełny wykład.* Wydawnictwa Naukowo-Techniczne, Warszawa, 2006.

[Hor12] A. Horzyk. Information freedom and associative artificial intelligence. pages 81–89, 2012.

[Hor13a] A. Horzyk. How does human-like knowledge come into being in artificial associative systems. In *Proc. of the 8-th International Conference on Knowledge, Information and Creativity Support Systems*, pages 189–200, Kraków, Poland, 2013.

[Hor13b] A. Horzyk. *Sztuczne systemy skojarzeniowe i asocjacyjna sztuczna inteligencja.* Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2013.

[Hor14] A. Horzyk. How does generalization and creativity come into being in neural associative systems and how does it form human-like knowledge? *Neurocomputing*, 2014.

[Kas14] A. Kashcha. *VivaGraphJS - graph drawing library*, 2014. `https://github.com/anvaka/VivaGraphJS`.

[MM08] R. C. Martin and M. Martin. *Agile. Programowanie zwinne. Zasady, wzorce i praktyki zwinnego wytwarzania oprogramowania w C#.* Helion, Gliwice, 2008.

[Tad93] R. Tadeusiewicz. *Sieci neuronowe.* Akademicka Oficyna Wydawnicza, Kraków, 1993.

[TGBL07] R. Tadeusiewicz, T. Gąciarz, B. Borowik, and B. Leper. *Odkrywanie właściwości sieci neuronowych przy użyciu programów w języku C#.* Polska Akademia Umiejętności, Kraków, 2007.

[uBJ96] J. Żurada, M. Barski, and W. Jędruch. *Sztuczne sieci neuronowe. Podstawy teorii i zastosowania.* Wydawnictwo Naukowe PWN, Warszawa, 1996.

[UW01] J.D. Ullman and J. Widom. *Podstawowy wykład z systemów baz danych.* Wydawnictwa Naukowo-Techniczne, Warszawa, 2001.