Politechnika Warszawska

Wydział Matematyki i Nauk Informacyjnych

Andrzej Wąsowski

andy@prioris.mini.pw.edu.pl

Zdalna generacja sieci bayesowskich z baz danych

Praca dyplomowa pod kierunkiem

dr hab. inż. Mieczysława A. Kłopotka

Warszawa, sierpień 2000

Spis treści

Rozdział 1

Wstęp

Celem projektu jest zaimplementowanie systemu uczącego sieci Bayesa. Wykonany system umożliwia porównanie różnych metod uczenia dla tych samych danych wejściowych. Wykorzystanie nowoczesnych technologii internetowych gwarantuje dostępność oprogramowania dla szerokiej grupy użytkowników korzystających z różnych platform sprzętowych.

Automatyczne i półautomatyczne wnioskowanie z wykorzystaniem baz wiedzy jest dziś jednym z najistotniejszych zastosowań sztucznej inteligencji. Jego znaczenie rośnie wraz z rozwojem kręgu użytkowników systemów eksperckich, czyli różnego rodzaju narzędzi wspomagających podejmowanie decyzji. Analiza danych mikro i makroekonomicznych, testowanie wiarygodności kredytowej, diagnostyka chorób, czy para-inteligentny system pomocy z usuwaniem problemów przy drukowaniu w systemie Microsoft Windows[8] to tylko niektóre z licznych zastosowań[14, 9, 7] takich systemów, niejednokrotnie opartych o zasady *analizy bayesowskiej*.

Sieci przyczynowo-skutkowe, zwane też *sieciami Bayesa*, należą do często stosowanych sposobów reprezentacji w bazach wiedzy systemów eksperckich. Sieć taka reprezentuje układ zależności między różnymi parametrami w postaci grafu skierowanego. W niniejszej pracy rozważa się problem zdalnego generowania bazy wiedzy o takiej strukturze.

Generację sieci nazywa się często *uczeniem* lub odtwarzaniem. Wyróżnia się tu sposoby uczenia z wykorzystaniem eksperta oraz tzw. uczenie bez nadzoru i bez wiedzy wstępnej. Wobec rosnącej ilości zadań, do których rozwiązywania stosuje się analizę zależności statystycznych, niezbędne stało się

poszukiwanie coraz skuteczniejszych sposobów uczenia bez nadzoru. Zatrudnianie ekspertów jest bowiem zwykle o wiele droższe niż skorzystanie z seryjnie sprzedawanego samodzielnego programu uczącego.

Dodatkowym atutem uczenia całkowicie zautomatyzowanego jest możliwość budowania baz wiedzy dla znacznie większej ilości atrybutów niż można to uczynić przy pomocy o wiele wolniejszego eksperta. Wiele badań nad algorytmami uczącymi koncentruje się właśnie wobec problemu wydajności. Szybkość stała się też w znacznym stopniu problemem niniejszej pracy, w ramach której wykonano implementację konkretnego systemu uczącego i przeprowadzono badania jego wydajności.

Dyskurs ograniczony został do analizy zależności zmiennych dyskretnych. Nie omawia się tu szczegółowo problemów dyskretyzacji zmiennych ciągłych, stosowania do uczenia niepełnych prób, poszukiwania zmiennych ukrytych (za wyjątkiem algorytmu FCI). Nie podaje się też szczegółowo algorytmów wnioskowania za pomocą sieci Bayesa. Informacje na te i pokrewne tematy można jednak znaleźć w bogatej literaturze. Dobrym punktem startowym są z pewnością przekrojowe pozycje z dziedziny, zwłaszcza [14, 9, 7] zawierające podstawowe informacje na temat wspomnianych problemów oraz obszerne wykazy literatury szczegółowej.

Zrealizowany w części praktycznej system zdalnej generacji *RENEG* intensywnie korzysta z technologii internetowych. Nikt nie ma już dziś wątpliwości, że Internet stał się motorem rozwoju nie tylko informatyki, ale i całej gospodarki. Rozwój sieci globalnej przerósł wszelkie oczekiwania, a łatwa dostępność do Internetu w wielu regionach sprawiła, że stał się on najtańszym i najskuteczniejszym medium dystrybucji produktów o charakterze elektronicznym (w tym oprogramowania).

Aplikacje, dostępne zdalnie za pomocą narzędzi wizualnych WWW, zyskują szerokie grono użytkowników, stawiając jednocześnie umiarkowane wymagania sprzętowo-systemowe. Przeglądarki WWW są obecnie dostępne dla wszystkich liczących się systemów operacyjnych, a zastosowanie technologii *Common Gateway Interface* (CGI), Java i JavaSript czyni oprogramowanie internetowe przenośnym z punktu widzenia klienta. CGI dokonuje tego, umożliwiając przenoszenie logiki obliczeniowej aplikacji na serwer. Jest to zwykle komputer o dużej wydajności, co przynosi dodatkowe korzyści. Java i JavaScript stosują nowoczesne technologie interpretacji meta-kodu i kodu, aby udostępnić stronie użytkownika niezbędną funkcjonalność (zwłaszcza w

zakresie interfejsu graficznego).

Rozwój oprogramowania internetowego doprowadził do zmiany jego kategoryzacji pojęciowej. Dziś tego rodzaje aplikacje nie są już postrzegane jako produkty. Znalazły się raczej w sferze usług – tyle, że świadczonych na odległość. Takie właśnie usługi z zakresu generacji sieci Bayesa realizuje zaimplementowany system.

Niniejszy tekst podzielono na pięć rozdziałów. W rozdziale drugim przedstawiono wybór materiału teoretycznego z zakresu uczenia sieci, który wykorzystano w rozdziale trzecim, gdzie szczegółowo opisano implementację zastosowanych metod uczenia. W rozdziale czwartym zawarto opis wykonanych doświadczeń, omawiając wydajność i efektywność zaimplementowanych algorytmów. Wreszcie końcowy rozdział piąty reasumuje wykonaną pracę i formułuje wnioski.

Rozdział 2

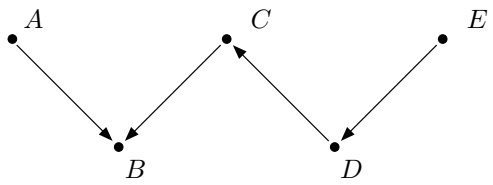
Wprowadzenie teoretyczne

Rozdział zawiera materiał, który stał się podstawą teoretyczną dla zaimplementowanego systemu opisanego w rozdziale trzecim. Wprowadza się tutaj pojęcia podstawowe i ich własności oraz określa się jednolity system oznaczeń. Następnie wyjaśnia się zasady funkcjonowania poszczególnych algorytmów.

Spośród wielu różnych sposobów reprezentacji wiedzy (bazy reguł, predykatów, drzewa wnioskowania, sieci Markova) przedyskutowano tu sieci Bayesa zwane też sieciami przyczynowo-skutkowymi. Ze względu na ogracznioną objętość pracy przedstawiono tylko najpotrzebniejsze informacje. Dowody twierdzeń i wyprowadzenia algorytmów można znaleźć w podawanej każdorazowo literaturze. Przyjęto zasadę „mimimum", że opis teoretyczny wraz z uwagami praktycznymi zawartymi w rozdziale trzecim powinien pozostać wystarczająco dokładny dla informatyka z podstawową wiedzą z zakresu prawdopodobieństwa dyskretnego i statystyki do wykonania implementacji analogicznego systemu.

2.1 Semantyka sieci Bayesa

Definicja 1 *Siecią Bayesa nazywamy skierowany graf acykliczny o wierzchołkach reprezentujących zmienne losowe i łukach określających zależności. Istnienie łuku pomiędzy dwoma wierzchołkami oznacza istnienie bezpośredniej zależności przyczynowo skutkowej pomiędzy odpowiadającymi im zmiennymi.*



Rysunek 2.1: Przykładowa sieć Bayesa

Siła tej zależności określona jest przez tablice prawdopodobieństw warunkowych.

Orientacja łuków jest niezbędna do wyrażenia tzw. zależności nieprzechodnich.

Na rysunku 2.1 przedstawiono prostą sieć przyczynowo-skutkową. Wymieńmy kilka przykładowych własności, które można odczytać z postaci grafu. Zmienne A i B są zależne bezpośrednio od siebie. Podobnie para B i C jest wzajemnie zależna. Zatem wierzchołek B reprezentuje zmienną, która zależy jednocześnie od A i C. Zmienne A i C pozostają jednak brzegowo niezależne dopóki nie zostanie ustalona wartość B. Mówimy, że nie są one d-separowane przez B lub, że są d-połączone.[1] Jednocześnie zmienne E i C są zależne pośrednio. Mówimy, że zmienna D d-separuje C i E. Podobnie zmienne C i D d-separują parę B, E.

Powyższy przykład pokazuje, że mimo, iż własność zależności jest przemienna (niezorientowana), to orientacja krawędzi wnosi istotną informację o rozkładzie, która w przeciwnym razie zostałaby utracona. Jest to właśnie wcześniej wspomniana przechodniość zależności.

Przyjrzyjmy się innej podstawowej własności sieci. Jeżeli przez $\Pi(X_i)$ oznaczymy zbiór rodziców danego wierzchołka w grafie, to rozkład prawdopodobieństwa zmiennych danej sieci opisuje się równaniem:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\Pi(X_i)) \tag{2.1}$$

W naszym przykładzie:

[1]Formalną definicję d-separacji podano w dalszym ciągu tego rozdziału, w paragrafie 2.3.3.

$$P(A, B, C, D, E) = P(A)P(B|A, C)P(C|D)P(D|E)P(E) \qquad (2.2)$$

Za pomocą tego równania można więc określić prawdopodobieństwo wystąpienia określonego wartościowania wszystkich zmiennych, znając jedynie lokalne prawdopodobieństwa warunkowe. Określając wartości tzw. przyczyn podstawowych, czyli zmiennych, które w grafie nie są przez nic poprzedzane (nie mają rodziców), można określić wartości oczekiwane innych atrybutów. Wszystkie pozostałe zmienne zależą bowiem pośrednio lub bezpośrednio od tego zbioru.

W swej istocie sieci przyczynowo-skutkowe przenoszą informacje o rozkładzie prawdopodobieństwa. W praktyce jednak wykorzystywane są zwykle nie tyle do opisywania dokładnych rozkładów, ile do ekstrakcji informacji o rozkładach nieznanych, o których wiadomości możemy czerpać tylko z dostępnych prób wynikowych z eksperymentów statystycznych. Ponieważ próby są zwykle duże a ukryte w nich informacje nie są obliczeniowo łatwe do wykorzystania, proces uczenia sieci może być postrzegany jako statystyczna kompresja informacji o rozkładzie do zwartej i jednocześnie bardziej przydatnej do wnioskowania bayesowskiego postaci.

Rozwinięto już badania (między innymi [17, 1, 2]) pozwalające na przekształcanie sieci przyczynowo-skutkowych tak, by możliwe było wnioskowanie z innych elementów niż te, które nazwano wyżej przyczynami podstawowymi. Obejmują one sposoby uczenia pod pewnymi warunkami (by sieć była odpowiednio tania obliczeniowo), sposoby odwracania łuków, czy poszukiwania zbiorów d-separujących o najmniejszej wadze (np. złożonych z atrybutów, których wartości najtaniej się uzyskuje). Elastyczność sieci Bayesa jako bazy wiedzy ciągle wzrasta, zwiększając jednocześnie ilość potencjalnych zastosowań.

2.2 Terminologia, oznaczenia i założenia

W pracy tej nie rozważa się sposobów wnioskowania i transformacji sieci, a jedynie zasady generacji jej podstawowej struktury. W kolejnych rozdziałach opisane są algorytmy rozwiązujące to zadanie. Aby ułatwić zrozumienie procedur pochodzących od różnych autorów w opisach przyjęto zaprezentowany tu jednolity system oznaczeń.

Niech $\mathcal{D}$ oznacza próbę statystyczną o rozkładzie dyskretnym (lub w ujęciu informatycznym prostokątną tablicę danych dyskretnych). Niech $\mathbf{V} = \{X_1, ..., X_n\}$ to zmienne z próby $\mathcal{D}$ (atrybuty z tabeli $\mathcal{D}$). Każda zmienna X_i przyjmuje r_i różnych wartościowań w $\mathcal{D}$. Liczba wszystkich przypadków w próbie (wierszy w tabeli) wynosi m. Wreszcie niech P oznacza rozkład prawdopodobieństwa zmiennych $X_1, ..., X_n$, z którego pochodzi próba $\mathcal{D}$.

Podajmy teraz teraz formalną definicję sieci Bayesa (nieco mniej formalnie zapisaliśmy to samo w def. 1).

Definicja 2 *Jeżeli B_S jest acyklicznym grafem skierowanym (DAG) opisującym strukturę sieci Bayesa, to przez $\Pi(X)$ rozumiemy zbiór wszystkich rodziców zmiennej X w B_S. Jeśli B_P jest zbiorem tablic prawdopodobieństw warunkowych takich, że zawierają prawdopodobieństwa zdarzeń $P(X_i|\Pi(X_i))$ to parę $B = (B_S, B_P)$ nazywamy siecią Bayesa (siecią przyczynowo-skutkową).*

Wszystkie zaimplementowane tu algorytmy zakładają, że próba $\mathcal{D}$ jest dyskretna, a zdarzenia w próbie występują niezależnie i podczas zbierania danych zależności między atrybutami były stałe. Informacja o zdarzeniach (przypadkach) w próbie jest pełna, tzn. w tablicy nie ma brakujących wartości (pustych komórek).

Wykonywane uczenie sieci jest uczeniem bez nadzoru i bez wstępnej wiedzy eksperckiej, a co za tym idzie zakłada się, że na wstępie (bez znajomości próby $\mathcal{D}$) wszystkie legalne struktury sieci są jednakowo prawdopodobne, a przy ustalonej strukturze B_S wszystkie możliwe poprawne zbiory tablic prawdopodobieństw warunkowych B_P są jednakowo prawdopodobne.

Podane poniżej algorytmy wymagają próby statystycznej $\mathcal{D}$ jako parametru wejściowego. Oczekują też różnych informacji dodatkowych (zależnych od algorytmu), wykorzystywanych w celu ustalenia warunku stopu lub obniżenia złożoności zadania, które w swej istocie jest NP-trudne [4].

2.3 Algorytmy

2.3.1 Algorytm Chow-Liu

Algorytm Chow-Liu [5] należy do klasycznych algorytmów odtwarzających kształt zależności w próbie. Algorytm ten, podany jeszcze w 1968 roku, nie

buduje sieci przyczynowo skutkowej a jedynie niezorientowane drzewo zależności. Jeżeli sieć Bayesa danego rozkładu ma postać drzewa, to algorytm Chow-Liu powinien poprawnie odtworzyć jego kształt.

Formalnie przyjmuje się, że sieć bayesowska ma strukturę drzewiastą, gdy dla każdej pary zmiennych X_{i_1}, X_{i_2} i wszystkich wartościowań zmiennych $x_1, x_2, \ldots, x_n$ w próbie $\mathcal{D}$ spełniony jest poniższy warunek:

$$\begin{aligned}
&(\forall\ i_1, i_2 = 1 \ldots n, i_1 \neq i_2)(\forall\ x_1, \ldots, x_n) \\
&P(X_{i_1} = x_{i_1} | X_1 = x_1, \ldots, X_{i_1-1} = x_{i_1-1}, X_{i_1-1} = x_{i_1+1}) = \\
&\quad = P(X_{i_1} = x_{i_1} | X_{i_2} = x_{i_2}) \vee \\
&P(X_{i_2} = x_{x_2} | X_1 = x_1, \ldots, X_{i_2-1} = x_{i_2-1}, X_{i_2+1} = x_{i_2+1}) = \\
&\quad = P(X_{i_2} = x_{i_2} | X_{i_1} = x_{i_1})
\end{aligned} \tag{2.3}$$

Algorytm Chow-Liu nie sprawdza, czy rozkład zmiennych zadanej próby $\mathcal{D}$ spełnia powyższy warunek. Jeśli struktura zależności między zmiennymi w próbie nie jest drzewem, to algorytm znajduje najlepszą strukturę drzewiastą, która opisuje postać zależności. Należy jednak pamiętać, że w skrajnie niekorzystnych wypadkach informacja zwrócona przez algorytm może być zupełnie bezwartościowa.

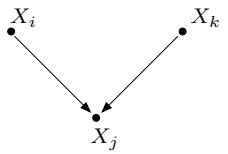
Algorytm Chow-Liu

1. Niech G oznacza graf pełny o zbiorze wierzchołków tożsamym ze zbiorem atrybutów próby $\mathcal{D}$. Wówczas niech T będzie nieskierowaną strukturą drzewiastą uzyskaną w wyniku zastosowania dowolnego algorytmu znajdowania minimalnego drzewa rozpinającego (MST) w grafie G z funkcją wagową określającą stopień zależności między zmiennymi.

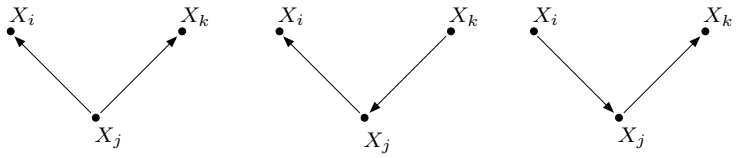
2. W drzewie T należy obrać kierunki w sposób dowolny, uzyskując wynikową strukturę sieci Bayesa B_S.

W pierwszym kroku zaproponowano wykorzystanie *odległości Kullback-Leiblera* jako funkcji wagowej:

$$DEP(X_i, X_j) = \sum_{x_i, x_j} P(x_i, x_j) \log \frac{P(x_i, x_j)}{P(x_i)P(x_j),} \tag{2.4}$$



Rysunek 2.2: Dwu rodziców: $DEP(X_i, X_k) = 0 \wedge DEP(X_i, X_k|X_j) > 0$



Rysunek 2.3: X_j d-separuje X_i, X_k: $DEP(X_i, X_k) > 0 \wedge DEP(X_i, X_k|X_j) = 0$

gdzie podobnie jak w warunku (2.3) małe litery x_i, x_j oznaczają wartościowania zmiennych X_i, X_j a sumowanie przebiega po wszystkich wartościowaniach w próbie $\mathcal{D}$.

2.3.2 Algorytm Pearla

Pearl[14] zaproponował modyfikację algorytmu Chow-Liu. Zauważył, że zamiast przyjmować dowolną orientację kierunków w drugim kroku, można, przy wykorzystaniu funkcji $DEP(\cdot,\cdot)$, wykryć węzły, w których strzałki spotykają się głowami (czyli węzły nie d-separujące swoich rodziców). Należy wykorzystać do tego funkcję warunkową:

$$DEP(X_i, X_j|X_k) = \sum_{x_i,x_j,x_k} P(x_i, x_j, x_k) \log \frac{P(x_i, x_j|x_k)}{P(x_i|x_k)P(x_j|x_k)} \tag{2.5}$$

Kryteria wykrywania węzłów o dwu rodzicach podano na rysunkach 2.2 i 2.3. Zgodnie z przedstawionymi tam warunkami możliwe jest jednoznaczne określenie orientacji typu $X_i \rightarrow X_j \leftarrow X_k$. Pozostałe węzły orientujemy zgodnie tzn. tak, aby, w miarę możliwości, w pozostałych węzłach strzałki nie spotykały się głowami.

Algorytm Pearla

1. Niech G oznacza graf pełny o zbiorze wierzchołków tożsamym ze zbiorem atrybutów próby $\mathcal{D}$. Wówczas niech T będzie nieskierowaną strukturą drzewiastą uzyskaną w wyniku zastosowania dowolnego algorytmu znajdowania minimalnego drzewa rozpinającego (MST) w grafie G z funkcją wagową określającą stopień zależności między zmiennymi (tak jak w algorytmie Chow-Liu).

2. W T zorientować krawędzie w węzłach spełniających warunek posiadania dwu rodziców (rys. 2.2).

3. Zorientować krawędzie w pozostałych węzłach tak, by (o ile to tylko możliwe) nie były one zależne od dwu rodziców.

4. Krawędzie, których orientacja nie wynika jednoznacznie z poprzednich dwu kroków zorientować dowolnie, otrzymując wynikową strukturę sieci Bayesa B_S.

Będąc rozszerzeniem algorytmu, Chow-Liu algorytm Pearla dziedziczy większość jego właściwości. Otrzymana struktura nie jest jednak drzewem skierowanym, gdyż nie ma gwarancji, że między korzeniem i każdym innym węzłem istnieje ścieżka skierowana. Struktura zwracana przez algorytm Pearla ma postać *poli-drzewa*.

2.3.3 Algorytm SGS

Poniżej opisano procedurę, która sama posiadając bardzo ograniczoną skuteczność, stała się podstawą do skonstruowania całej serii algorytmów uczenia sieci przyczynowo skutkowych. Ten i następne algorytmy oparte o testy d-separacji podali Spirtes, Glymour, Scheines ([18]) w 1993 roku.

Definicja 3 *Niech S będzie ścieżką nieskierowaną w acyklicznym grafie skierowanym G. Wówczas wierzchołek X należący do S nazywamy wierzchołkiem kolidującym w S wtedy i tylko wtedy, gdy jest on wierzchołkiem wewnętrznym ścieżki (nie jest żadnym z jej końców) i krawędzie łączące X z jego bezpośrednimi sąsiadami w S są skierowane do X (np. $Y \rightarrow X \leftarrow Z$).*

Definicja 4 (Pearl[14]) *Niech* $\mathbf{X}$*,* $\mathbf{Y}$ *i* $\mathbf{Z}$ *będą rozłącznymi zbiorami wierzchołków w acyklicznym grafie skierowanym* G*. Wówczas mówimy, że* $\mathbf{Z}$ *d-separuje* $\mathbf{X}$ *i* $\mathbf{Y}$*, jeśli dla każdej pary wierzchołków, z których jeden pochodzi z* $\mathbf{X}$ *a drugi z* $\mathbf{Y}$ *nie istnieje żadna ścieżka nieskierowana spełniająca warunki: (i) wszystkie wierzchołki kolidujące na ścieżce należą do* $\mathbf{Z}$ *lub mają potomka w* $\mathbf{Z}$ *oraz (ii) wszystkie pozostałe wierzchołki na ścieżce nie należą do* $\mathbf{Z}$*.*

Autorzy nie wskazują metody testowania tej własności zakładając, że jest ona znana na wejściu algorytmu. Poniżej podano wszystkie założenia algorytmu:

1. Próba D nie zawiera zmiennych ukrytych.
2. Rozkład zmiennych próby $\mathcal{D}$ pochodzi od pewnego rozkładu, dla którego istnieje acykliczny graf skierowany opisujący zależności pomiędzy zmiennymi.
3. Próba spełnia warunki niezbędne do podejmowania statystycznie istotnych decyzji przez algorytm.
4. Znane są własności d-separacji między atrybutami w próbie $\mathcal{D}$ (tzn. można je sprawdzić).

Założenia trzecie i czwarte są bardzo silne i sposób ich traktowania w konkretnej implementacji istotnie wpływa na skuteczność i wydajność uczenia. Więcej szczegółów na temat rozwiązań zastosowanych w tej konkretnej pracy zawarte jest w rozdziale 3.3.3.

Algorytm SGS

1. Niech H będzie nieskierowanym grafem pełnym, którego zbiorem wierzchołków jest $\mathbf{V}$ (zbiór zmiennych próby $\mathcal{D}$).
2. Dla każdej pary wierzchołków X_i, X_j usuń z grafu H krawędź $X_i - X_j$, jeśli istnieje zbiór $S \subseteq \mathbf{V} \setminus \{X_i, X_j\}$ d-separujący X_i od X_j.

3. Niech K będzie nieskierowanym grafem uzyskanym w wyniku zastosowania wyczerpująco kroku drugiego. Dla każdej trójki wierzchołków sąsiadujących X_i, X_j, X_k w grafie K (oznaczenie: $X_i - X_j - X_k$) takich, że nie istnieje krawędź $X_i - X_k$, należy zorientować krawędzie do wewnątrz ($X_i \rightarrow X_j \leftarrow X_k$) wtedy i tylko wtedy, gdy nie istnieje zbiór wierzchołków $\mathbf{S} \subseteq \mathbf{V} \setminus \{X_i, X_k\}$, $X_j \in \mathbf{S}$ i $\mathbf{S}$ d-separuje X_i od X_k.

4. Powtarzać poniższe kroki, aż nie przynosi to orientacji żadnych krawędzi:

   Jeśli istnieje krawędź $X_i \rightarrow X_j$, X_k jest sąsiadem X_j, i X_k nie jest sąsiadem X_i, i krawędź $X_k - X_j$ nie jest zorientowana, należy ją zorientować jako $X_j \rightarrow X_k$.

   Jeśli istnieje ścieżka skierowana od X_i do X_k i niezorientowana krawędź $X_i - X_k$, należy ją zorientować $X_i \rightarrow X_k$.

Wadą algorytmu SGS jest jego wysoka złożoność obliczeniowa. Wynika ona z faktu, że dla każdej pary atrybutów próby $\mathcal{D}$ konieczne jest sprawdzenie własności d-separacji zwykle opartej o kosztowny test niezależności zmiennych. Koszt testu d-separacji w najgorszym przypadku wymaga sprawdzenia niezależności warunkowej względem niemal wszystkich możliwych podzbiorów zbioru wierzchołków.

2.3.4 Algorytm PC

Niestety w przypadku algorytmu SGS najgorszy przypadek jest przypadkiem oczekiwanym i ma miejsce dla każdej krawędzi, która pozostaje w grafie K po wykonaniu kroku drugiego. Właśnie dla tych krawędzi wykonywana jest ilość testów d-separacji zależna wykładniczo od ilości zmiennych w próbie.

SGS wykonuje także testy d-separacji wysokiego rzędu, które zwykle nie przynoszą oczekiwanych rezultatów ze względu na trudne do spełnienia wymaganie ogromnych prób wejściowych. Szybsze i efektywniejsze jest wykonywanie testów niższego rzędu. Taka strategia, minimalizująca ilość niezbędnych testów, zrealizowana została w algorytmie PC, będącym istotną modyfikacją SGS podaną przez tych samych autorów.

Niech $\mathbf{Sąsiedzi}_H(X)$ oznacza zbiór sąsiadów wierzchołka X w grafie H. Podczas działania algorytmu postać grafu H zmienia się w każdym kroku, więc wartość $\mathbf{Sąsiedzi}_H(X)$ również zmienia się dynamicznie.

Algorytm PC

1. Niech H będzie nieskierowanym grafem pełnym, którego zbiorem wierzchołków jest $\mathbf{V}$ (zbiór zmiennych próby $\mathcal{D}$).

2. $m := 0$

   Powtarzaj

   Powtarzaj

   Niech X_i, X_j będzie uporządkowaną parą wierzchołków sąsiadujących w H takich, że

   $$|\mathbf{Sąsiedzi}_H(X_i) \setminus \{X_j\}| \geqslant m \tag{2.6}$$

   Niech $\mathbf{S}$ będzie podzbiorem $\mathbf{Sąsiedzi}_H(X_i)$ o liczności m. Jeśli $\mathbf{S}$ d-separuje X_i i X_j należy usunąć krawędź X_i–X_j z grafu H i zapamiętać $\mathbf{S}$ w zbiorach $\mathbf{Sepset}(X_i, X_j)$ i $\mathbf{Sepset}(X_j, X_i)$.

   aż d-separacja zostanie sprawdzona dla wszystkich uporządkowanych par wierzchołków sąsiadujących takich, że spełniony jest warunek (2.6) i wszystkich podzbiorów $\mathbf{S} \subseteq \mathbf{Sąsiedzi}_H(X_i) \setminus \{X_j\}$ o liczności m.

   aż warunek (2.6) jest nie spełniony dla wszystkich uporządkowanych par wierzchołków w grafie H.

3. Dla każdej trójki wierzchołków X_i, X_j, X_k takiej że X_i, X_j są sąsiadami i X_j, X_k są sąsiadami, ale X_i, X_k nie są sąsiadami, zorientuj X_i–X_j–X_k jako $X_i \rightarrow X_j \leftarrow X_k$ wtedy i tylko wtedy, gdy $X_j \notin \mathbf{Sepset}(X_i, X_k)$

4. Powtarzać poniższe kroki, aż nie przynosi to orientacji żadnych krawędzi:

   Jeśli istnieje krawędź $X_i \rightarrow X_j$, X_k jest sąsiadem X_j, i X_k nie jest sąsiadem X_i, i krawędź $X_k - X_j$ nie jest zorientowana, należy ją zorientować jako $X_j \rightarrow X_k$.

   Jeśli istnieje ścieżka skierowana od X_i do X_k i niezorientowana krawędź $X_i - X_k$, należy ją zorientować $X_i \rightarrow X_k$.

2.3.5 Algorytm PC*

Opisany powyżej algorytm PC jest znacznie szybszy obliczeniowo od swojego protoplasty - SGS, jest także asymptotycznie poprawny, jednak wykonuje niepotrzebnie testy d-separacji na niektórych zbiorach wierzchołków sąsiednich **S**. Dla każdej pary X_i, X_j wykonywane są testy na potencjalnie wszystkich podzbiorach zbioru sąsiadów X_i i zbioru sąsiadów X_j. Tymczasem jeśli zmienne X_i i X_j są niezależne warunkowo, to, jak bezpośrednio mówi definicja d-separacji (def. 4), są niezależne pod warunkiem **S** podzbioru $\Pi(X_i)$ lub podzbioru $\Pi(X_j)$ złożonego tylko z wierzchołków leżących na nieskierowanych ścieżkach między X_i i X_j. Zatem wystarczy sprawdzać własność d-separacji względem takich zbiorów.

Niech $\mathbf{Sąsiedzi}^*_H(X_i, X_j)$ oznacza zbiór sąsiadów wierzchołka X_i w grafie H leżących na ścieżkach nieskierowanych pomiędzy X_i i X_j. Podczas działania algorytmu postać grafu H zmienia się w każdym kroku, więc wartość $\mathbf{Sąsiedzi}^*_H(X_i, X_j)$ również zmienia się dynamicznie.

Algorytm PC*

1. Niech H będzie nieskierowanym grafem pełnym, którego zbiorem wierzchołków jest **V** (zbiór zmiennych próby $\mathcal{D}$).

2. $m := 0$

   Powtarzaj

   Powtarzaj

   Niech X_i, X_j będzie uporządkowaną parą wierzchołków sąsiadujących w H takich, że

   $$|\mathbf{Sąsiedzi}^*_H(X_i, X_j) \setminus \{X_j\}| \geqslant m \tag{2.7}$$

   Niech **S** będzie podzbiorem $\mathbf{Sąsiedzi}^*_H(X_i, X_j)$ o liczności m. Jeśli **S** d-separuje X_i i X_j należy usunąć krawędź $X_i - X_j$ z grafu H i zapamiętać **S** w zbiorach $\mathbf{Sepset}(X_i, X_j)$ i $\mathbf{Sepset}(X_j, X_i)$.

   aż d-separacja zostanie sprawdzona dla wszystkich uporządkowanych par wierzchołków sąsiadujących takich, że spełniony jest warunek (2.7) i wszystkich podzbiorów $\mathbf{S} \subseteq \mathbf{Sąsiedzi}^*_H(X_i, X_j) \setminus \{X_j\}$ o liczności m.

aż warunek (2.7) jest nie spełniony dla wszystkich uporządkowanych par wierzchołków w grafie H.

3. Dla każdej trójki wierzchołków X_i, X_j, X_k takiej że X_i, X_j są sąsiadami i X_j, X_k są sąsiadami, ale X_i, X_k nie są sąsiadami, zorientuj $X_i – X_j – X_k$ jako $X_i \rightarrow X_j \leftarrow X_k$ wtedy i tylko wtedy, gdy $X_j \notin \mathbf{Sepset}(X_i, X_k)$

4. Powtarzać poniższe kroki, aż nie przynosi to orientacji żadnych krawędzi:

   Jeśli istnieje krawędź $X_i \rightarrow X_j$, X_k jest sąsiadem X_j, i X_k nie jest sąsiadem X_i, i krawędź $X_k – X_j$ nie jest zorientowana, należy ją zorientować jako $X_j \rightarrow X_k$.

   Jeśli istnieje ścieżka skierowana od X_i do X_k i niezorientowana krawędź $X_i – X_k$, należy ją zorientować $X_i \rightarrow X_k$.

2.3.6 Modyfikacje PC-2 i PC-3

W drugim kroku algorytmu PC wybierana jest para zmiennych do testu d-separacji. Szybsza eliminacja krawędzi z grafu H obniża koszty testów wykonywanych w kolejnych iteracjach drugiego kroku. Dlatego należałoby najpierw wybierać do testu te pary krawędzi X_i i X_j i te zbiory wierzchołków **S**, które najprawdopodobniej są d-separowane przez **S**. Autorzy algorytmu proponują tu trzy strategie:

PC-1 Wybierać pary i podzbiory wierzchołków w kolejności słownikowej.

PC-2 Wybierać najpierw te pary zmiennych, które są najmniej zależne od siebie (najmniejsza wartość statystyki χ^2).

PC-3 Wybierać najpierw te pary zmiennych X_i, X_j, które są najmniej zależne od siebie i te zbiory **S**, które są najbardziej probabilistycznie zależne od X_i.

Podane powyżej usprawnienia można zastosować również do algorytmu PC*. Ostatecznie powstaje więc grupa sześciu algorytmów rodziny PC: PC-1, PC-2, PC-3, PC*-1, PC*-2 i PC*-3 o analogicznej postaci.

2.3.7 Algorytm K2

Algorytm K2 (Cooper-Herskovits [6]) należy do grupy algorytmów heurystycznych posługujących się bayesowska funkcją dopasowania struktury sieci do rozkładu jako funkcją oceniającą. Dane wejściowe algorytmu K2 powinny spełniać wszystkie założenia opisane w paragrafie 2.2 oraz dodatkowo dwa warunki podane poniżej:

1. ξ jest kompletnym porządkiem w zbiorze zmiennych próby takim, że jeśli $X_i <_\xi X_j$ to w strukturze sieci nie występują łuki skierowane $X_j \rightarrow X_i$.
2. Znane jest górne ograniczenie liczby rodziców każdego węzła – u.

Rozważenie równań na prawdopodobieństwo odzwierciedlania struktury rzeczywistego rozkładu P przez daną strukturę sieci Bayesa B_S prowadzi do sformułowania bayesowskiej funkcji oceniającej dla algorytmu heurystycznego. Funkcja ta określa stopień dopasowania konfiguracji danego węzła X_i do rzeczywistego rozkładu, przy czym pod uwagę bierze jedynie krawędzie skierowane od węzłów ze zbioru $\Pi(X_i)$ do węzła X_i. Niech π oznacza zawartość zbioru $\Pi(X_i)$ w danej chwili działania algorytmu. Wówczas:

$$g(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{2.8}$$

gdzie q_i oznacza liczbę różnych wartościowań zmiennych ze zbioru π_i w próbie D, N_{ij} liczbę przypadków w D, takich że π_i ma wartościowanie w_{ij} ($j = 1...q_i$), a N_{ijk} liczbę przypadków takich, że zmienne z π_i przyjmują wartościowanie w_{ij} a zmienna X_i wartość ν_{ik} ($k = 1...r_i$).

Algorytm dodaje do każdego węzła nowych rodziców dopóki przynosi to poprawę wartości funkcji dopasowania lub, gdy osiągnięta zostanie maksymalna liczba rodziców określona przez użytkownika.

Algorytm K2

Poniższe kroki należy zastosować kolejno dla każdej zmiennej X_i próby D.

1. Niech X_i będzie ustaloną zmienną w próbie, $\pi_i := \emptyset$, $P_{old} := g(i, \pi_i)$

2. Powtarzaj poniższe kroki tak długo, aż nie będzie możliwe poprawienie wartości oceny konfiguracji wierzchołka X_i zawartej w zmiennej P_{old} lub wierzchołek będzie już miał zadaną przez operatora maksymalną liczbę rodziców u.

   Niech Z będzie zmienną z D taką, że $Z <_\xi X_i$ i wartość $g(i, \pi_i \cup \{Z\})$ jest największa.

   $P_{new} := g(i, \pi_i \cup \{Z\})$

   if $P_{new} > P_{old}$ **then**

   $P_{old} := P_{new}$

   $\pi_i := \pi_i \cup \{Z\}$, czyli dodaj do grafu krawędź $Z \rightarrow X_i$

   end

Wymaganie określenia z góry porządku zmiennych jest stosunkowo łatwe do określenia w tych zastosowaniach, w których poszczególne atrybuty próby występują w szeregu czasowym. Jednak określenie kompletnego porządku atrybutów może się często okazać bardzo trudne lub wręcz niemożliwe. Założenie to jest jedną z najsłabszych stron algorytmu K2. Podanie nieodpowiedniego porządku atrybutów w skrajnych przypadkach owocuje wygenerowaniem bardzo złej struktury sieci przyczynowo skutkowej, gdyż skierowania krawędzi odwrotne do określonych w porządku ξ nie są w ogóle testowane.

Dla stosunkowo niedużych prób możliwe jest stosowanie strategii wspomagających. Jeżeli ustalenie porządku nie jest możliwe, lub jest możliwie jedynie fragmentarycznie należy rozpocząć obliczenia dla różnych porządków a następnie porównać jakości otrzymanych wyników za pomocą jakiejś ogólnej metody oceny (np. funkcja *cross entropy* Kullback-Leibler'a) i wybrać sieć najlepiej dopasowaną.

2.3.8 Algorytm Ramowy

Algorytm ramowy ([11]) należy do grupy meta-algorytmów, czyli algorytmów operujących na innych algorytmach, zwanymi dalej algorytmami podrzędnymi. Proponuje on lokalne stosowanie algorytmu uczącego jako wsparcia dla funkcji oceniającej siłę wyboru lub orientacji poszczególnych krawędzi. Przyjęcie lub odrzucenie dodania/orientacji krawędzi zależy od tego, czy z grafu

tak utworzonego podrzędny algorytm uczący uzyskuje lepiej, czy gorzej dopasowaną strukturę wyjściową. Celem tej strategii jest unikanie przedwczesnej orientacji niektórych krawędzi, jak dzieje się to w przypadku algorytmów tradycyjnych.

Poniżej przedstawiono wersję korzystającą z algorytmu *pog-to-dag* podobnego do grupy PC i z *dag-from-pog* – algorytmu dokonującego zgodnej orientacji krawędzi. Prawdopodobnie można zaproponować inne skuteczne meta-algorytmy korzystające z innych procedur tradycyjnych jako algorytmów podrzędnych, które działałyby na podobnej zasadzie.

Algorytm pog-from-data

Niech$\mathbf{V}$ będzie zbiorem zmiennych próby $\mathcal{D}$, a H niezorientowanym grafem pełnym o wierzchołkach z $\mathbf{V}$.

1. Niech k oznacza maksymalną liczność zbioru warunkującego w poniższych krokach. $n := 0$.

2. Dla każdej pary zmiennych $X_i, X_j \in \mathbf{V}$ wierzchołków sąsiadujących w H takich, że każdy z nich posiada co najmniej n sąsiadów należy sprawdzić niezależność warunkową zmiennych względem każdego $\mathbf{S}$ podzbioru sąsiadów X_i o liczności n. Jeśli istnieje taki zbiór $\mathbf{S}$ to należy go zapamiętać jako $\mathbf{Sepset}(X_i, X_j)$ i usunąć krawędź $X_i - X_j$ z grafu H.

3. $n := n + 1$. Jeśli istnieje para wierzchołków sąsiadujących X_i, X_j w grafie H i każdy z tych wierzchołków ma w H więcej niż n sąsiadów należy powtórzyć krok drugi. w Przeciwnym razie należy kontynuować od kroku czwartego.

4. Dla każdej pary krawędzi $X_i - X_j - X_k$ takiej, że X_i, X_k nie są sąsiadami w grafie H, należy zorientować krawędzie do wewnątrz: $X_i \rightarrow X_j \leftarrow X_k$ jeśli $X_j \notin \mathbf{Sepset}(X_i, X_k)$.

5. Dla każdej pary krawędzi $X_i \rightarrow X_j - X_k$[2] takiej, że X_i, X_k nie są sąsiadami w grafie H, należy zgodnie zorientować krawędzie w prawo $X_i \rightarrow X_j \rightarrow X_k$

[2]Pierwsza krawędź zorientowana w prawo, druga niezorientowana

6. Dla każdej niezorientowanej krawędzi $X_i - X_j$ w H należy ustalić jej kierunek jako $X_i \rightarrow X_j$ jeśli w H istnieje ścieżka skierowana prowadząca od X_i do X_j.

7. Dla każdej czwórki zmiennych X_i, X_j, X_k, X_l takiej, że w grafie H istnieją i są zorientowane krawędzie $X_i \rightarrow X_j \leftarrow X_k$, krawędź $X_i - X_k$ nie istnieje, krawędź $X_j - X_l$ jest niezorientowana i co najmniej jedna z krawędzi $X_i - X_l$ lub $X_k - X_l$ jest niezorientowana, zorientuj $X_j \leftarrow X_l$.

8. Jeśli w krokach 5, 6, lub 7 dokonano orientacji wcześniej niezorientowanej krawędzi należy powtórzyć kroki poczynając od piątego. W przeciwnym wypadku działanie algorytmu jest zakończone.

Algorytm pog-to-dag

Definicja 5 *Wierzchołek X jest* legalnie usuwalny *z częściowo zorientowanego grafu H wtedy i tylko wtedy, gdy wszystkie zorientowane krawędzie incydentne z X są zorientowane w kierunku X i wszystkie pary krawędzi incydentnych z X, z których co najmniej jedna jest niezorientowana mają mosty (istnieje krawędź łącząca ich wierzchołki różne od X).*

1. W częściowo zorientowanym grafie H znajdź X wierzchołek legalnie usuwalny i usuń go wraz z krawędziami incydentnymi, zapisując, że usuwane krawędzie niezorientowane mają być zorientowane w kierunku X.

2. Powtarzaj poprzedni krok aż wszystkie krawędzie zostaną usunięte.

3. W oryginalnym grafie H (tzn. w grafie wejściowym algorytmu) zorientuj niezorientowane krawędzie zgodnie z informacjami zapisanymi w kroku pierwszym.

Algorytm Ramowy

1. Niech H będzie częściowo zorientowanym grafem pustym nad zbiorem wszystkich zmiennych w próbie $\mathcal{D}$.

2. Dla każdej pary wierzchołków X_i, X_j nie sąsiadujących w H wykonaj następujące kroki:

   (a) Zastosuj do grafu $H \cup \{X_i – X_j\}$[3] kroki 4-8 algorytmu *pog-from-data* i algorytm *pog-to-dag* otrzymując graph H'_{ij}

   (b) Jeśli nie pojawiła się sprzeczność (nie ma cykli w grafie H'_{ij}) to należy obliczyć funkcję dopasowania grafu H'_{ij} do rzeczywistego rozkładu próby $\mathcal{D}$.

   (c) Jeśli w grafie H istnieje zorientowana krawędź $X_i \leftarrow X_k$, gdzie $k \neq i \wedge k \neq j$, lub niezorientowana krawędź $X_i – X_k$ i wierzchołki X_j, X_k nie są sąsiadami wówczas:

      i. Utwórz graf tymczasowy dodając krawędź $X_i – X_j$, orientując krawędzie do X_i: $X_j \rightarrow X_i \leftarrow X_k$ i zastosuj do niego kroki 4-8 algorytmu *pog-from-data* i algorytm *pog-to-dag* otrzymując graph H''_{ij}.

      ii. Jeśli nie pojawiła się sprzeczność (nie ma cykli w grafie H''_{ij}) to należy obliczyć funkcję dopasowania grafu H''_{ij} do rzeczywistego rozkładu próby $\mathcal{D}$.

   (d) Symetrycznie zastosuj krok (c) do wierzchołka X_j (należy zamienić zmienne X_i i X_j w powyższym postępowaniu), otrzymując graf H'''_{ij} i jego wartość funkcji dopasowania.

3. Dodaj do grafu H krawędź niezorientowaną (z ewentualną orientacją tej i sąsiadującej krawędzi), której dodanie spowodowało otrzymanie grafu o najlepszym dopasowaniu spośród grafów $\{H'_{ij}, H''_{ij}, H'''_{ij}\}$[4].

4. Jeśli w poprzednim kroku dodano krawędź i warunek stopu nie jest spełniony przejdź do kroku drugiego, w przeciwnym razie zakończ działanie algorytmu.

Autorzy sugerują zastosowanie następującej miary dopasowania struktury do rzeczywistego rozkładu:

$$\sum_{i=1}^{n} \sum_{x_i, \pi(X_i)} \Big[P(X_i, \Pi(X_i)) \log P(X_i | \Pi(X_i)) \Big]_{x_i, \pi(X_i)} \tag{2.9}$$

[3] tzn. do grafu utworzonego z H przez dodanie krawędzi $X_i – X_j$

[4] Dla wszystkich nie sąsiadujących par wierzchołków X_i, X_j

gdzie $\Pi(X)$ oznacza zbiór rodziców wierzchołka X w grafie, a wewnętrzne sumowanie przebiega po wszystkich wartościowaniach zmiennej X_i i wszystkich wartościowaniach zbioru $\Pi(X_i)$ w próbie $\mathcal{D}$.

2.3.9 Algorytm FCI

FCI (*Fast Casual Inference*) jest jedynym algorytmem, spośród tu prezentowanych, rozwiązującym problem zmiennych ukrytych (czyli niereprezentowanych w próbie zmiennych, będącymi wspólnymi przyczynami zmiennych należących do próby). Należy on do grupy algorytmów SGS-PC i został opublikowany w tej samej pracy [18]. Tam też podano precyzyjną definicję zmiennych ukrytych. Przedstawienie algorytmu FCI wymaga wprowadzenia nowych pojęć i oznaczeń i niestety nieco skomplikowanej teorii. Wnikliwego czytelnika odsyłam do źródła [18], gdzie znajdzie wielostronicowy i bardzo dokładny opis. Poniżej zamieszczam jedynie niezbędne informacje.

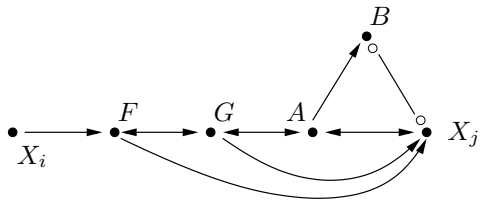
Spirtes, Glymour i Scheines wprowadzają nowy rodzaj grafu częściowo skierowanego reprezentującego struktury o zmiennych ukrytych[5]. Nowy rodzaj grafów może zawierać kilka rodzajów krawędzi: $X_i \rightarrow X_j$, $X_i \leftarrow X_j$ (krawędź skierowana), $X_i \circ\!\!\rightarrow X_j$, $X_i \leftarrow\!\!\circ X_j$ (krawędź skierowana do jednego wierzchołka, przy drugim orientacja jest nieznana), $X_i \circ\!\!-\!\!\circ X_j$ (orientacja krawędzi jest nieokreślona) i $X_i \leftrightarrow X_j$ (krawędź zorientowana w obie strony). Dodatkowo używany jest symbol „$*$" oznaczający dowolny rodzaj zakończenia krawędzi spośród znaku „$\circ$", strzałki i znaku pustego. Symbol „$*$" nie występuje w grafach a jedynie w opisie algorytmu.

Definicja 6 *Wierzchołek X_j nazywamy wierzchołkiem kolidującym na ścieżce nieskierowanej X_i, X_j, X_k w grafie częściowo skierowanym ze zmiennymi ukrytymi, jeżeli strzałki ścieżki spotykają się w X_j głowami (tzn. $X_i * \!\!\rightarrow X_j \leftarrow\!\! * X_k$).*

Definicja 7 *Wierzchołek X_j jest wierzchołkiem definitywnie niekolidującym na ścieżce nieskierowanej U, jeżeli U zawiera jedną z podścieżek $X_i \leftarrow X_j * \!\!- * X_k$, lub $X_i * \!\!- * X_j \rightarrow X_k$.*

Jeżeli o jakimś wierzchołku wiadomo, że jest definitywnie niekolidujący wzdłuż zadanej ścieżki (nawet jeśli krawędzie są niezorientowane) to oznacza się go podkreśleniem $X_i * \!\!- * \underline{X_j} * \!\!- * X_k$.

[5] *ang. Inducing path graph*



Rysunek 2.4: $\langle X_i, F, G, A, B, X_j \rangle$ jest ścieżką włączającą dla B pomiędzy X_i i X_j.

Dla tego rodzaju grafów autorzy wprowadzają wiele cech i własności analogicznych do własności grafów częściowo skierowanych reprezentujących klasy sieci Bayesa. Znaczenie tych pojęć jest zrozumiałe intuicyjnie i zrezygnowano ze szczegółowego ich przedstawiania.

Definicja 8 *Ścieżkę U nazywamy ścieżką włączającą dla wierzchołka M pomiędzy X_i i X_j, jeżeli ścieżka U jest ścieżką nieskierowaną o końcach X_i i X_j, wierzchołek B należy do ścieżki, a każdy wierzchołek na ścieżce poza B, X_i, X_j jest wierzchołkiem kolidującym lub definitywnie niekolidującym w U. Dodatkowo spełnione muszą być następujące warunki.*

(i) Jeśli wierzchołki X_k i X'_k są sąsiadami w U i X'_k jest pomiędzy X_k i B na ścieżce U, wówczas prawdziwe jest $X_k * \!\!\rightarrow X'_k$.

(ii) Jeśli X_k jest pomiędzy X_i a B na ścieżce U i X_k jest wierzchołkiem kolidującym w U, wówczas prawdą jest, że $X_k \rightarrow X_j$. W przeciwnym wypadku $X_k \leftarrow\!\!* X_j$.

(iii) Jeśli X_k leży pomiędzy X_j a B i X_k jest wierzchołkiem kolidującym na U wówczas prawdziwe jest $X_k \leftarrow X_i$. W przeciwnym wypadku $X_k \leftarrow\!\!* X_i$.

(iv) X_i i X_j nie są sąsiadami w grafie.

Rysunek 2.4 przedstawia przykładowy graf ze ścieżką włączającą.

Algorytm korzysta także z pojęcia zbioru możliwych wierzchołków d-separujących dwa wierzchołki X_i i X_j. Zbiór **Possible-D-SEP**(X_i, X_j) zawiera wszystkie wierzchołki X_k takie, że istnieje nieskierowana ścieżka w grafie pomiędzy X_i i X_k, na której każdy wierzchołek poza końcami jest

wierzchołkiem kolidującym lub jego orientacja jest ukryta, gdyż znajduje się w trójkącie.

Algorytm FCI

1. Niech H będzie nieskierowanym grafem pełnym nad zbiorem atrybutów próby $\mathcal{D}$.

2. $m := 0$

   Powtarzaj

   Powtarzaj

   Niech X_i, X_j będzie uporządkowaną parą wierzchołków sąsiadujących w H takich, że

   $$|\mathbf{Sąsiedzi}_{X_j}(X_i)H \setminus \{X_j\}| \geqslant m \tag{2.10}$$

   Niech $\mathbf{S}$ będzie podzbiorem $\mathbf{Sąsiedzi}_{X_j}(X_i)$H o liczności m. Jeśli $\mathbf{S}$ d-separuje X_i i X_j należy usunąć krawędź $X_i - X_j$ z grafu H i zapamiętać $\mathbf{S}$ w zbiorach $\mathbf{Sepset}(X_i, X_j)$ i $\mathbf{Sepset}(X_j, X_i)$.

   aż d-separacja zostanie sprawdzona dla wszystkich uporządkowanych par wierzchołków sąsiadujących takich, że spełniony jest warunek (2.7) i wszystkich podzbiorów $\mathbf{S} \subseteq \mathbf{Sąsiedzi}_{X_j}(X_i)H \setminus \{X_j\}$ o liczności m.

   aż warunek (2.7) nie jest spełniony dla wszystkich uporządkowanych par wierzchołków w grafie H.

3. Niech H' oznacza graf częściowo skierowany utworzony z nieskierowanego grafu otrzymanego w poprzednim kroku przez orientację wszystkich krawędzi jako $\circ\!\!-\!\!\circ$. Dla każdej trójki wierzchołków X_i, X_j, X_k takiej że X_i, X_j są sąsiadami i X_j, X_k są sąsiadami, ale X_i, X_k nie są sąsiadami w H', zorientuj $X_i *\!\!-\!\!* X_j *\!\!-\!\!* X_k$ jako $X_i *\!\!\rightarrow * X_j \leftarrow\!\!* X_k$ wtedy i tylko wtedy, gdy $X_j \notin \mathbf{Sepset}(X_i, X_k)$.

4. Dla każdej pary zmiennych X_i, X_j wierzchołków sąsiadujących w H' należy sprawdzić, czy X_i i X_j są d-separowane przez zbiór **S** będący podzbiorem **Possible-D-SEP**$(X_i, X_j)\backslash\{X_i, X_j\}$ lub **Possible-D-SEP**$(X_j, X_i)\backslash\{X_i, X_j\}$. Jeżeli tak, to w wynikowym grafie H'' nie znajdzie się krawędź łącząca X_i i X_j a zbiór **S** należy zapisać w **Sepset**(X_i, X_j) i w **Sepset**(X_j, X_i). Jeżeli d-separujący zbiór **S** nie istnieje to krawędź łącząca X_i i X_j w H' jest kopiowana do wynikowego H'' bez orientacji (tzn. $X_i \circ\!\!-\!\!\circ X_j$).

5. Dla każdej trójki wierzchołków X_i, X_j, X_k takiej że X_i, X_j są sąsiadami i X_j, X_k są sąsiadami, ale X_i, X_k nie są sąsiadami w H'', zorientuj $X_i *\!\!-\!\!* X_j *\!\!-\!\!* X_k$ jako $X_i * \!\!\rightarrow X_j \leftarrow\!\!* X_k$ wtedy i tylko wtedy, gdy $X_j \notin$ **Sepset**(X_i, X_k). Jeśli $X_j \in$ **Sepset**(X_i, X_k), krawędzie powinny być zorientowane w następujący sposób: $X_i *\!\!-\!\!* X_j *\!\!-\!\!* X_k$.

6. Poniższe kroki powinny być powtarzane w otrzymanym grafie kolejno, dopóki w każdym przebiegu jest orientowana co najmniej jedna krawędź.

   (a) Jeżeli istnieje ścieżka skierowana od X_i do X_j i krawędź $X_i *\!\!-\!\!* X_j$, należy ją zorientować w kierunku X_j, czyli $X_i * \!\!\rightarrow X_j$.

   (b) W przeciwnym wypadku, jeżeli X_j jest wierzchołkiem kolidującym na ścieżce wzdłuż X_i, X_j, X_k, a X_l jest wierzchołkiem sąsiadującym z X_j i wierzchołki X_i, X_k nie są d-połączone przez zbiór $\{X_l\}$, wówczas należy zorientować $X_j *\!\!-\!\!* X_l$ jako $X_j \leftarrow\!\!* X_l$.

   (c) W przeciwnym wypadku jeżeli U jest ścieżką włączającą dla wierzchołka M pomiędzy X_i i X_j, wierzchołki P i R są sąsiadami M w U i $P - M - R$ jest trójkątem wówczas:

      – Jeżeli $M \in$ **Sepset**(X_i, X_j) to M nie jest wierzchołkiem kolidującym na ścieżce, czyli $P *\!\!-\!\!* M *\!\!-\!\!* R$.

      – W przeciwnym wypadku jeśli M może być wierzchołkiem kolidującym na tej ścieżce to należy zorientować $P *\!\!-\!\!* M *\!\!-\!\!* R$ jako $P * \!\!\rightarrow M \leftarrow\!\!* R$.

      – W przeciwnym wypadku jeśli $P * \!\!\rightarrow M *\!\!-\!\!* R$ to $P * \!\!\rightarrow M \rightarrow R$.

Algorytm FCI w powyższej postaci zwraca strukturę grafu częściowo skierowanego, w którym niektóre krawędzie mogą być zorientowane ku obu wierzchołkom. Krawędzie te oznaczają, że ich wierzchołki mają wspólną ukrytą

przyczynę. Uzyskanie struktury sieci Bayesa wymaga zastosowania algorytmu usuwającego takie podwójne krawędzie i wstawiającego w ich miejsce nowe zależności oparte jedynie o węzły zawarte w próbie. Pozostawienie w sieci zmiennych spoza próby uniemożliwiłoby obliczenie tablic prawdopodobieńst B_P i stosowanie wynikowej sieci do wnioskowania. Pomocny okazuje się tu algorytm (*Edge Reversal Algorithm*) podany przez Shachtera [17] w 1986 roku. Algorytm ten służy odwracaniu kierunków łuków w sieciach tak, aby zachować istniejące własności d-połączeń rozkład prawdopodobieństwa. W naszym przypadku stosujemy go dwukrotnie, a następnie usuwamy ukryty węzeł. Kolejne etapy przedstawione są szczegółowo na rysunku 2.5.

Algorytm Edge Reversal[6]

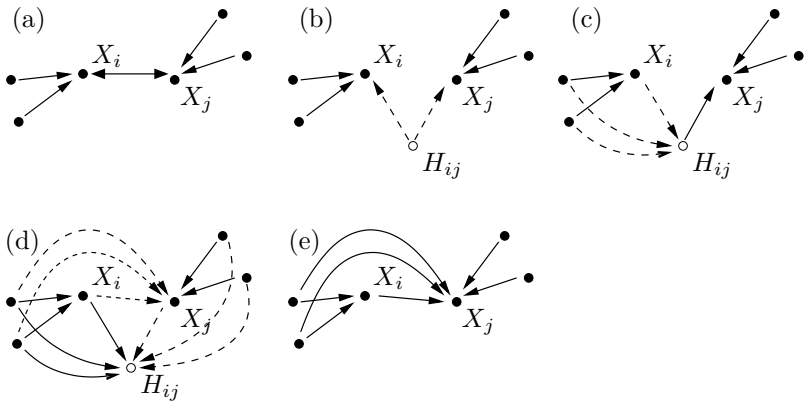
W rezultacie do każdego wierzchołka ukrytego krawędzi dwukrotnie skierowanej $X_i \leftrightarrow X_j$ (rys. 2.5a), wystarczy zastosować poniższe kroki:

1. Skierować zadaną krawędź w kierunku jednego tylko wierzchołka, np. $X_i \rightarrow X_j$.
2. Dodać krawędzie łączące wszystkie zmienne będące przyczynami zmiennej u początku strzałki (X_j) z węzłem u głowy (X_i) – wszystkie przyczyny X_i stają sie przyczynami X_j (2.5e).

2.4 Podsumowanie

W rozdziale tym przedstawiono i zilustrowano podstawowe pojęcia z dziedziny sieci Bayesa (definicja sieci, sieć drzewiasta, d-separacja, d-połączenie, główna zasada wnioskowania, itd.). Następnie zaprezentowano szereg algorytmów generejących struktury sieci (lub klasy struktur sieci) na podstawie informacji o rozkładzie zawartej w próbie statystycznej. Na zakończenie opisano nieco bardziej zaawansowany algorytm, który dodatkowo umożliwia wykrywanie na podstawie próby istnienie ukrytych zmiennych wpływających na zmienne jawne. Wszystkie te informacje okazały się niezbędne do wykonania implementacji systemu opisanego w następnym rozdziale.

[6] Autor dziękuje dr M.A.Kłopotkowi za przedstawienie zasady działania algorytmu.



Rysunek 2.5: Zastosowanie algorytmu Edge Reversal do usuwania podwójnie skierowanych krawędzi: a) podwójnie skierowana krawędź. b) interpretacja podwójnie skierowanej krawędzi jako wspólnej przyczyny (węzeł ukryty) c) i d) Odwrócenie krawędzi węzła ukrytego tak, by stał się liściem e) usunięcie dodanego węzła

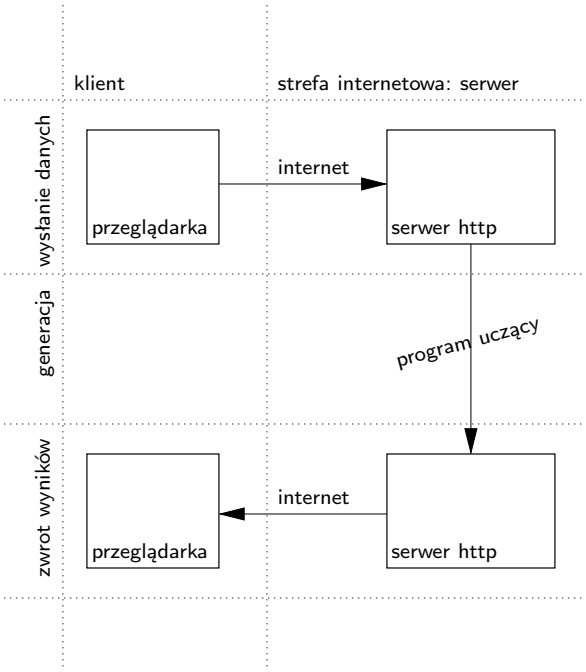
Rozdział 3

Internetowy system uczący *RENEG*

Rozdział obejmuje definicję i opis wykonanego zadania na różnych poziomach szczegółowości. Kolejne paragrafy omawiają aspekty funkcjonalne systemu, zasady użytkowania, instalacji i administracji oraz szczegółowe rozwiązania techniczne i algorytmiczne.

System generacji sieci przyczynowo-skutkowych RENEG powstał w ramach praktycznej części niniejszego projektu. Zadanie polegało na zaprojektowaniu i zrealizowaniu oprogramowania świadczącego usługi uczenia sieci Bayesa dla użytkowników zgłaszających się przez Internet. Strona klienta miała funkcjonować w dowolnym z powszechnie używanych systemów operacyjnych i przeglądarek WWW, a strona serwera co najmniej w systemie Microsoft Windows. Projekt systemu obejmował podział logiki między stację klienta i serwer (w tym podział funkcjonalności na rozproszone aplikacje), graficzny interfejs użytkownika, sposób przechowywania danych w systemie plików serwera, sposoby komunikacji między klientem i serwera oraz integrację z aplikacją wizualizującą wyniki (program BATMAN).

Przeznaczenie projektu jest akademickie, tzn. służyć ma on porównywaniu różnych algorytmów uczących zwłaszcza pod względem wydajności (szybkości) i efektywności (czyli jak bliskie rzeczywistości wyniki generuje). Z tego powodu zrezygnowano z typowych dla oprogramowania komercyjnego wzmocnionych sposobów zabezpieczeń i rozbudowanego interfejsu użytkownika. Uznano jednak, że niezbędny jest dostępny zdalnie system podstawowej



Rysunek 3.1: Diagram funkcjonalny systemu.

pomocy, gdyż potencjalny użytkownik nie musi mieć dostępu do odpowiedniej literatury. Wybrany interfejs użytkownika (zdalny interfejs WWW) bardzo sprzyja potencjalnemu odiorcy, gdyż środowisko akademickie w ramach uczelni i między uczelniami intensywnie wykorzystuje elektroniczne sieci rozległe.

Zamierzone badania wydajności narzuciły też konieczność korzystania z szybkich, kompilowanych języków programowania takich jak C/C++, a nie z nowocześniejszych, ale wciąż jeszcze wolniejszych technologii interpretowanych (Java, PHP, Perl. itp.). Oczywiście wymóg ten dotyczył tylko fragmentów obliczeniowych projektu, dlatego zaplanowano system hybrydowy, którego różne części powstały przy użyciu odpowiednich dla siebie technologii.

Na rysunku 3.1 przedstawiono zasadę funkcjonowania systemu.

Na osi poziomej zaznaczono obszary sieciowe funkcjonowania. System działa na serwerze http włączonym w Internet (prawa strona rysunku), ale

komunikacja z użytkownikiem odbywa się poprzez jego lokalną stację i przeglądarkę WWW (lewa strona rysunku).

Na osi pionowej zaznaczono podstawowe fazy interakcji systemu i użytkownika. Czytając od góry do dołu (wraz z upływem czasu) widzimy, że użytkownik najpierw dostarcza wszystkich niezbędnych do uruchomienia generacji danych wejściowych (obejmują one tabelę z próbą i parametry początkowe), następnie system rozpoczyna proces uczenia (część środkowa) i udostępnia wyniki przez Internet (na dole). Zaproponowany proces uczenia jest procesem bez nadzoru. Dlatego środkowa kratka w lewej kolumnie diagramu jest pusta – podczas generacji nie jest wymagana żadna ingerencja ze strony operatora.

Ogólnodostępna autorska instalacja systemu znajduje się pod adresem:

`http://bayes.cjb.net`

Niestety ze względu na ograniczenia obliczeniowe serwera dostępnego autorowi, nie jest możliwe przeprowadzanie obliczeń trwających dłużej niż minutę. Zwraca się uwagę, że nie jest to cecha samego systemu uczącego, a jedynie wynik arbitralnej konfiguracji tego konkretnego serwera WWW, który obsługuje także wiele innych zadań i użytkowników.

W niniejszej części przedstawiono zasady obsługi (paragraf 3.1) i eksploatacji (paragraf 3.2) systemu oraz szczegółowo omówiono praktyczne aspekty wykonanej implementacji (paragraf 3.3), opisując między innymi architekturę systemu, paradygmaty programowania i autorskie modyfikacje algorytmów przedstawionych w rozdziale 2.

3.1 Instrukcja użytkownika

Niniejsza część stanowi instrukcję obsługi zaimplementowanego systemu. Jako instrukcja stara się ona być możliwie samowystarczalna i wyjaśniać wszystko, co niezbędne użytkownikowi do skorzystania z systemu. Dlatego wiele spraw teoretycznych, lub bardzo technicznych traktuje bardzo pobieżnie, jednocześnie powtarzając niektóre bardzo podstawowe informacje uprzednio wymienione. Adresowana jest do użytkownika mającego podstawy wiedzy z zakresu generacji sieci przyczynowo-skutkowych, baz danych i korzystania z Internetu.

3.1.1 Wstęp – ogólna charakterystyka systemu

System RENEG jest systemem zdalnej generacji sieci Bayesa. Cała interakcja systemu z użytkownikiem odbywa się za pośrednictwem Internetu i przeglądarki WWW. System przyjmuje nadsyłane za pośrednictwem WWW tabele danych i generuje sieć Bayesa reprezentującą zależności między atrybutami wewnątrz tabeli. Interakcja systemu z użytkownikiem ma 4 wyraźne fazy opisane kolejno w paragrafach 3.1.5, 3.1.6, 3.1.7 i 3.1.8. Są to odpowiednio: dostarczenie danych, ustalenie parametrów, oraz potwierdzenie przyjęcia i odbiór wyników.

RENEG przyjmuje od użytkownika plik z próbą statystyczną o rozkładzie dyskretnym, który ustala rodzaj algorytmu i parametry wejściowe. Po kontroli danych aplikacja wyświetla potwierdzenie przyjęcia i rozpoczyna obliczenia. Ostatecznie wysyła pocztą elektroniczną wiadomość o zakończeniu i udostępnia wyniki. Proces ten został szczegółowo zilustrowany w kolejnych paragrafach.

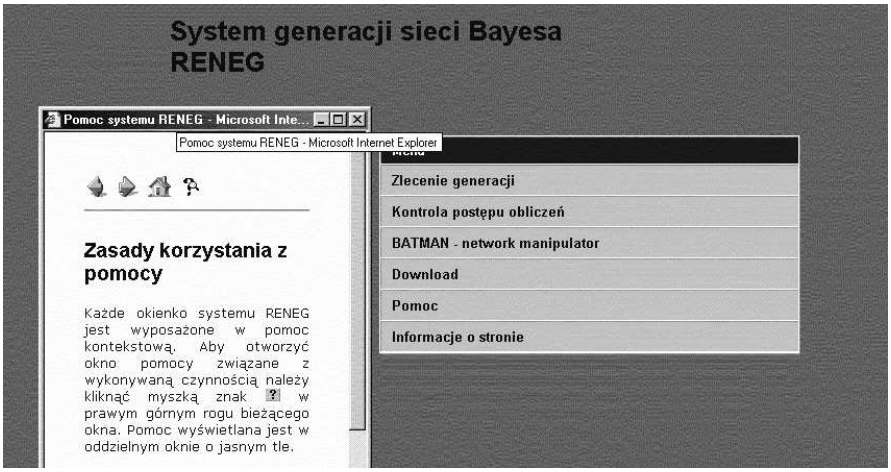
Niniejsze opracowanie stanowi dokumentację systemu. Internetowy charakter aplikacji przesądza jednak o konieczności dostarczenia opisu drogą elektroniczną – tak, aby każdy użytkownik mógł z niego skorzystać. Dlatego też system *RENEG* został wyposażony w pomoc kontekstową. Każde okno ma w prawym górnym rogu przycisk ze znakiem zapytania, który należy nacisnąć, aby uzyskać pomoc. Treść pomocy dostępnej on-line jest skróconą wersją niniejszej instrukcji.

Dodatkowo (po wybraniu polecenia download z menu głównego) użytkownik ma możliwość uzyskania niniejszego opracowania w postaci pliku w formacie PostScript.

3.1.2 Wymagania sprzętowe i systemowe

Każdy, kto chce skorzystać z systemu *RENEG* powinien posiadać komputer wyposażony w połączenie z Internetem i przeglądarkę internetową interpretującą JavaScript w wersji 1.2. Korzystanie z aplikacji wizualizacyjnej BATMAN wymaga dodatkowo Java Plugin 1.2 firmy SUN lub przeglądarki z maszyną wirtualną w standardzie Java 2.

Spośród produktów dostępnych na rynku można wskazać dwie przeglądarki, Internet Explorer w wersji 4.0 lub nowszej i Netscape Communicator



Rysunek 3.2: Menu główne systemu *RENEG*. Po lewej fragment otwartego okno pomocy kontekstowej.

for Windows w wersji 4.0 lub nowszej, które spełniają powyższe warunki (ściślej rzecz biorąc spełniają je po uzyskaniu z serwera firmy SUN pakietu Java Plugin 1.2 for Windows). Przeglądarki dostępne systemach uniksowych nie oferują jeszcze maszyny wirtualnej Java 2, co oznacza, że, przynajmniej tymczasowo, nie jest możliwe korzystanie z aplikacji BATMAN w środowisku uniksowym[1]. Sytuacja ta powinna wkrótce ulec zmianie.

Sprzęt komputerowy użytkownika musi spełniać wymagania systemu operacyjnego i oprogramowania przeglądarki. Nie jest konieczne, aby był to sprzęt szczególnie wydajny, zwłaszcza, jeżeli operator nie zamierza korzystać z programu BATMAN. Zasadnicza funkcjonalność systemu (generacja sieci) jest realizowana przy wykorzystaniu zasobów serwera WWW, dlatego wydajność sprzętu po stronie klienta nie ma znaczącego wpływu na wydajność systemu *RENEG*.

3.1.3 Format danych wejściowych (próby)

Podstawową informacją wejściową dla algorytmu jest próba danych z doświadczenia statystycznego. Próba powinna być zapisana w pliku .dbf, format

[1]Za wyjątkiem systemu Solaris.

dBase III/IV. Szczegółowy opis formatu jest dostępny w Internecie (między innymi pod adresem [21]).

Zmienne w próbie są dyskretne i przyjmują do 100 różnych wartości oznaczanych liczbami od 0 do 99. Każde zdarzenie w próbie stanowi jeden wiersz w tabeli z pliku dBase. Wszyskie kolumny są typu znakowego szerokości 2. Wszystkie komórki w tabeli muszą być wypełnione. Nazwy kolumn w pliku dBase są używane przez system *RENEG* jako nazwy zmiennych(węzłów) w wynikowej sieci Bayesa.

W skład systemu wchodzi zestaw plików przykładowych wygenerowanych przez autora i używanych jako dane testowe w niniejszej pracy. Są one dostępne w menu głównym systemu, pozycja *Download*. Sposób przekazywania danych testowych do systemu opisano w paragrafie 3.1.5.

3.1.4 Format danych wyjściowych (pliki ze strukturą sieci Bayesa)

Na podstawie otrzymanej próby i dodatkowych parametrów system zwraca plik zawierający tekstowy opis sieci przyczynowo-skutkowej w formacie BENIOFSKI. Format ten jest zbliżony do szerzej znanego formatu BNIF pochodzącego z *Decision Theory Group of Microsoft Research*. Kompletny opis formatu dostępny jest w [20]. Poniżej podane są jedynie elementy formatu wykorzystywane przez system *RENEG*.

Przykładową sieć i jej zapis w standardzie BENIOFSKI pokazano na rys. 3.3. Składnia tego zapisu zbliżona jest nieco do zasad języka C. Każdy obiekt opisywany w pliku posiada nazwę, po której następuje blok właściwości ujęty w nawiasy klamrowe. Format dopuszcza cztery rodzaje obiektów (bloków): sieć, właściwość, węzeł i tablica prawdopodobieństw warunkowych. Spośród nich tylko 3 (sieć, węzeł i tablica prawdopodobieństw) wykorzystywane są w systemie RENEG.

Blok sieci (`network`) definiuje nazwę sieci, format i języki naturalne użyte do opisu nazw. Format zakłada, że jest to pierwszy blok użyty do opisu pliku, jest on obowiązkowy i występuje dokładnie jeden. Po nim następuje jeden opcjonalny blok właściwości (nie wykorzystywany w systemie *RENEG*), a dalej kolejno grupa definicji węzłów i grupa tablic prawdopodobieństw.

Blok węzła (`node`) określa nazwę zmiennej i zbiór jej wartości. Liczba takich bloków nie jest ograniczona, ale wszystkie powinny być umieszczone

(a)

```
network "Nazwa opisowa sieci" {
        format is BNFormat;
        language is POL;
}

// definicje węzłów

node "N1" {
        type : discrete[2] choice of[val1, val2];
}
node "N2" {
        type : discrete[2] choice of[val1, val2];
}

// tablice prawdopodobieństw

probability ("N1") {
        default : 0.517, 0.483;
}
probability ("N2" | "N1") {
        (1) : 0.188,0.812;
        (2) : 0.761,0.239;
}
```

(b)

Rysunek 3.3: Przykładowa sieć przyczynowo-skutkowa zapisana w postaci tekstowej, w formacie BENIOFSKI (a) i przedstawiona graficznie (b).

obok siebie. Po grupie węzłów następuje grupa tablic prawdopodobieństw.

Blok tablicy prawdopodobieństw określa prawdopodobieństwa warunkowe przy kolejnych wartościowaniach zmiennych dla zmiennych i warunku pokazanego w nazwie bloku (w takiej kolejności w jakiej wartości zostały wymienione w blokach definiujących węzły). W przykładzie na rys. 3.3a są zatem zapisane poniższe wartości prawdopodobieństw:

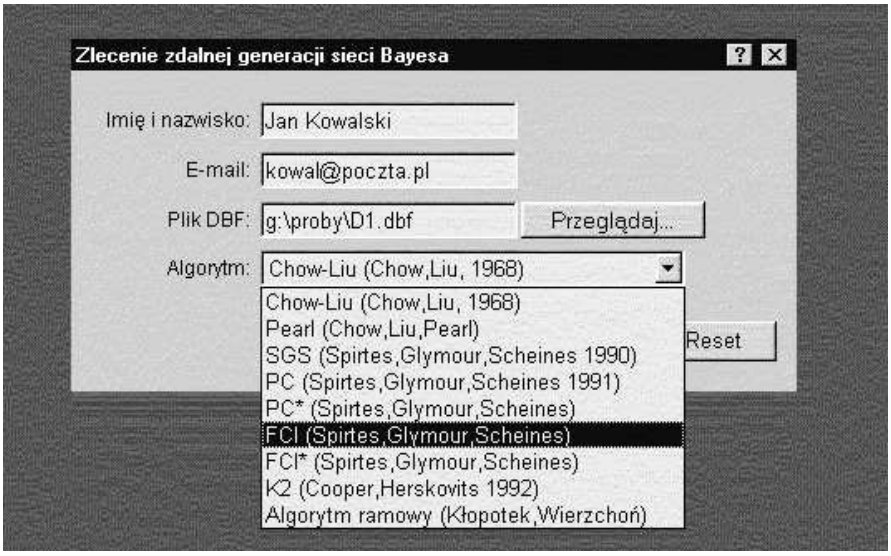
$$
\begin{aligned}
&P(N_1 = val_1) = 0.517 \\
&P(N_1 = val_2) = 0.483 \\
&P(N_2 = val_1 | N_1 = val_1) == 0.188, \quad P(N_2 = val_2 | N_1 = val_1) = 0.812 \\
&P(N_2 = val_1 | N_1 = val_2) == 0.761, \quad P(N_2 = val_2 | N_1 = val_2) = 0.239
\end{aligned}
$$

W systemie *RENEG* nazwa pliku wyjściowego składa się z 8-znakowego kodu dostępu generowanego przez system i rozszerzenia ".net" (np. `12345678.net`). W nazwie sieci (blok `node`) zawarta jest nazwa pliku wejściowego dBase i nazwa zastosowanego algorytmu uczenia. Nazwy węzłów są identyczne z nazwami atrybutów w pliku dBase. Ze względu na formułowanie przez system nazwy sieci i ewentualnych komentarzy systemu w języku polskim, w opisie sieci dodawany jest wiersz `language is POL`. Pliki generowane przez *RENEG* nie zawierają opcjonalnego bloku `property`.

Autor wyszedł z założenia, że wszystkie informacje, które nie opisują struktury sieci (np. komentarze, nazwy pól w różnych językach, dodatkowe właściwości, itp.) nie mogą być wynikiem sensownego działania algorytmu uczącego. Istotne wartości może nadać im tylko człowiek, często ekspert w danej dziedzinie i powinien uczynić to za pomocą odpowiednich narzędzi edycyjnych dla sieci przyczynowo-skutkowych, gdyż jest to o wiele wygodniejsze niż podawanie ich przy każdej generacji przez interfejs WWW.

Pliki generowane w systemie RENEG spełniają wymogi aplikacji wizualizującej BATMAN i są przez nią poprawnie odczytywane. W pakiecie zawierającym przykładowe pliki można znaleźć też przykładowe pliki `.net` odpowiadające wygenerowanym przez autora próbom (zob. menu główne, download).

Pliki generowane przez system przechowywane są na serwerze WWW. Sposoby uzyskiwania wyników opisano w rozdziałach 3.1.7–3.1.9.



Rysunek 3.4: Okno zlecenia generacji

3.1.5 Zlecenie generacji sieci

Zlecenia generacji dokonuje się po wybraniu odpowiedniego polecenia w menu głównym systemu. Użytkownik musi więc najpierw przygotować plik zawierający próbę i umieścić go na dysku widocznym w lokalnym systemie plików. Następnie uruchomić przeglądarkę WWW (Internet Explorer lub Netscape Navigator), połączyć się ze stroną WWW systemu (np. z instalacją autorską zob. początek rozdziału) i wybrać pierwsze polecenie z menu.

Okno zlecenia przedstawiono na rys. 3.4. W tym oknie należy podać informacje potrzebne do zlecenia obliczeń: imię i nazwisko, adres poczty elektronicznej, położenie pliku zawierającego próbę i rodzaj algorytmu uczącego.

Imię i nazwisko może być potem niezbędne do powtórnego zalogowania się w systemie i obejrzenia wyników. Należy więc pamiętać wpisaną wartość. Podawanie adresu e-mail nie jest obowiązkowe, jednak może okazać się bardzo wygodne, bowiem system wysyła na wskazany adres wiadomość o zakończeniu obliczeń.

Wskazany plik dBase zostanie wysłany do serwera po naciśnięciu przycisku [**Dalej**]. Przycisk [**Przeglądaj**] pomaga odnaleźć plik w lokalnym systemie plików użytkownika. Plik z próbą musi spełniać warunki wejściowe

systemu opisane w paragrafie 3.1.3.

Ostatnią informacją, która musi być podana w tej formatce jest nazwa algorytmu uczącego. Dostępne są algorytmy z poniższej listy.

algorytmy Chow-Liu i Pearla – algorytmy odtwarzające sieci drzewiaste z i bez badania orientacji zależności.

algorytm SGS – algorytm oparty o kosztowne testy niezależności zmiennych w próbie.

algorytmy PC i PC* – zoptymalizowane wersje SGS o obniżonej złożoności

algorytmy FCI, FCI* – modyfikacja algorytmów grupy PC dopuszczająca występowanie zmiennych ukrytych.

algorytm K2 – algorytm heurstyczny budujący grafy skierowane. Zakłada, że znany jest porządek wejściowy zmiennych.

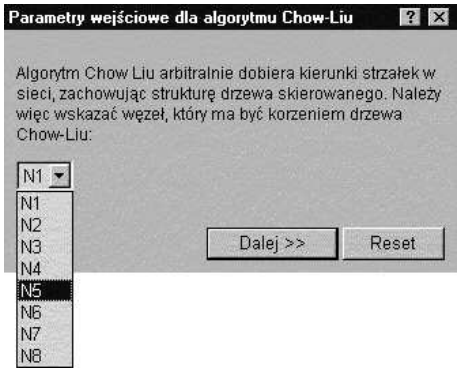
algorytm ramowy – meta-algorytm heurystyczny operujący na grafach częściowo zorientowanych.

Po wskazaniu algorytmu należy wybrać przycisk [**Dalej**] i oczekiwać na pojawienie się okna z pytaniem o parametry dla wybranego algorytmu. Przed pojawieniem się kolejnego okna może upłynąć trochę czasu, gdyż wskazany plik musi zostać załadowany na serwer i wstępnie zinterpretowany. Jeżeli próba jest duża a sieć bardzo obciążona, to czynność ta może zająć nawet więcej niż kilka minut.

Więcej informacji o parametrach poszczególnych algorytmów zawarto w kolejnych paragrafach. Zasady działania opisano szczegółowo w rozdziale drugim.

3.1.6 Parametry wejściowe algorytmów

Po załadowaniu pliku zawierającego próbę otwierane jest okno z parametrami wejściowymi algorytmu. Postać i ilość parametrów zależy od wybranego algorytmu, a czasem także od konkretnej próby. Po określeniu parametrów wejściowych (lub pozostawieniu wartości domyślnych) należy wybrać przy-



Rysunek 3.5: Parametry wejściowe algorytmu Chow-Liu

cisk [**Dalej**]. System zwraca informację o przyjęciu zlecenia i rozpoczyna obliczenia.

Algorytm Chow-Liu

Algorytm Chow-Liu należy do klasycznych algorytmów uczących. Odtwarza on jedynie sieci o strukturze drzewiastej. Algorytm nie sprawdza w żaden sposób, czy rzeczywisty rozkład próby ma strukturę drzewiastą, może się więc okazać przydatny przy poszukiwaniu drzewiastych przybliżeń niedrzewiastych struktur zależności.

Zasada działania algorytmu Chow-Liu jest niezmiernie prosta i polega na poszukiwaniu maksymalnego drzewa rozpinającego w grafie pełnym, w którym wierzchołkami są atrybuty próby. Jako funkcja wagowa używana jest tzw. odległość Kullback-Leibler'a.

Chow-Liu generuje nieskierowane drzewo zależności. Aby zwrócić poprawną sieć przyczynowo-skutkową system *RENEG* rozszerza oryginalną funkcjonalność, dokonując arbitralnego zorientowania krawędzi. Użytkownik musi podać korzeń drzewa (zob. rys. 3.5), a program zorientuje wszystkie krawędzie tak, by uzyskana struktura była drzewem skierowanym (w drzewie skierowanym dla każdego wierzchołka istnieje dokładnie jedna ścieżka skierowana prowadząca od korzenia).

Algorytm Pearla

Judea Pearl zaproponował rozszerzenie algorytmu Chow-Liu. Polega ono na wykorzystaniu odległości Kullback-Leibler'a do wykrycia niektórych orientacji strzałek i zgodnym zorientowaniu pozostałych.

Algorytm ten ma tylko jeden parametr wejściowy: użytkownik powinien określić numeryczne przybliżenie zera. Wartość ta jest wykorzystywana w warunkach podanych przez Pearla służących do wykrywania węzłów mających dwu rodziców - występuje tam bowiem porównanie odległości Kullback-Leibler'a z zerem.

Algorytm SGS

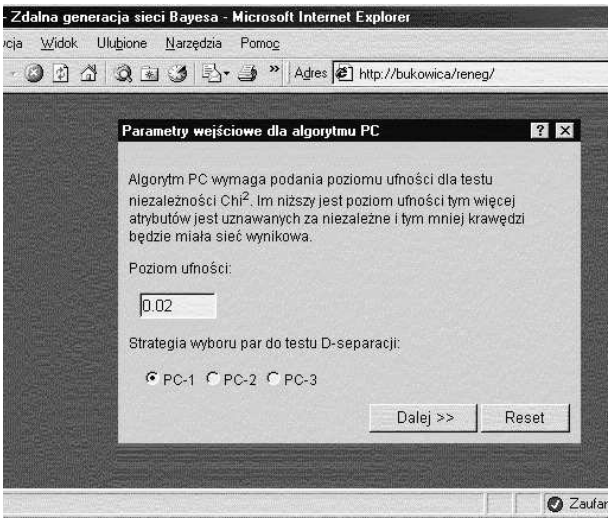
Działanie algorytmu SGS opiera się o proste testy d-separacji zmiennych. Ponieważ d-separacja zmiennych w sieci Bayesa jest równoważna niezależności, algorytm intensywnie wykonuje kosztowne testy niezależności i niezależności warunkowej na dostarczonym pliku. Z tego względu SGS jest najwolniejszym algorytmem systemu *RENEG*.

W pierwszej fazie generowana jest struktura sieci, po czym orientowane są niektóre krawędzie. W oryginalnej wersji zwracana jest struktura częściowo zorientowana reprezentująca klasę potencjalnych sieci Bayesa zgodnych z rozkładem. System *RENEG* stosuje algorytm *pog-to-dag*, aby uzyskać wybraną zorientowaną strukturę sieci Bayesa z klasy zwróconej przez SGS.

Algorytm wymaga podania poziomu ufności dla testu niezależności χ^2. Im niższy jest poziom ufności tym więcej atrybutów jest uznawanych za niezależne i tym mniej krawędzi będzie miała sieć wynikowa.

Algorytm PC i PC*

Algorytm PC powstał na bazie algorytmu SGS. Innowacja polegała na ograniczeniu kosztownych testów d-separacji przez określenie zbioru wierzchołków-kandydatów, które mogą d-separować daną parę zmiennych. Algorytm PC bada tutaj tylko sąsiadów pary wierzchołków, a PC* tylko sąsiadów leżących na ścieżkach nieskierowanych pomiędzy parą wierzchołków. W efekcie algorytmy te są wielokrotnie szybsze od swojego protoplasty zachowując podobną skuteczność.



Rysunek 3.6: Parametry algorytmów serii PC

W pierwszej fazie działania obu algorytmów następuje ustalenie struktury sieci, po czym w miarę możliwości następuje orientacja niektórych krawędzi. Generowana jest więc struktura częściowo zorientowana reprezentująca całą klasę potencjalnych sieci Bayesa. Niniejsza implementacja wykorzystuje algorytm *pog-to-dag*, aby uzyskać wybraną zorientowaną strukturę sieci Bayesa z klasy zwróconej przez PC.

Oba algorytmy przyjmują takie same parametry wejściowe (zob. rys. 3.6). Pierwszy to poziom ufności dla testu niezależności χ^2. Im niższy jest poziom ufności tym więcej atrybutów jest uznawanych za niezależne i tym mniej krawędzi będzie miała sieć wynikowa. Drugi parametr to strategia doboru par do testu d-separacji.

Test d-separacji wykonywany jest dla wszystkich par atrybutów w próbie. Złożoność testu dla każdej pary zależy od ilości krawędzi wyeliminowanych w poprzednich testach. Dlatego też wprowadzono dodatkowe strategie (PC-2 i PC-3) wybierające do testu najpierw te krawędzie, które mają największe prawdopodobieństwo odrzucenia. Strategia PC-1 testuje pary w kolejności słownikowej, czyli bez dodatkowej heurystyki.

Algorytm K2

Algorytm K2 jest algorytmem heurystycznym przeszukującym przestrzeń możliwych rozwiązań i budującym wynik na podstawie oceny sytuacji chwilowej. W każdym kroku K2 rozszerza rozwiązanie o jedną krawędź skierowaną – tą która najlepiej poprawia dopasowanie wyniku do rozkładu próby. Działanie jest przerywane, gdy poprawa nie jest możliwa do osiągnięcia. Algorytm korzysta z własnej funkcji oceniającej podanej przez Coopera i Herskovitsa.

K2 to sprawny i szybki algorytm, ale jego wadą jest wymóg podania porządku zmiennych ograniczającego możliwe kierunki strzałek. Jeśli zmienna X poprzedza Y w porządku, to Y nie może być rodzicem X. Wskazanie takiego porządku jest czasem łatwe, zwłaszcza gdy zmienne mierzone są w określonym szeregu czasowym.

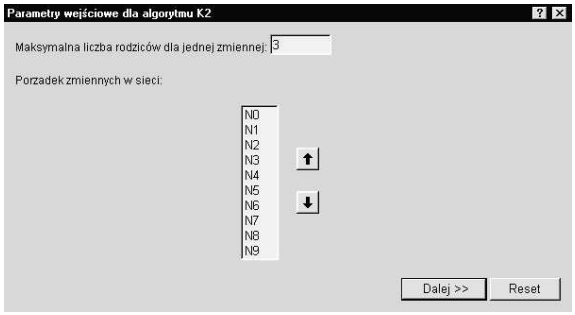
W sytuacjach, gdy zdefiniowanie porządku atrybutów jest trudne lub niemożliwe, stosowalność K2 jest istotnie ograniczona. W przypadku podania złego porządku zwrócona sieć może być zupełnie nieprzydatna. Pewną alternatywą jest wielokrotne uruchamianie algorytmu z różnym porządkiem zmiennych i wybranie wyniku, który ma najlepsze dopasowanie. System *RENEG* nie zwraca jednak informacji o dopasowaniu wyniku.

Algorytm K2 przyjmuje dwa parametry wejściowe (zob. rys. 3.7). Pierwszy z nich to maksymalna liczba rodziców dla jednej zmiennej. Wartość ta oznacza, że po znalezieniu takiej ilości krawędzi skierowanych do danego wierzchołka pozostali kandydaci nie będą rozpatrywani. Podanie liczby mniejszej niż liczba wierzchołków może istotnie skrócić czas uczenia dla dużych sieci, gdyż wartość ta bardzo ogranicza złożoność algorytmu. Nie ma ona większego znaczenia dla małych sieci, dla których algorytm i tak jest sprawny.

Drugim argumentem jest wspomniany porządek węzłów w sieci. System wyświetla listę zmiennych z dostarczonej próby i umożliwia zmianę kolejności. Aby przesuną w porządku jedną ze zmiennych należy wybrać myszką jej nazwę i skorzystać ze strzałek obok listy. Zaznaczając myszką więcej niż jedną zmienną możemy przesuwać lub rotować całe grupy.

Algorytm Ramowy

Algorytm ramowy jest algorytmem heurystycznym o działaniu zbliżonym do algorytmów K2 i BENEDICT. Na podobnej zasadzie przeszukuje przestrzeń



Rysunek 3.7: Parametry algorytmu K2

możliwych rozwiązań i buduje wynik na podstawie oceny bieżącej sytuacji.

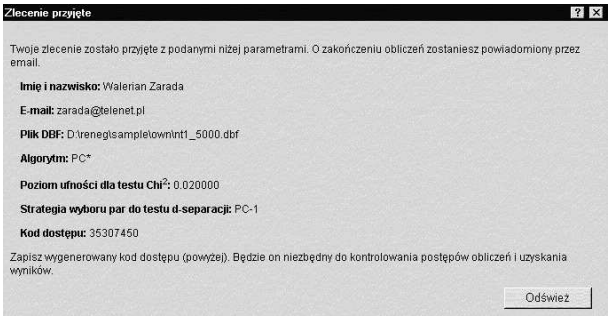
W odróżnieniu od K2 rozwiązanie budowane przez algorytm ramowy jest rozszerzane w każdym kroku o jedną krawędź skierowaną lub nieskierowaną, w zależności od tego, która najlepiej poprawia dopasowanie wyniku do rozkładu próby. W ten sposób algorytm ramowy unika przedwczesnej orientacji krawędzi, co jest słabością niektórych innych rozwiązań. Działanie jest przerywane, gdy poprawa większa niż założono nie jest możliwa do osiągnięcia.

Parametrem algorytmu ramowego jest wartość błędu względnego dla warunku stopu. Jeżeli w przebiegu pętli nie jest możliwe uzyskanie poprawy dopasowania o wartość większą niż podany parametr pomnożony przez wartość dopasowania z poprzedniego przebiegu, to działanie algorytmu jest przerywane a zbudowany do tej pory graf zwracany jako wynikowa struktura sieci Bayesa.

Algorytmy FCI i FCI*

Algorytm FCI jest jedynym algorytmem systemu RENEG uwzględniającym problem zmiennych ukrytych. W konsekwencji nie obowiązuje tu założenie, że dostarczona próba zawiera zamknięty ze względu na wzajemne zależności zestaw zmiennych. Główny mechanizm poszukiwania struktury sieci zaczerpnięto z algorytmu PC, rozszerzając go o kolejne kroki (między innymi poszukiwanie ścieżek wyłączających).

Podobnie jak w przypadku wszystkich pozostałych algorytmów, wynikowa sieć Bayesa udostępniona jest w formacie BENIOFSKI. Na końcu pliku dołączany jest komentarz z listą par zmiennych mających ukrytych rodziców.



Rysunek 3.8: Okno z potwierdzeniem przyjęcia zlecenia.

FCI operuje na grafach częściowo zorientowanych i zwraca strukturę reprezentującą całą klasę potencjalnych struktur sieci Bayesa. System *RENEG* stosuje więc dodatkowo połączenie algorytmów *pog-to-dag* i *edge reversal*, aby uzyskać jedną całkowicie zorientowaną strukturę sieci Bayesa z klasy zwróconej przez FCI.

FCI* różni się od FCI tym samym czym różni się PC* od PC, czyli wielkością zbiorów, względem których testowana jest własność d-separacji w pierwszej fazie działania. Oba algorytmy przyjmują takie same parametry wejściowe jak algorytm PC.

3.1.7 Potwierdzenie przyjęcia zlecenia i rozpoczęcie obliczeń

Okno *Zlecenie przyjęte* pojawia się na ekranie przeglądarki po ustaleniu i wysłaniu do serwera parametrów algorytmu. Jego pojawienie się jest znakiem, że obliczenia zostały już rozpoczęte. Okno zawiera wszystkie informacje o Twoim zleceniu: nazwisko i e-mail użytkownika, nazwę pliku, wybrany algorytm i jego parametry. Widoczny jest także czas rozpoczęcia obliczeń i tzw. kod dostępu (zob. rys. 3.8).

Kod dostępu wraz z podanym wcześniej imieniem i nazwiskiem jest niezbędny do korzystania z polecenia *kontrola postępu obliczeń* z menu głównego. Korzystanie z tego polecenia nie jest jednak jedynym sposobem uzyskania dostępu do wygenerowanych wyników. Jeżeli próba jest niewielka (co implikuje krótki czas obliczeń) wystarczy chwilę po zleceniu wybrać przycisk

[**Odśwież**], po którym następuje automatyczne (bez pytania o kod) przeniesienie do formatki postępu obliczeń. Sposób ten nie nadaje się do zastosowania dla dużych sieci, dla których czas obliczeń mierzymy w godzinach, bo wymaga utrzymywania włączonej stacji klienta i otwartej przeglądarki internetowej przez cały czas pracy (bardzo kłopotliwe w przypadku korzystania z połączenia dial-up).

Kod dostępu okazuje się także przydatny przy próbie wizualizacji wyników za pomocą aplikacji BATMAN (zob. p. 3.1.9).

3.1.8 Kontrola postępu obliczeń.

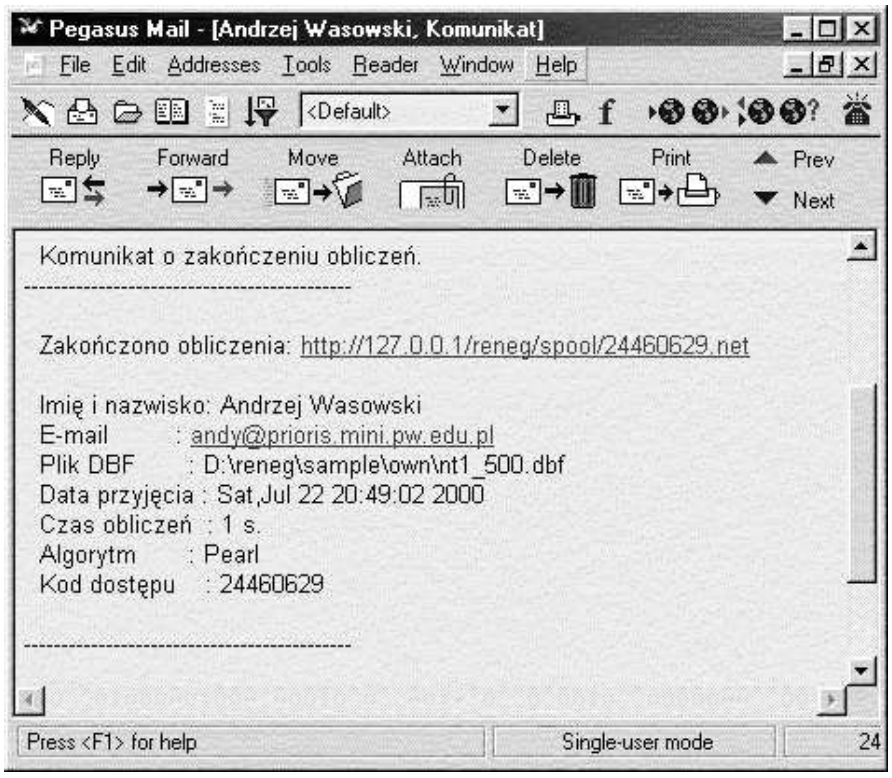
Użytkownik znający swój kod dostępu może skontrolować stan obliczeń za pomocą przeglądarki WWW. Wystarczy, wybrawszy z menu głównego pozycji *kontrola postępu obliczeń*, wpisać imię i nazwisko oraz kod dostępu i wybrać przycisk [**OK**]. Imię i nazwisko musi być wpisane dokładnie w taki sposób jak uczyniono to podczas zlecania obliczeń.

Okno *Stan obliczeń* jest bardzo podobne do okna *Zlecenie przyjęte* i zawiera wszystkie informacje o Twoim zleceniu: nazwisko i e-mail użytkownika, nazwę pliku, wybrany algorytm i jego parametry. Widoczny jest także czas rozpoczęcia obliczeń i kod dostępu.

W pierwszym wierszu podany jest status. Zapis *Trwa uczenie sieci* oznacza, że algorytm jeszcze działa i należy dalej czekać na wyniki (po chwili oczekiwania można wybrać przycisk odśwież i zobaczyć, czy coś się zmieniło).[2] Zapis Zakończono obliczenia oznacza, że działanie algorytmu zakończyło się poprawnie i zapisano plik ze strukturą. Informacja o zakończeniu jest jednocześnie hiperlinkiem do pliku wynikowego. Korzystając z poleceń przeglądarki można zapisać go na dysku lokalnym, lub obejrzeć w oknie (w postaci tekstowej). Możliwości graficznej wizualizacji podano w rozdziale poświęconym przeglądarce BATMAN (zob. p. 3.1.9).

Wygodnym sposobem kontroli postępu obliczeń może być często korzystanie z poczty elektronicznej. Jeżeli w oknie zlecenie generacji został podany adres e-mail, to system wyśle na niego wiadomość o zakończeniu obliczeń.

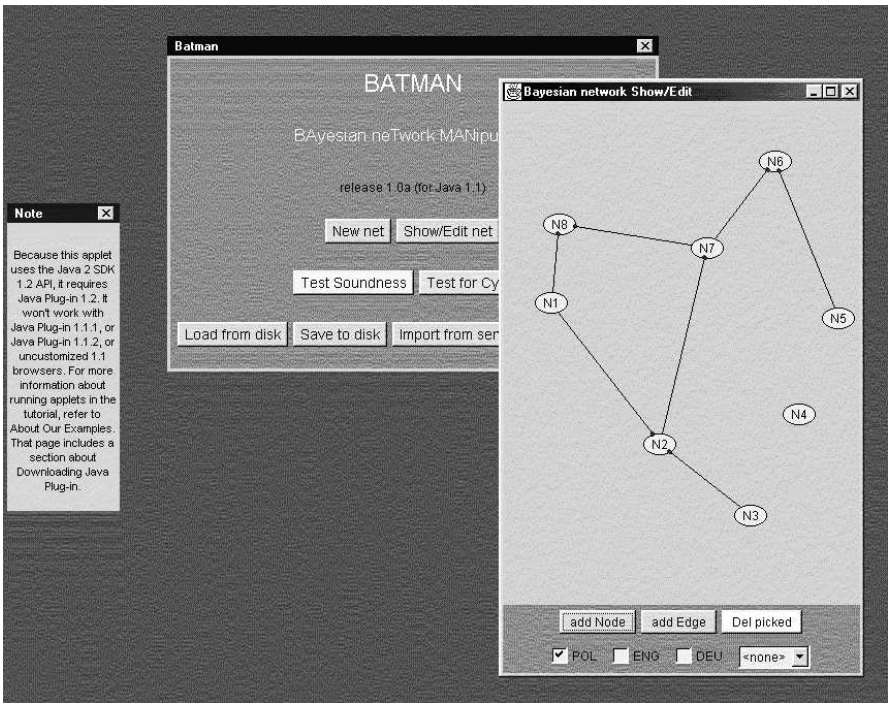
[2] Wyjątkiem jest sytuacja, gdy użytkownik otrzymuje pocztą elektroniczną powiadomienie o zakończeniu obliczeń z takim właśnie statusem. Wówczas oznacza on, że obliczenia zostały przerwane z powodu błędu. Jest to możliwe np. dla algorytmów grupy PC/FCI, jeżeli próba zawierała zbyt mało informacji i nie da się uniknąć utworzenia cykli.



Rysunek 3.9: Otwieranie pliku wynikowego ze środka programu pocztowego Pegasus Mail. Aby otworzyć plik sieci, wystarczy dwukrotnie kliknąć na URL w wierszu z napisem *Zakończono obliczenia*.

Wiadomość zawiera adres URL, pod którym można odebrać wyniki. Ponieważ w treści wiadomości podany jest też kod dostępu, nie ma potrzeby go nigdzie zapisywać. Wystarczy skorzystać z mechanizmu Cut&Paste, aby uruchomić *kontrolę postępu obliczeń*, lub obejrzeć wyniki za pomocą przeglądarki BATMAN.

Niektóre programy pocztowe (większość tzw. *mime-aware mail clients*) pozwalają obejrzeć wyniki w jeszcze prostszy sposób: wystarczy kliknąć myszką na adresie URL zawartym w powiadomieniu i program pocztowy sam uruchamia przeglądarkę z otwartym plikiem wynikowym (oczywiście niezbędne jest wówczas aktywne połączenie z Internetem).



Rysunek 3.10: Przykładowa sieć przedstawiona w programie BATMAN

3.1.9 Aplikacja wizualizacyjna BATMAN

Przeglądarka BATMAN jest programem zewnętrznym dołączonym do projektu. Z tego względu jej pełna integracja z systemem nie była możliwa. Uruchomienie przeglądarki wymaga instalacji Java plugin 1.2 (lub przeglądarki z wbudowaną maszyną wirtualną w standardzie Java 2).

Istnieją dwa sposoby wizualizacji wygenerowanych sieci Bayesa za jej pomocą. Pierwszy sposób polega na otworzeniu lokalnej kopii pliku wygenerowanej sieci (po uprzednim uzyskaniu jej z serwera – zob. paragraf 3.1.8). Aby otwarcie lokalnej kopii sieci było możliwe, należy nadać uprawnienia appletowi za pomocą narzędzia `policytool`, będącego częścią pakietu Java plugin 1.2. Jest to sposób polecany zaawansowanym użytkownikom.

Alternatywna metoda (nie wymagająca nadawania uprawnień) polega na odczytaniu pliku ze strukturą sieci bezpośrednio z serwera. Należy wybrać polecenie [**Import from server**] i wpisać nazwę pliku z wynikami. Jest ona

zbudowana z ośmiocyfrowego kodu dostępu i rozszerzenia `.net`.

Niezależnie od metody wczytania pliku, otwarcie okna ze strukturą sieci następuje po wybraniu przycisku [**Show/Edit Net**].

3.2 Instrukcja eksploatacyjna

Niniejsza część jest skierowana do osoby odpowiedzialnej za eksploatację systemu *RENEG* na serwerze WWW. Zawiera informacje niezbędne do instalacji, konfiguracji i nadzorowania bieżącej pracy.

3.2.1 Wstęp – ogólna charakterystyka strony serwera

System *RENEG* wykorzystuje serwer WWW jako serwer obliczeniowy realizujący generację sieci Bayesa. Dane dla systemu są dostarczane za pomocą mechanizmu CGI a odbierane przez proste połączenia http. Dodatkowym kanałem przepływu informacji od systemu do użytkownika jest poczta elektroniczna. Zarówno dane użytkownika jak i wyniki obliczeń są przechowywane w przestrzeni dyskowej serwera WWW.

3.2.2 Wymagania sprzętowe i programowe

Konfiguracja sprzętowa. System *RENEG* bardzo intensywnie wykorzystuje zasoby obliczeniowe serwera. Działanie chociażby jednego procesu obliczeniowego w znaczącym stopniu absorbuje zasoby jednostki obliczeniowej. Z tego względu zaleca się wydzielenie na potrzeby systemu oddzielnego serwera WWW. Jeśli nie jest to możliwe, należy się liczyć ze znacznymi utrudnieniami w pracy dla innych użytkowników tej samej maszyny[3].

Zaleca się używanie możliwie mocnego komputera z dużą ilością pamięci. Wartości oczekiwane to Pentium 200 Mhz i 64MB RAM. Większość prac wykonano jednak na komputerze: Pentium Celeron 533 ze 128MB RAM. Dodatkowo wymaga się około 20MB[4] przestrzeni dyskowej na instalację i kilkadziesiąt megabajtów w zapasie na pliki użytkowników.

[3] Aby uniknąć "pożerania zasobów" przez *RENEG* w systemach typu Unix wykorzystywanych do innych celów należy odpowiednio obniżyć priorytet programów uczących.

[4] Obejmuje programy, źródła, strony html, pliki przykładowe i dokumentację

System operacyjny: Windows 95/98, Windows NT, Linux lub FreeBSD. Uruchomienie na innych systemach uniksowych (zwłaszcza pochodnych BSD) nie powinno sprawić większych trudności. W dalszej części występuje zasadniczo rozróżnienie na dwa rodzaje instalacji uniksową (Linux, FreeBSD i inne) i Windows (95/98/NT).

Serwer WWW: Powinien wystarczyć dowolny serwer WWW obsługujący *Common Gateway Interface*. Instalacje przeprowadzono i testowano na serwerach Apache (platformy Windows/Linux/FreeBSD), WebSite 1.1 (platforma Windows) i Personal Web Server (plaforma Windows).

Transport poczty elektronicznej: Obie wersje instalacji umożliwiają komunikację z użytkownikiem przez pocztę elektroniczną. Domyślne ustawienia korzystają tu z aplikacji Pegasus Mail (platforma Windows) i polecenia mail (platforma Unix). Nic nie stoi na przeszkodzie, by skonfigurować wykorzystanie innych wsadowych systemów pocztowych, lub wręcz bezpośrednio oprogramowania transportującego pocztę. Zawsze jednak jakiś rodzaj tego oprogramowania jest wymagany. Wiecej informacji można znaleźć w sekcji 3.2.4.

3.2.3 Instalacja

Należy rozpakować archiwum `reneg.tar.gz` (lub `reneg.zip` w wersji Windows) do katalogu zwanego dalej *katalogiem głównym systemu*. W systemach Windows można to uczynić wykorzystując ogólnie dostępny program WinZip lub jedną z jego wielu pochodnych. W systemach uniksopodobnych należy wydać polecenie `tar xzvf reneg.tar.gz` lub kompatybilne. Rozpakowanie archiwum owocuje utworzeniem podstawowej hierarchi katalogów systemu.

W zależności od własności i konfiguracji serwera uniksowego może okazać się konieczne ręczne ustawienie flagi SUID na plikach programów i skryptach z katalogu `cgi-bin`.

Ilekroć w niniejszej dokumentacji podaje się nazwę katalogu nie podając jego lokalizacji, zakłada się, że znajduje się on w *katalogu głównym systemu*.

3.2.4 Konfiguracja

Na konfigurację systemu *RENEG* składają się 4 grupy czynności: konfiguracja serwera http, konfiguracja skryptów systemu, konfiguracja statycznych

stron systemu (nie zawsze konieczna) i konfiguracja transportu poczty elektronicznej. Opisano je w kolejnych paragrafach. Konfigurację wykonuje się jednorazowo – po instalacji systemu na serwerze. Nie powinna wymagać ona zmian po wdrożeniu.

Konfiguracja serwera WWW

Administrator powinien ustalić w drzewie serwera WWW następujące aliasy (zob. rys.3.11):

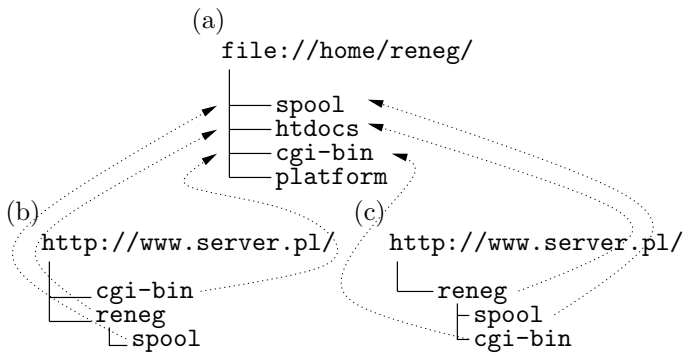
- alias, pod którym widoczne będą strony systemu *RENEG*, np. `http://moj.serwer.pl/reneg/`. Alias ten powinien wskazywać na katalog `htdocs` w katalogu głównym systemu.
- alias do katalogu, w którym oczekiwać będą na odbiór pliki wynikowe, np. `http://moj.server.pl/reneg/spool/`. Powinien on wskazywać na katalog `spool`.
- alias do katalogu z aplikacjami systemu, np. `http://moj.serwer.pl/reneg/cgi-bin/` z uprawnieniami do uruchamiania skryptów CGI. Powinien wskazywać do katalogu `cgi-bin`.

Jeżeli administrator systemu *RENEG* nie jest jednocześnie administratorem serwera WWW (np. dysponuje tylko kontem na tym serwerze z własną przestrzenią stron) może nie mieć uprawnień do mapowania struktury katalogów utworzonej podczas instalacji. W takiej sytuacji należy wybrane katalogi przenieść w miejsc wymagane przez system,[5] a w katalogu głównym instalacji utworzyć dowiązania symboliczne do rzeczywistych położeń.

Plik konfiguracyjny `reneg.ini`

Wszystkie aplikacje systemu korzystają z jednego scentralizowanego zbioru ustawień – pliku `reneg.ini`. Standardową lokalizacją dla tego pliku jest katalog `cgi-bin`. Plik konfiguracyjny działającego systemu powinien znajdować się w katalogu, który jest katalogiem bieżącym dla skryptów CGI uruchamianych przez serwer WWW.

[5] np. `htdocs` i `spool` do środka katalogu `public_html`, a `cgi-bin` do systemowego katalogu `cgi-bin` na serwerze.



Rysunek 3.11: Katalogi systemu plików (a) i aliasy serwera WWW: z globalnym(b) i z prywatnym(c) katalogiem `cgi-bin`. W obu przypadkach może zaistnieć konieczność dowiązania symbolicznego katalogów `cgi-bin` i `spool` do innych katalogów, gdy użytkownik nie ma prawa konfigurowania aliasów serwera http.

W przypadku serwerów Personal Web Server i Apache domyślna lokalizacja nie wymaga zmiany. Katalogiem bieżącym uruchamianyh skryptów jest tu bowiem katalog położenia skryptów. Serwer WebSite uruchamia wszystkie skrypty, ustawiając ich bieżący katalog na *ServerRoot* (katalog główny instalacji serwera WWW). W przypadku korzystania z tego (lub podobnego serwera) należy przenieść plik `reneg.ini` pod wspomnianą lokalizację.

W poniższych akapitach przedstawiono przykładowy plik `reneg.ini`, opisując znaczenie poszczególnych opcji. Pola opisywane są w takiej kolejności w jakiej występowały w oryginalnym pliku instalacji autorskiej. Wiersze rozpoczynające się znakiem # zawierają komentarze i są ignorowane przez aplikacje.

```
#Order of fields is insignificat except for ROOT field
#which should to be defined first.
ROOT=/home/andy/reneg/
```

ROOT – katalog główny instalacji. Należy tu wpisać położenie katalogu, w którym znajduje się drzewo systemu. Wartość tego pola jest używana jako przedrostek dla wartości następnych pól jeśli zawierają wartości względne. Pole to musi być zdefiniowane jako pierwsze w pliku. Kolejność pozostałych pól nie ma znaczenia.

```
HTML=htdocs
```

HTML – katalog zawierający pliki HTML projektu. Dopuszczalny adres względny (względem katalogu głównego systemu) lub bezwzględny (w systemie plików). To i 3 kolejne pola są interpretowane przez aplikacje systemu w kontekście lokalnego systemu plików, a nie drzewa katalogów serwera WWW. Zazwyczaj nie zachodzi potrzeba wpisywania tutaj wartości innej niż `htdocs`.

```
SPOOL=spool
```

SPOOL – położenie plików użytkowników i wyników obliczeń. Wartość względna w katalogu głównym projektu, lub wartość bezwzlędna w systemie plików.

```
BIN=cgi-bin
```

BIN – położenie skrytpów CGI. Wartość względna w katalogu głównym projektu, lub wartość bezwzględna w systemie plików.

```
LOG=spool/reneg.log
```

LOG – położenie i nazwa centralnego logu błędów systemu. Plik ten zawiera informacje o błędach, które wystąpiły niezależnie od generacji konkretnej sieci. Każda sesja generacji ma niezależnie od tego swój własny log zdarzeń w pliku `spool/id_zlecenia.log`.

```
NOTIFICATION=notification.txt
```

NOTIFICATION – nazwa pliku zawierającego wzorzec wysyłanego pocztą elektroniczną powiadomienia o zakończeniu obliczeń. Nie należy podawać położenia pliku. Plik jest poszukiwany w katalogu określonym zmienną `HTML`. Zazwyczaj nie ma potrzeby modyfikowania tego pola, zaleca się natomiast dostosowanie jego zawartości (zob. p. 3.2.4).

Kolejna grupa ustawień odnosi się do aliasów w drzewie serwera WWW (a nie katalogów w systemie plików).

```
HTTP_SERVER=moj.serwer.pl
```

HTTP_SERVER – nazwa lub adres IP serwera systemu *RENEG*.

```
HTTP_ROOT=/reneg/
```

HTTP_ROOT – przedrostek podanych poniżej aliasów. Jest on dodawany do wartości poniższych pól jeśli nie rozpoczynają się znakiem slash (/).

```
HTTP_BIN=/cgi-bin/
```

HTTP_BIN – położenie katalogu cgi-bin systemu, W powyższym przykła-

dzie pokazano wartość właściwą dla instalacji korzystającej z głównego aliasu skryptowego na serwerze. Zaowocuje ona wywołaniami skryptów z lokalizacji `http://moj.serwer.pl/cgi-bin/`. Podanie wartości `cgi-bin/` spowodowałoby poszukiwanie skryptów pod aliasem

`http://moj.serwer.pl/reneg/cgi-bin/`

`HTTP_SPOOL=spool/`

HTTP_SPOOL – alias do katalogu z plikami wynikowymi. Wartość względna lub bezwzględna na zasadzie identycznej do powyższej.

HTTP_HTDOCS – alias do stron HTML systemu *RENEG*. Wasrtość zazwyczaj pozsotaje pusta, chyba, że przedrostek `HTTP_ROOT` nie wskazuje bezpośrednio do katalogu stron HTML.

Konfiguracja stron HTML

Stosowanie mieszanej technologii statycznego HTML i dynamicznego HTML z użyciem CGI ma niestety jedną wadę. Odwołania do skrytpów CGI ze statycznych stron HTML muszą być zmieniane jeśli zmieniane jest położenie aliasu `cgi-bin`. Dokładniej jest to konieczne, gdy alias `cgi-bin` przenoszony jest na inny poziom zagnieżdżenia w drzewie serwera.

Komplikacji tej możnaby uniknąć, opierając działanie systemu całkowicie o dynamicznie budowany HTML. Jednak takie rozwiązanie obciąża niepotrzebnie serwer i generuje zbędny ruch w sieci (zob. p. 3.3.2), gdyż nawet strony o statycznym charakterze wymagają uruchamiania skryptów i wysyłania do przeglądarki przy każdym żądaniu pobrania z pominięciem lokalnego bufora i serwerów proxy.

Autor zdecydował się na wariant alternatywny: system zawiera statyczne strony, ale mogą one wymagać modyfikacji administratora (jednorazowo po dokonaniu instalacji). Zmian wymagać mogą 2 pliki: `upload.html` i `status.html`. W domyślnej instalacji pliki te przystosowane są do uruchamiania skryptów z aliasu:

`http://adres_serwera/alias_systemu_reneg/cgi-bin/`

Jeżeli skrypty z tej lokalizacji nie mogą być uruchamiane, należy zmienić wywołania CGI na odpowiednio inne. Przykładowo, jeżeli skrypty mają być uruchamiane z globalnego katalogu CGI na serwerze (`http://adres_serwera/cgi-bin/`), to zamiast domyślnego `cgi-bin/` należy wpisać wywołanie `/cgi-bin/`.

Fragment nagłówka pliku `upload.html` tak jak zawarto w dystrybucji:

```
<FORM METHOD="POST" ACTION="cgi-bin/upload.exe"
 NAME="uploadForm" ENCTYPE="multipart/form-data">
```

Wywołanie następuje z katalogu (aliasu) `cgi-bin` znajdującego się poziom niżej od katalogu głównego systemu w drzewie serwera WWW. Aby skrypty muszą być uruchamiane z globalnego `cgi-bin` należy powyższy fragment zamienić na:

```
<FORM METHOD="POST" ACTION="/cgi-bin/upload.exe"
 NAME="uploadForm" ENCTYPE="multipart/form-data">
```

Modyfikacja pliku `status.html` przebiega analogicznie. Pozostałe pliki nie wymagają ingerencji.

Transport poczty elektronicznej i lokalizacja powiadomień

System *RENEG* wysyła powiadomienie do użytkownika zlecającego generację każdorazowo po zakończeniu obliczeń (lub w sytuacji awaryjnej – po przerwaniu obliczeń z powodu błędu). Przykładowe powiadomienie przedstawiono w instrukcji obsługi (zob. p. 3.1.8 i rys. 3.9).

Format wiadomości jest zapisany w pliku `htdocs/notification.txt`. Przykładowa zawartość zaprezentowana jest na rys. 3.12. Jego postać może być dowolnie zmieniana z wykorzystaniem zawartych w nim makr. Wystąpienie wszystkich makr nie jest obowiązkowe. Zaleca się jednak pozostawienie wywołania `<!--VAR text_status>`. Jest ono rozwijane do tekstu informującego na jakim etapie program przerwał/zakończył pracę. W przypadku pomyślnego wygenerowania pliku `.net` rozwinięcie `text_status` zawiera także dowiązanie URL do pliku wynikowego.

Wiadomość wysyłana jest za pomocą programu zewnętrznego na wskazany przez użytkownika adres e-mail. Standardowa dystrybucja systemu *RENEG* zawiera dwa skrypty wysyłające powiadomienia: `email.sh` (w wersji

```
  Komunikat o zakończeniu obliczeń.
-----------------------------------------

  <!--VAR text_status-->

  Imię i nazwisko: <!--VAR user-->
  E-mail         : <!--VAR email-->
  Plik DBF       : <!--VAR filename-->
  Data przyjęcia : <!--VAR received-->
  Czas obliczeń  : <!--VAR elapsed-->
  Algorytm       : <!--VAR algorithm-->
  Kod dostępu    : <!--VAR id-->

-----------------------------------------
System zdalnej generacji sieci bayesa RENEG.
http://<!--VAR http_server-->
```

Rysunek 3.12: Standardowy wzorzec powiadomienia. Zmienne w prawej kolumnie zamieniane są na odpowiednie wartości zgodnie z nazwami w lewej kolumnie

```
#!/bin/sh
cat $1 | mail -s "$3" $2
rm -f $1
```

Rysunek 3.13: Standardowy skrypt (`email.sh`) wysyłający powiadomienie w systemie uniksowym.

```
"J:\Program Files\pmail\winpm-32" -T %2 -F %1 -S "%3"
del %1
```

Rysunek 3.14: Standardowy skrypt (`email.bat`) wysyłający powiadomienie w systemie Microsoft Windows. Skrypt ten należy zmodyfikować w każdej instalacji Windows.

uniksowej) i `email.bat` (w wersji dla Microsoft Windows). Oba skrypty pokazane są na rysunkach, odpowiednio 3.13 i 3.14.

O ile skrypt `email.sh` powinien zachowywać się poprawnie w większości systemów uniksowych (program `mail` należy do standardowych narzędzi systemu), to `email.bat` zawsze wymaga modyfikacji. W standardowej wersji zawiera on wywołania programu *Pegasus Mail* powodujące wsadowe wysłanie pliku na wskazany adres. Można w tym celu wykorzystać inny program dostępny na serwerze WWW.

Oba skrypty otrzymują 3 parametry: ścieżkę dostępu do pliku tekstowego, który należy wysłać a po wysłaniu usunąć, adres poczty elektronicznej adresata i temat wiadomości (w podanej kolejności). W systemie Windows interpretacja linii poleceń zależy od zastosowanego programu wysyłającego pocztę. W niektórych wypadkach może okazać się konieczne ujęcie niektórych parametrów w cudzysłowy. Należy pamiętać, że temat wiadomości zawsze zawiera spacje, a nazwa pliku będzie je zawierać, jeżeli zawierając je katalogi skonfiurowane w pliku `reneg.ini`.

3.2.5 Przeglądarka BATMAN

BATMAN – przeglądarka sieci Bayesa, udostępnia dwa sposoby pobierania danych do wizualizacji: z lokalnego dysku użytkownika i z sieciowego dysku serwera WWW. Oba sposoby wymagają konfiguracji. Odczyt plików z lokalnego dysku wymusza na użytkowniku nadanie appletowi uprawnień za pomocą narzędzia `policytool`. Operacji tej na pewno nie mogą wykonać wszyscy użytkownicy, ze względu na brak doświadczenia w nadawaniu uprawnień apletom. Dlatego zaleca się, aby administrator skonfigurował serwerową stronę BATMANa tak, by możliwe było podawanie plików bezpośrednio z katalogu `spool` z serwera WWW.

Zmiana konfiguracji polega na zmianie domyślnych serwerów i katalogów programów `bayesin.exe` i `bayesout.exe` oraz ich rekompilacji. Alternatywnie można wykorzystać ich wersje skryptowe `bayes-in.cgi` i `bayes-out.cgi` – zmiana jest wówczas o wiele łatwiejsza i nie wymaga rekompilacji. Oba skrypty zawierają ścieżkę dostępu do katalogu wejściowego, którą należy zmienić na ścieżkę wskazującą na katalog `spool`. W każdym wypadku jest jednak niezbędna rekompilacja klasy `Config` i umieszczenie jej w archiwum `batman.jar` zgodnie z dokumentacją aplikacji BATMAN.

W pakiecie instalacyjnym zawarto oba skrypty zawierające względne odwołania do katalogu `spool`. Dlatego w wielu systemach żadne zmiany nie są wymagane.

3.2.6 Nadzór

Nadzór eksploatacyjny nad systemem ogranicza się do regularnej kontroli zawartości katalogu `spool`. System nie usuwa plików danych automatycznie. Odpowiedzialność za ograniczanie zajmowanej przestrzeni dyskowej (lub przeznaczanie dla systemu dodatkowej przestrzeni) spoczywa całkowicie na administratorze.

Ilość w plików w katalogu `spool` przyrasta z każdym zleceniem użytkownika. Pliki związane z tym samym zleceniem mają takie same nazwy (tzn. pierwsze 8 znaków bez rozszerzenia). Rozszerzenia plików wskazują na ich zawartość.

Najwięcej przestrzeni dyskowej zajmują pliki baz danych (rozszerzenie `.dbf`). Pliki te można usunąć dla zleceń, dla których zakończono już obli-

czenia (istnieje plik o tej samej nazwie i rozszerzeniu `.net`). Pliki zawierające wyniki (rozszerzenie `.net`) i informacje dodatkowe (rozszerzenie `.tag`) zajmują zwykle stosunkowo niewiele miejsca. Zazwyczaj można sobie pozwolić na długotrwałe ich przechowywanie. Pliki tymczasowe powiadomień dla użytkownika (rozszerzenie `.tmp`) powinny być usuwane automatycznie przez skrypt wysyłający (`email.bat` lub `email.sh`) natychmiast po wysłaniu wiadomości.

W plikach `*.log` zapisywane są informację diagnostyczne systemu, pomocne przede wszystkim podczas uruchamiania implementacji. Pliki te mogą (a nawet powinny) być usuwane z serwera,

Administrator powinien ustalić czas przechowywania starych plików na serwerze i podać go do wiadomości użytkowników na stronie systemu *RENEG*. Pliki przebywające na tym serwerze poza wyznaczony okres powinny być usuwane (czynność łatwa do zautomatyzowania). Takie podejście pozwala zapanować nad ilością przestrzeni dyskowej zajmowanej przez system.

3.2.7 Rekompilacja projektu

W katalogu głównym instalacji projektu znajdują się trzy podkatalogi ściśle związane z rekompilacją: `src` (źródła projektu), `platform` (profile kompilacji dla różnych systemów) i `cgi-bin` (binaria wynikowe). Jest bardzo ważne, aby struktura katalogów pozostała nienaruszona. Nawet jeśli pliki binarne muszą znaleźć się w innym katalogu, ze względu na ustawienia serwera, należy utworzyć dowiązanie do rzeczywistej lokalizacji.

Katalog `platform` jest podzielony na podkatalogi dla różnych systemów. W podkatalogu `VisualC` znajdują się pliki *Workspace* z Microsoft Visual Studio 6.0. W podkatalogach `linux` i `freebsd` znajdują się pliki `Makefile` dla tych systemów.

Uwaga!: Zaleca się, aby wszelkie poprawki wprowadzane w systemie zachowywały jego przenośność między różnymi platformami.

Rekompilacja w Microsoft Visual Studio

Należy uruchomić Visual C/C++ i otworzyć plik zawierający przestrzeń projektów `reneg.dsw`. Następnie skompilować projekty odpowiednim poleceniem Visual Studio (w wersji 6.0 był to klawisz F7). Skompilowane pliki są

przenoszone automatycznie do katalogu `cgi-bin` w katalogu głównym projektu.

Visual Studio tworzy bardzo dużo plików tymczasowych podczas kompilacji. Można usunąć je poleceniem *Clean*. Wersję wdrożeniową należy kompilować w trybie *Release*, gdyż wpływa to znacząco na wydajność aplikacji.

Rekompilacja w Linux/FreeBSD

Do wykonania kompilacji w systemach Linux/FreeBSD niezbędny jest kompilator GNU C/C++ (autor wykorzystywał wersję 2.95) i program sterujący kompilacją GNU Make. Należy wejść do odpowiedniego podkatalogu platformy (np. `platform/freebsd`) i wydać polecenie `make` (w niektórych systemach `gmake`).

Po kompilacji pliki są automatycznie przenoszone do katalogu `cgi-bin` w katalogu głównym projektu – dlatego tak istotne jest istnienie tam przynajmniej dowiązania symbolicznego do rzeczywistej lokalizacji. Przełączanie profilu *Debug/Release* jest dokonywane w nagłówku pliku `Makefile` przy pomocy standardowych opcji kompilatora GNU. Nie powinno to sprawić problemu każdemu, kto uprzednio korzystał już z tego znakomitego narzędzia. Zaleca się kompilację wersji wdrożeniowej w trybie bez informacji debuggera i z pełną optymalizacją. Po zakończeniu kompilacji można usunąć pliki tymczasowe wydając polecenie `make clean`.

Podobna kompilacja nie powinna stanowić większego problemu we wszystkich systemach operacyjnych wyposażonych w narzędzia programisyczne GNU. Zaleca się korzystanie z `Makefile` platformy FreeBSD jako pliku bazowego do generowania implementacji dla nowych systemów.

Uwaga!: Jeżeli w projekcie wykonywano zmianę przy użyciu narzędzi systemu Windows, to należy zadbać, by w plikach źródłowych znalazły się krótkie znaki końca linii takie, jakie obowiązują w systemie Unix. W pewnych specyficznych sytuacjach podwójne znaki końca wiersza nie są akceptowane przez kompilatory GNU.

3.3 Opis implementacji

W poprzednich częściach rozdziału trzeciego przedstawiono zasady instalacji, użytkowania i eksploatacji systemu *RENEG* ujęte w postać odpowiednich instrukcji na wzór instrukcji systemów komercyjnych. Treścią kolejnej części staje się budowa wewnętrzna systemu z uwzględnieniem wielości jej aspektów: zastosowane technologie, architektura systemu, przyjęte paradygmaty i metody, podział na klasy i w końcu własne rozszerzenia, modyfikacje i uwagi dotyczące implementacji poszczególnych algorytmów.

3.3.1 Metody i paradygmaty

Programowanie rekurencyjne

Wydawać się może zabawne, że w opracowaniu poświęconym sieciom przyczynowo-skutkowym, poświęca się osobny rozdział pojęciu tak trywialnemu jak rekurencja i wyróżnia się je jako jedną z trzech najistotniejszych metod i paradygmatów projektu. Podczas realizacji algorytmów okazało się, że rekurencja jest narzędziem podstawowym nie tylko ze względu na swoją prostotę, ale także ze względu na zakres zadań, które realizuje.

Na potrzeby systemu zaimplementowano kilkanaście funkcji rekurencyjnych, czasem parami rekurencyjnych, wykonujących rozmaite zadania: proste przeszukiwanie grafów wgłąb, poszukiwanie ścieżek, zbiorów optymalnych – mówiąc ogólniej: poszukujących rozwiązań rozmaitych zagadnień plecakowych. Jest rekurencja po prostu podstawowym narzędziem przeszukującym daną przestrzeń a także techniką znakomicie pasującą do specyfiki zagadnień grafowych. Jeśli dodatkowo uwzględnić rekurencyjne algorytmy bibilioteki STL wykorzystywane w projekcie trudno umniejszyć znaczenie tej metody programowania.

W swoim doświadczeniu programistycznym spotkałem się z dość dużym oporem środowiska do stosowania rekurencji. Ma ona opinię narzędzia trudnego w stosowaniu i bardzo zasobochłonnego. Stwierdzenie o rzekomej trudności rekurencji uznaję za z gruntu błędne. Moim zdaniem jest ona właśnie jednym z najbardziej przejrzystych sposobów zapisu algorytmów. Podejrzewam, że niechęć do rekurencji wiąże się też z niską popularnością programowania funkcjonalnego, w którym jest ona podstawową techniką.

Nadszedł już czas, aby obalić też przekonanie o zasobochłonności rekurencji. Pochodzi ono jeszcze z czasów 8 i 16-bitowych systemów operacyjnych o bardzo ograniczonej wielkości stosu procesora. W obecnych na rynku od kilku lat systemach 32-bitowych długość stosu może być wystarczająco duża dla wielu zadań rekurencyjnych. Podczas realizacji systemu *RENEG* ani razu nie napotkano na problemy z długością stosu.

Programowanie obiektowe

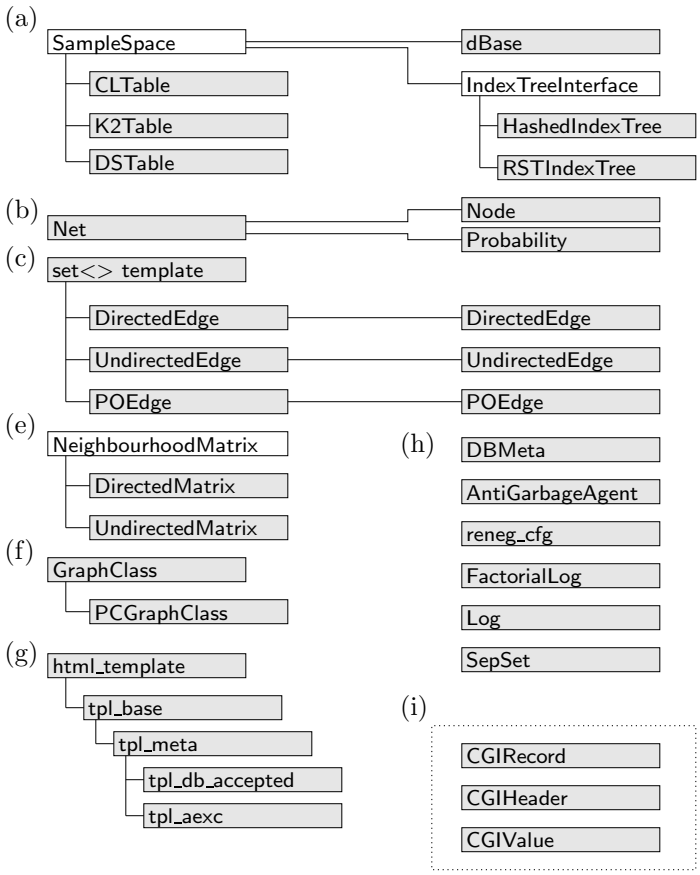
Powiedzieć, że system *RENEG* został zaimplementowany w oparciu o paradygmat obiektowy to zdecydowanie za wiele, ale powiedzieć, że programowanie obiektowe ma dla projektu niewielkie znaczenie to odebrać mu fundament organizacji. Implementacji dokonano bowiem stosując hybrydę programowania proceduralnego i obiektowego. Większość obiektów, na których operują algorytmy zamknięto bowiem w klasy języka C++, podczas, gdy same algorytmy są programami z gruntu proceduralnymi i tylko wykorzystają obiekty języka C++. Innymi słowy: zobietkywizowano w projekcie wszystko poza głównym zadaniem – nie ma obiektu algorytmu.

Podejście takie, choć stosowane nieświadomie przez bardzo wielu programistów, może spotkać się z krytyką purystów obiektowych. Na swoją obronę autor podaje, że opisy wszystkich algorytmów jakie otrzymał miały bardzo prosty proceduralny charakter a operowały często na obiektach wysokiego poziomu (np. ścieżka w grafie). Siłą rzeczy, dążąc do zachowania zawsze pożądanej bliskości opisu i implementacji, zaprojektowano zbiór obiektów a sam przebieg algorytmów zrealizowano proceduralnie.

Rysunek 3.15 uzmysławia znaczenie technologii obiektowych w systemie *RENEG*. Wystarczy zwrócić uwagi na ilość występujących w nim hierarchii i zauważyć, że w prawie wszystkich wykorzystywane są funkcje wirtualne i polimorfizm. Na rysunku zaznaczono tylko wybrane klasy. Pominięto licznie wykorzystywane klasy-konkretyzacje szablonów STL, klasy wyjątków i inne pomniejsze klasy. Klasy przedstawione na rysunku podzielono na grupy:

(a) reprezentacja próby danych na wysokim (próba statystyczna, drzewo wystąpień) i niskim poziomie (plik w standardzie dBase)

(b) reprezentacja pliku w formacie BENIOFSKI



Rysunek 3.15: Podział obiektów projektu na klasy i hierarchie klas. Na biało zaznaczono klasy abstrakcyjne.

(c)(f) reprezentacja grafów (skierowany, nieskierowanych i częściowo nieskierowanych).

(e) Macierze sąsiedztwa.

(g) Interpretery wzorców HTML (zob. 3.3.2).

(i) Interpretery wejścia CGI (metoda POST).

(h) Inne obiekty niezwiązane hierarchią.

Hierarchiczność i polimorfizm pozwala rozwijać projekt systemu wraz z zaawansowaniem prac. Przykładowo precyzyjne określenie interfejsu `SampleSpace` pozwoliło z czasem rozszerzać jego funkcjonalność o obszary obliczeniowe wymagane w różnych algorytmach. Skolei wprowadzenie nowej implementacji interfejsu `IndexTree` pozwoliło osiągnąć wzrost wydajności **wszystkich algorytmów** bez bezpośredniej ingerencji w ich kod.

Programowanie przenośne

Kolejną, obok obiektowości, ideą przyświecającą tworzeniu implementacji było programowanie przenośne. W czasach, gdy programowanie przenośne kojarzy się głównie z językiem Java, wielu zapomaniało już, że niegdyś ta sama myśl wiodła projektantów języka C, a potem C++. Do dziś istnieje wiele aplikacji przenośnych napisanych w tych językach, należących niestety głównie do świata systemu Unix i pochodnych. Tylko niektóre z nich mogą poszczycić się wydaniem dla systemu Windows. Wynika to głównie z faktu, że nie zaproponowano jeszcze dojrzałych rozwiązań przemysłowych pozwalających uogólnić implementację nowoczesnego interfejsu użytkownika tak, by mogła zostać skompilowana na różnych platformach. Współczesne programy zwykle cechuje bardzo silny związek logiki i interfejsu, który skolei jeszcze silniej związany jest z graficznie sterowanym systemem operacyjnym.

System *RENEG* znakomicie nadawał się do zaprogramowania przenośnego. Wynika to głównie z bardzo niewielkiego związku aplikacji CGI z systemem operacyjnym: korzystają one tylko ze strumieni wejścia i wyjścia oraz bardzo nielicznych funkcji systemowych (np. do pomiaru czasu). Podjęto zatem decyzję o równoległym testowaniu i rozwijaniu implementacji w środowisku uniksowym i w systemie Windows. Docelowo projekt miał zafunkcjonować (i tak się stało) w systemach Windows, Linux i FreeBSD.

Dla każdej platformy docelowej stworzony oddzielny plik `Makefile`[6] oraz zestaw ustawień kompilacji warunkowych zgromadzony w nagłówku `profile.h`. Różnice dały się we znaki głównie na linii Windows–Linux i dotyczyły między innymi różnego traktowania znaków końca linii, różnych zachowań biblioteki `iostream` w sytuacjach awaryjnych, różnych nazw plików nagłówkowych i różnych implementacji STL. Ten ostatni problem rozwiązano kompilując w Microsoft Visual Studio implementację STL pochodząca z firmy Silicon Graphics dostępną także w systemie Unix. Przeniesienie projektu z Linux na FreeBSD nie wymagało praktycznie żadnej pracy pogramistycznej.

Decyzja, że projekt ma działać zarówno na serwerze opartym o Windows jak i odmianę Unixa, pociągała za sobą kolejną rozsądną decyzję o uniezależnieniu się od konkretnego serwera WWW. Tym samym niemożliwym stało się korzystanie z rozmaitych rozszerzeń proponowanych przez producentów, którzy na wiele sposobów próbują narzucić rynkowi swój sposób tworzenia dynamicznego HTML. Aby uniknąć trudności w przenoszeniu projektu na różne platformy, zaimplementowano wspomniany już własny mechanizm rozwijania wzorców HTML, któremu poświęcono nieco więcej uwagi w kolejnym paragrafie. Jest on oparty o podstawowe skrypty CGI dostępne na praktycznie każdym serwerze http.

Ze względu na fakt, że dostępne serwery WWW dla windows opierają uruchamianie skryptów CGI na standardowych zasadach uruchamiania programów w systemie Windows, niezbędnym okazało się nadanie tym skryptom rozszerzenień `.exe`. Zachowanie tej zasady w implementacji unixowej pozwoliło zminimalizować ilość zmian w stronach html wykonywaną przy przenoszeniu instalacji na inne platformy, czy serwery. Oczywiście programy z rozszerzeniem `.exe` wyglądają w Unix co najmniej egzotycznie.

W efekcie powstał system otwarty, który kompiluje się na platformach Microsoft Windows (Microsoft Visual C++), Linux (GNU C/C++) i FreeBSD (GNU C/C++) oraz działa w oparciu o serwery: Website, Personal Web Server i Apache. Wymienione nazwy systemów i serwerów odnoszą się do rzeczywistych instalacji testowych wykonanych przez autora. Nic nie stoi jednak na przeszkodzie, by na obecnym poziomie przenośności, dokonać kompilacji na innych systemach (zwłaszcza uniksowych) i instalacji na innych serwerach http.

[6] W przypadku Microsoft Visual Studio był to zbiór projektów (*Work Space*).

3.3.2 Technologie

HTML i JavaScript

Strony systemu utworzono przy użyciu języka HTML i krótkich skryptów JavaScript 1.2. Starano się stworzyć dające poczucie pewności wrażenie, że formatki pochodzą ze zwykłych aplikacji Windows. Są to jednak jedynie zagnieżdżane tabele HTML, odpowiednio pokolorowane.

Na potrzeby projektu powstała także biblioteka `html_template` obsługująca rodzaj szablonów HTML. Szablony to sparametryzowane pliki HTML, w których zawarto specjalne komentarze rozwijane w skryptach CGI do odpowiednich wartości. Przykładowo zapis

```
<B>Imię i nazwisko: </B><!--VAR user-->
```

jest zamieniany przez program `status.exe` na zapis

```
<B>Imię i nazwisko: </B>Andrzej Wąsowski
```

podczas zwracania opisu parametrów zlecenia generacji.[7]

Biblioteka jest zbudowana w technologii obiektowej i silnie korzysta z własności polimorfizmu i dziedziczenia. Programowanie szablonów polega na definiowaniu klas obiektów będących w stanie zinterpretować mniejsze lub większe zbiory szablonów. Czas pracy nad implementacją klas obsługujących nowe szablony jest istotnie skrócony dzięki możliwości powtórnego wykorzystania kodu szablonów wcześniej napisanych. Programy CGI nie zawierają w sobie żadnych nadmiarowej logiki służącej interpretacji szablonów. Jest on gromadzona tylko w implementacjach klas, co umożliwia ich późniejsze wykorzystanie.

Jedną z najważniejszych zalet biblioteki jest fakt, że kod html nie jest zawarty w binariach programów CGI. Przechowywany jest w specjalnych plikach, różniących się od zwykłych plików html tylko tym, że zawierają komentarze z ukrytymi polami do rozwinięcia. Takie pliki html mogą być poddawane edycji przez narzędzie autorskie html, co jest nieporównanie wygodniejsze niż zmiana kodu ukrytego w wywołaniach funkcji `printf` programu w C. Dodatkową zaletą jest możliwość modyfikacji stron (np. w celu przetłumaczenia na inny język) bez rekompilacji kodu.

[7] przy założeniu, że zleceniodawca podał nazwisko Andrzej Wąsowski.

Serwery http różnych producentów oferują różnego rodzaju rozszerzenia pozwalające uzyskać podobny efekt bez tworzenia specjalnej biblioteki. Jednak żadne z tych rozszerzeń nie jest przenośne. Zaprojektowanie własnej bilioteki umożliwiło uruchamianie systemu w różnych środowiskach.

Common Gateway Interface

CGI (ang. *Common Gateway Interface*) należy do najdłużej stosowanych technologii internetowych. Jej działania opiera się o mechanizm zdalnego uruchamianie programów przez użytkowników korzystających z klienta protokołu http i przekierowywania strumieni wyjściowych na wyjście przeglądarki. Schematyczny przebieg sesji CGI wygląda jak następuje:

- Użytkownik otwiera stronę HTML zawierającą wywołania CGI.
- Uruchamia skrypt CGI na zdalnym komputerze, wybierając ze strony odpowiedni przycisk lub hyperdowiązanie URL.
- Skrypt otrzymuje informację o wywołaniu na standardowe wejście i/lub poprzez zmienne widoczne w środowisku wywołania.
- Przetworzywszy dane wejściowe, generuje zwykle pewne wyjście, przesyłane następnie do klienta, gdzie jest interpretowane i (najczęściej) wyświetlane na ekranie.

Wprowadzenie technologii CGI zrewolucjonizowało rynek serwisów WWW. Ze względu na integrację technologii z formularzami języka HTML i możliwość dynamicznego tworzenia wynikowych stron przez skrypty, CGI stała się najpowszechniej używanym narzędziem do budowania aplikacji internetowych, zdecydowanie prześcigając w popularności inne rozwiązania (np. *Java, parsed HTML*, technologie związane z bazami danych). Na bazie CGI zbudowano także wiele nowszych technologii wyższego poziomu; bibliotek, generatorów, inerfejsów do baz danych itp. Do popularności CGI przyczynił się także rozwój języka skryptowego *Perl* i szeroka dostępność bardzo wielu gotowych skrytów realizujących podstawowe zadania (liczniki odwiedzin, zegarki, datowniki, książki gości, czy wreszcie tzw. banery reklamowe, etc.).

Jako bardzo stara technologia, CGI niestety posiada kilka poważnych wad, które raczej niespodziewanie nie ograniczają jej popularności. Jest to

przede wszystkim technologia bardzo niewygodna w programowaniu zwłaszcza w tradycyjnych, kompilowanych językach programowania. Bardzo niewygodne jest testowanie aplikacji CGI a debugowanie jest wręcz niemożliwe. Dodatkowo strony podawane przez CGI generują o wiele większe obciążenie sieci niż tradycyjne statyczne strony HTML. Wynika to z faktu, że pobierając dynamiczną stronę(wyniki działania skryptu) wielokrotnie, przeglądarka ma bardzo ograniczone możliwości korzystania z wcześniej zbuforowanych informacji. Zazwyczaj większość informacji na stronie musi być ponownie w całości załadowana z serwera, podczas gdy w przypadku strony statycznej zostałyby odczytane z lokalnego bufora dyskowego. Na tej samej zasadzie technologia CGI ogranicza skuteczność serwerów proxy.

Wśród następców CGI wymienia się obecnie bardzo często podobną technologię nazywaną PHP. Od samego CGI odróżnia ją lepsza integracja z serwerem WWW, a co za tym idzie łatwiejsze programowanie.

Aplikacje CGI mogą być programowane niemal we wszystkich językach programowania obsługujących standardowe strumienie we/wy. Nazywane są one skryptami niezależnie od tego, czy są to rzeczywiście wsadowe skrypty, czy „prawdziwe" kompilowane programy w języku C/C++.

W systemie *RENEG* jest w sumie 5 skryptów CGI, z których dwa (`bayesin` i `bayesout`) są rzeczywiście skryptami w języku *Perl*. Pozostałe są zwykłymi programami binarnymi napisanymi w języku C++ i skompilowanymi. Na diagramie 3.16 przedstawiono funkcje poszczególnych skrytpów. Schemat pokazuje, że tylko programy odpowiedzialne za interakcję z użytkownikiem (przyjmowanie danych, parametrów i wyświetlanie wyników) korzystają z technologii CGI. Programy, które zajmują się właściwym uczeniem sieci są wywoływane przez `prep.exe` i działają jak normalne procesy systemu operacyjnego. Trzy skrypty CGI (a zwłaszcza program `prep.exe`) modyfikują swoje działanie w zależności od wskazanego przez użytkownika algorytmu. Skolei niektóre programy uczące realizują więcej niż jeden algorytm – podejście takie było wyjątkowo korzystne dla algorytmów podobnych do siebie (Chow-Liu i Pearl, PC i PC*, oraz FCI i FCI*).

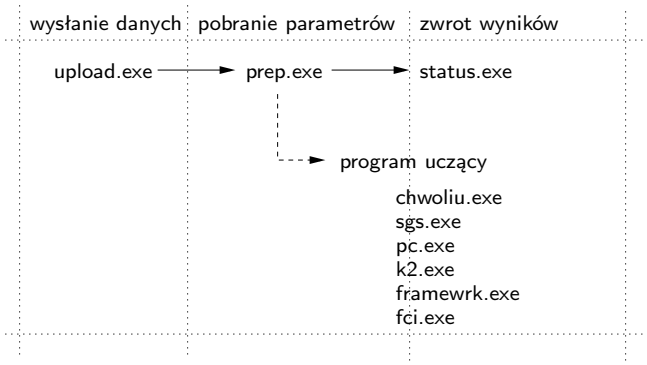
Wywoływanie programów uczących jako zewnętrznych procesów było konieczne, aby uniknąć zabijania ich przez niektóre serwery WWW ograniczające czas działania programów CGI. Podejście to nie rozwiązuje problemu całkowicie, ale daje poprawne rezultaty między innymi dla wszystkcih serwerów Apache i Personal Web Server.

Dodatkowo należało też rozwiązać problem oczekiwania przeglądarki na wyjście CGI. Każdorazowo po uruchomieniu skryptu przeglądarka klienta czeka na wyniki jego pracy dopóki skrypt nie zakończy działania. W przypadku programów uczących czas działania może być jednak liczony w godzinach, podczas gdy większość przeglądarek przerywa oczekiwanie po kilku minutach z komuniaktem o błędzie. Opisane powyżej wywoływanie oddzielnego procesu uczącego z programu CGI nie rozwiązuje problemu, gdyż proces potomny przejmuje wyjście programu CGI i przeglądarka traktuje skrypt jako działający nawet jeśli w rzeczywistości funkcjonują tylko jego potomkowie. Problem ten rozwiązano wypisując w programie CGI wyjście w formacie HTML z informacją o rozpoczęciu obliczeń i *następnie* zamykając wszystkie strumienie wejściowe i wyjściowe programu (w języku C wystarczyło zamknąć strumienie `stderr`, `stdout` i `stdin`). Zamknięcie strumieni zrywa jedyne połączenie programu z serwerem WWW, który informuje przeglądarkę klienta, że dalsze wyjście już nie nastąpi. Przeglądarka uznaje konsekwetnie, że skrypt zakończył działanie i kończy oczekiwanie wyświetlając uzyskaną stronę HTML.

Programy CGI systemu RENEG korzystają z metody POST przekazywania parametrów przez przeglądarkę. Metoda POST dostarcza skryptowi wartości parametrów w postaci rekordów HTTP na standardowe wejście. Wartości te nie są w żaden sposób kodowane, w odróżnieniu do popularnej metody GET. Metoda POST jest też jedyną metodą, która umożliwia przesyłanie na serwer dużych plików. Analiza wejścia jest wykonywana za pomocą dedykowanej biblioteki `mycgi`, zaimplementowanej specjalnie na potrzeby tego projektu.

3.3.3 Implementacje algorytmów

W niniejszej pracy przyjęto założenie, że wszystkie implementowane pocedury generacji powinny kończyć swoje działanie zwróceniem w pełni zorientowanej struktury sieci Bayesa wraz z tablicami prawdopodobieństw warunkowych dla węzłów. Problem generacji prawdopodobieństw warunkowych nie jest rozpatrywany przez algorytmy same w sobie i rozważam go w punkcie 3.3.3. Skolei problem ustalania kierunków zależności między atrybutami jest rozwiązywany różnie w zależności od algorytmu. Algorytmy nie rozwiązują tego problemu lub rozwiązują go tylko częściowo. Zastosowano rozszerzenia prowadzące w rezultacie do całkowicie zorientowanego grafu. Rozszerzenia opisano w rozdziałach dotyczących poszczególnych algorytmów.



Rysunek 3.16: Diagram kolejności wywołań skryptów.

Nie było możliwe przedstawienie implementacji w szczegółach ze względu na objętość pracy. Podaje się tu tylko uwagi i usprawnienia autora w stosunku do opisów teoretycznych z rozdziału 2.

Technicznie rzecz biorąc algorytmy są modułami języka C++, złożonymi z funkcji globalnych operujących na obiektach wysokiego poziomu reprezentujących grafy, krawędzie, drzewa, ścieżki itp. Różnego rodzaju obiekty grafowe proponują różnego rodzaju reprezentacje. Często algorytm korzysta jednocześnie z reprezentacji macierzowej (macierz sąsiedztwa) i listowej (zbiór krawędzi), aby zyskać na efektywności. Niektóre algorytmy zaczynają z jedną reprezentacją a kończą z drugą, aby ograniczyć zużycie pamięci. Dotyczy to zwłaszcza algorytmów rozpoczynających pracę od grafu pełnego i usuwających krawędzie.

Obliczanie tablic prawdopodobieństw

Algorytmy uczące zajmują się jedynie budowaniem struktury sieci, nie określają jednak ilościowego charakteru przedstawionych zależności, czyli tablic prawdopodobieństw warunkowych będących częścią opisu sieci w formacie BENIOFSKI. Dlatego po ustaleniu struktury sieci przez którykolwiek algorytm stosowany jest jeszcze mechanizm obliczania tablic prawdopodobieństw.

Zaproponowano tu zliczanie wystąpień instantyzacji grupy zmiennych w próbie w postaci struktury drzewiastej o widocznych dwu poziomach zagłębienia. W tym celu zdefiniowano interfejs `IndexTreeInterface`. Klasy tego

interfejsu przechowują i udostępniają bazę danych pozwalając zadawać zapytania o ilość wystąpień danej instantyzacji z podziałem na przedrostek i przyrostek.

Przykładowo algorytm może zapytać o ilość wystąpień kolejnych instantyzacji kolumn drugiej, trzeciej i czwartej. Przestrzeń próbkowania jest skanowana metodami interfejsu `SampleSpace` a jako wynik udostępniana jest klasa pochodna `IndexTreeInterface` umożliwiająca iterowanie po kolejnych instantyzacjach kolumn, informując o ilości wystąpień danej instantyzacji, ilości różnych instantyzacji zadanych kolumn w grupie i ilości zdarzeń w próbie. W pamięci przechowuje się tylko informacje o wartościowaniach mających niezerowe ilości wystąpień. Są one dodatkowo skompresowane przez zastosowanie struktury drzewiastej (sklejanie wspólnych podciągów początkowych).

`IndexTreeInterface` umożliwia też zadawanie pytań o warunkowe ilości wystąpień i inne szczegółowe informacje niezbędne niektórym algorytmom. Można więc zadać pytanie o instantyzacje kolumn czwartej i piątej pod warunkiem kolumny pierwszej. W źródłach projektu stosowane jest tu oznaczenie odwrotne od przjętego w teorii prawdopodobieństwa $(1|4,5)$ wynikające z kolejności analizowania instantyzacji przez klasy `IndexTreeInterface`. W przypadku takiego zapytania algorytm pozwala iterować instantyzacje w próbie na dwu poziomach zagnieżdżeń. Na ogólnym poziomie iteracji podawane są wartościowania przedrostka (w przykładzie jest nim kolumna pierwsza) wraz z ilością różnych instantyzacji prefixu i ilością wystąpień jego poszczególnych instantyzacji. Dla każdego wartościowania przedrostka można iterować wartościowania przyrostka (kolumny czwarta i piąta) poznając wartości przyrostka, ilość wystąpień poszczególnych wartościowań i ilość różnych wartościowań.

Tak szeroki zakres odpowiedzi dawanej przez obiekty klasy `IndexTreeInterface` pozwala obliczać prawdopodobieństwa warunkowe, a także różne inne cechy statystyczne potrzebne w algorytmach (np. statystykę χ^2). Wyróżnienie tego interfejsu bardzo uprościło implementację wszyskich algorytmów.

`IndexTreeInterface` nie odpowiada za zbudowania struktury wystąpień wartościowań na podstawie próby. Interfejs ten organizuje tylko dostarczone mu dane udostępniając metody do dodawania i zadawania zapytań. Baza danych jest czytana przez klasy reprezentujące przestrzeń próbkowania. Za-

równo baza danych jak i drzewo wystąpień są typami o niższym poziomie abstrakcji i wydaje się słusznym zamknięcie bezpośredniego dostępu do nich w klasie wyższego poziomu – `SampleSpace`.

Baza danych jest skanowana każdorazowo po określeniu kolumn uczestniczących w zapytaniu. Takie podejście ogranicza nieco ilość pamięci potrzebnej na przechowywanie struktury. Złożoność pamięciowa zależy tu bezpośrednio od ilości kolumn uczestniczących w zapytaniu. Dlatego zadecydowano o nie przechowywaniu danych o wartościowaniach wszystkich atrybutów w pamięci. Oznaczałoby to praktycznie przechowywanie w pamięci całej tabeli. Zamiast tego przechowywuje się jedynie watościowania kolumn których dotyczy zapytania. Złożoność pamięciowa zostaje istotnie ograniczona, przy jednoczesnym wzroście szybkości odpowiedzi na pytania.

Podczas budowania implementacji interfejsu `IndexTreeInterface` kierowano się dwiema zasadami: szybkości i ograniczoności pamięci. Najpierw powstała implementacja `HashedIndexTree` składająca się z talicy rozdzielczej (ang. *hash table*) na poziomie prefixu i takich samych tablic dla przyrostków na niższym poziomie. Wartościowania zakodowano jako ciągi znaków i stosowano standardowe szablony STL z dostępną w bibliotece funkcją rozdzielczą.

Tablica hashująca wydaje się być dobrą strukturą do przechowywania różnych ciągów znaków, które muszą być często wyszukiwane. Po analizie zadawanych zapytań okazało się jednak, że mają one zawsze charakter czysto iteracyjny i nie wymagają w związku z tym indeksowania funkcją rozdzielczą (zbiór wyników jest zawsze przeglądany liniowo jak lista). Wówczas zaproponowano całkowicie własną strukturę, w której tablice zastąpiono drzewami pozycyjnymi podobnymi nieco do drzew wyszukiwania pozycyjnego (klasa `RSTIndexTree`). Drzewa te dawały możliwość iterowania po kolejnych elementach przez proste przestawianie wskaźników, co jest czynnością bardzo szybką. W efekcie po wymienieniu klasy `HashedIndexTree` na `RSTIndexTree` uzyskano kilkukrotny wzrost szybkości działania wszystkich algorytmów, bez dokonywania w nich jakichkolwiek zmian. Było to możliwe, gdyż algorytmy operują na referencjach do abstrakcyjnych obiektów `IndexTreeInterface`.

Algorytm Chow-Liu i Algorytm Pearla

Algorytm Chow-Liu i algorytm Pearla zostały zaimplementowane jako jeden program CGI ze względu na swoje podobieństwo. Pierwsza część obu algorytmów jest identyczna. Budowane jest minimalne drzewo rozpinające

z odległością Kullback-Leiblera jako funkcją wagową. W powszechnym użyciu są dwa algorytmy rozwiązywania tego zadania: algorytm Prima-Dijkstry i algorytm Kruskala. Algorytm Prima-Dijkstry cechuje nieznacznie większa wydajność dla sieci gęstych o niedużych ilościach wierzchołków (rzędu 100) niż nieco częściej stosowany algorytm Kruskala (zob. [19]). Zważywszy, że liczba atrybutów w próbie zwykle nie przekracza 100, a graf, w którym algorytm poszukuje drzewa rozpinającego, jest grafem pełnym zdecydowano o wykorzystaniu algorytmu Prima-Dijkstry.

Dodatkowy parametr przekazywany do skryptu z przeglądarki określa zasadę działania drugiej części algorytmu: czy zastosować orientowanie drzewiaste, czy metodę podaną przez Pearla. W pierwszym wypadku krawędzie są orientowane tak, aby utworzyć drzewo skierowane o korzeniu w wierzchołku wskazanym przez użytkownika (Chow i Liu nie sprecyzowali jak zorientować krawędzie). Jeżeli użytkownik wybrał wersję Pearla, stosowane są reguły orientacji określone w paragrafie 2.3.2, w części teoretycznej pracy.

Algorytm SGS

W odróżnieniu od oryginalnego sformułowania algorytmu dokonano pewnego odwrócenia pierwszej fazy, które zaowocowało ograniczeniem zużycia pamięci. Zamiast rozpoczynać od grafu pełnego i usuwać krawędzie, system *RENEG* dodaje krawędzie do grafu pustego, stosując negację warunku odrzucenia. Nowa krawędź dodawana jest do grafu jeśli nie istnieje dla niej zbiór d-separujący. Zwraca się uwagę, że nie jest to żadna zmiana w idei algorytmu, a jedynie równoważny, odwrócony sposób reprezentacji.

Testy d-separacji wykonywane są w oparciu o test niezależności χ^2 zgodnie z poniższym twierdzeniem.

Twierdzenie 9 ([18]) *Jeśli P jest dyskretnym rozkładem prawdopodobieństwa zgodnym z grafem zależności G to zmienne X_i i X_j są d-separowane przez zbiór wierzchołków $\mathbf{S}$ $\iff$ X_i i X_j są warunkowo niezależne względem $\mathbf{S}$ w rozkładzie P.*

Testy niezależności korzystają z wcześniej opisywanego mechanizmu `IndexTreeInterface`. Poszukiwanie zbioru d-separującego jest niezwykle kosztowne. Trzeba bowiem wykonać rzędu 2^n testów niezależności, gdzie n

jest licznością zbioru atrybutów $\mathbf{V}$. Do każdego testu niezbędne jest przeskanowanie całej bazy danych i dwukrotna iteracja przez drzewo wystąpień. Co gorsza taki test musi być wykonany dla każdej pary wierzchołków.

Otrzymujemy więc $O(\frac{1}{2}2^n n^2)$ testów niezależności (przypadek pesymistyczny), z których każdy intensywnie analizuje bazę danych i drzewo wystąpień. To właśnie nieprawdopodobnie niska efektywność tego algorytmu zaowocowała pomysłem alternatywnej implementacji interfejsu `IndexTreeInterface`.

Ponieważ czas potrzebny do iteracji przez próbę i przez drzewo wystąpień, a także niezbędna do tego ilość pamięci zależą wykładniczo od rozmiaru potencjalnego zbioru d-separującego wprowadzono własne usprawnienie poszukując zbiorów d-separujących o rosnących wielkościach. Najpierw testowane są singletony, następnie zbiory dwuelementowe, itd. Intuicja za takim postepowaniem jest następująca, jeśli $\mathbf{S}$ d-separuje zmienne X_i i X_j, to d-separuje je także dowolny zbior $\mathbf{S}' \in \mathbf{S}$. Jednak wykonanie testu dla $\mathbf{S}$ jest tańsze niż dla $\mathbf{S}'$. Dlatego poszukiwane są minimalne zbiory d-separujące.

Test χ^2 dla niezależności dwu zmiennych zaimplementowano na podstawie formalnego wprowadznia podanego w [10], adoptując procedurą testu zgodności χ^2 w języku Fortran podaną w [15].

Test χ^2 wymaga, aby liczność każdego wartościowania była większa niż 5. W przypadku, gdy próba nie spełnia takiego warunku wartościowanie jest pomijane w teście. Procedura traktuje ten przypadek jak w sytuacji gdy, nie ma podstaw do odrzucenia hipotezy. W efekcie zbyt mało liczna baza danych, może dać odpowiedź o niezależności (czyli o d-separacji) dla zbyt wielu krawędzi, a co za tym idzie graf wynikowy będzie gęsty.

Algorytm SGS nie zwraca jako wyniku struktury sieci Bayesa, a częściowo zorientowany graf zależności reprezentujący całą klasę sieci przyczynowo-skutkowych. Zastosowano więc algorytm *pog-to-dag*, opisany w paragrafie 2.3.8 jako część algorytmu ramowego. Algorytm ten stosując jedynie szybkie operacje na grafach częściowo zorientowanych próbuje możliwie zgodnie dokonać orientacji niezorientowanych krawędzi. Unika przy tym powstawania cykli i kierowania krawędzi ku sobie[8].

W pewnych specyficznych przypadkach algorytmy mogą jednak zwracać

[8]Krawędzie są skierowane ku sobie są orientowane przez większość algorytmów. Często stosuje się więc założenie, że skoro algorytm nie zorientował krawędzi ku sobie to, o ile to możliwe, powinny być one zorientowane inaczej.

struktury zawierające cykle, lub struktury, których nie da się zorientować tak, aby cykli nie zawierały. W takich sytuacjach algorytm informuje o błędzie i sieć wynikowa nie jest generowana. Błędy wynikają zazwyczaj ze zbyt małej liczności próby.

Jednym z pomysłów na zaradzenie takiej sytuacji mogłoby być zintegrowanie algorytmu *pog-to-dag* z próbą wejściową. Można by wówczas przeorientowywać pewne krawędzie w cyklach, które są ze sobą możliwie mało związane (z minimalna stratą dopadsowania). Rozwiązanie takie nie jest na pewno uzasadnione statystycznie, ale może prowadzić do niezłych wyników, w sytuacji, gdy nie da się osiągnąć lepszego zbioru przypadków uczących.

Rodzina algorytmów: PC-1, PC-2, PC-3, PC-1*, PC-2*, PC-3*

Algorytmy serii PC cechuje wysokie podobieństwo. Z tego względu zostały zaimplementowane w jednym programie `pc.exe` a rodzaj strategii dobierania krawędzi i sposób określania zbioru kandydatów do zbioru d-separującego podawane są jako parametry dodatkowe skryptu.

Implementacja algorytmów z tej grupy jest w zasadzie dość wierna opisowi podanemu przez autorów poza dwoma wyjątkami. Po pierwsze wbrew sugestii autorów w algorytmach PC* nie przechowuje się wszystkich nieskierowanych ścieżek istniejących w grafie w danej chwili. Uznano, że postępowanie takie nie może być akceptowalne ze względu na ograniczenia ilości dostępnej pamięci. Dodatkowo przy występowaniu ogromnej liczy ścieżek czas dostępu do informacji o połączeniach z pewnością przekroczy czas sprawdzenia, czy ścieżka rzeczywiście istnieje w bieżącej strukturze grafu. Dlatego też takie rozwiązanie przyjęto. Każdorazowo przed przystąpieniem do poszukiwania zbioru d-separującego dwa wierzchołki ogranicza się zbiór kandydatów do rodziców jednego z wierzchołków, sprawdzając, którzy z nich leżą na ścieżkach nieskierowanych pomiędzy badaną parą.

Testy d-separacji przeprowadzane są z uwzględnieniem usprawnień zaproponowanych w algorytmie SGS (poprzedni paragraf). Podobnie po zakończeniu działania stosowany jest algorytm *pog-to-dag*.

Algorytm K2

Rozważywszy równanie (2.8), idąc zresztą za sugestią autorów algorytmu, zaproponowano usprawnienie obliczeniowe algorytmu, Podane w równaniu (2.8) sformułowanie funkcji $g(i, \pi_i)$ jest niewygodne w implementacji ze względu na trudności w upraszczaniu bardzo rozwiniętych ułamków czy też w obliczaniu istniejących tam silni z potencjalnie dużych wartości. Dlatego w implementacji zawartej w systemie *RENEG* w miejscu funkcji g wykorzystano funkcję $\bar{g}$ będącą zlogarytmowaną postacią g:

$$\bar{g}(i, \pi_i) = \log g(i, \pi) = q_i L(r_i - 1) + \sum_{j=1}^{qi} \Big(\sum_{k=1}^{r_i} L(N_{ijk}) - L(N_{ij} + r_i - 1) \Big) \quad (3.1)$$

gdzie $L(i) = \sum_{n=1}^{i} \log n$. Taka postać funkcji zachowuje monotoniczność oryginału posiadając jednocześnie wartości o istotnie mniejszych modułach, a co za tym idzie nastręczających znacznie mniej trudności przy obliczaniu, gdyż potrzebne wartości funkcji L mieszczą się bez trudu w zakresach zmiennych podwójnej precyzji. Jeśli dalej zauważyć, że funkcja L zachowuje rekurencyjną własność silni: $L(i + 1) = L(i) + \log(i + 1)$, to okazuje się, że jej wartości można niskim kosztem stablicować unikając wielokrotnego obliczenia sum tych samych logarytmów, a co za tym idzie, obniżając złożoność obliczeniową algorytmu.

Jeśli przyjąć, że $r = \max_{i=1..n} r_i$, to ilość obliczanych logarytmów dziesiętnych w równaniu (3.1) nie przekracza $(m + r - 1)$. Obliczenie i stablicowanie wartości funkcji $L()$ nie wymaga więc więcej niż $O(m + r - 1)$ obliczeń logarytmów, co jest raczej niewielką liczbą przy ogólnej złożoności algorytmu szacowanej przez autorów na $O(mu^2n^2r)$.

W systemie *RENEG* zaimplementowano podaną w paragrafie 2.3.7 wersję algorytmu K2 zamieniając funkcję oceny g na jej zlogarytmowaną postać $\bar{g}$ ze stablicowanymi wartościami sum logorytmów. Parametry wejściowe algorytmu to próba statystyczna, porządek na atrybutach z próby oraz górne ograniczenie ilości rodziców dla jednego węzła. Użytkownik może określić porządek zmiennych przesuwając jedynie zmienne w domyślnym porządku (za domyślny przyjmuje się taki porządek jaki miały atrybuty w pliku dBase). Było to możliwe dzięki zastosowaniu dynamicznego rozwijania wzorców html i technologii JavaScript. Po załadowaniu bazy przez system budowana

jest strona zawierająca listę atrybutów (wczytanych z bazy). Dołączony do strony kod JavaScript pozwala przesuwać zmienne w porządku (testowano w przeglądarkach Internet Explorer 5.0 i Netscape Navigator 4.5).

Algorytm Ramowy

Algorytm ramowy jest realizowany przez program CGI o nazwie `framewrk.exe`. Jego implementacja nie odbiega zasadniczo od przedstawionej w części teoretycznej. Niezbędnym było tylko dookreślenie drobnych szczegółów niezdefiniowanych w oryginalnym opisie.

W drugim kroku algorytmu występuje polecenie zastosowania kroków 4-8 algorytmu *pog-from-data*. W kroku czwartym tego algorytmu korzysta się jednak ze zbiorów $\mathbf{Sepset}(\cdot,\cdot)$ obliczanych w krokach wcześniejszych. Uznano zatem, że zamiast sprawdzać przynależność wierzchołka X_j do zbioru $\mathbf{Sepset}(X_i, X_k)$ należy sprawdzić, przynależność X_j do zbioru d-separującegp X_i i X_k. Poszukuje się takiego zbioru i jeśli istnieje on to zapisuje się go w odpowiednim $\mathbf{Sepset}(\cdot,\cdot)$, aby nie powtarzać obliczeń w kolejnych próbach (test d-separacji jest najbardziej czasochłonną częścią algorytmu). Jeżeli X_j nie należy do żadnego zbioru d-separującego X_i, X_k, wówczas sam wierzchołek X_j jest zapisywany w zbiorze $\neg\mathbf{Sepset}(X_i, X_k)$, aby unikać powtarzania testu przy kolejnych próbach.

Zbiory $\mathbf{Sepset}(\cdot,\cdot)$ i zbiory przeciwne są przechowywae pomiędzy różnymi etapami pracy algorytmu, ponieważ ich zawartość nie zależy od budowanej struktury sieci, a jedynie od własności rozkładu próby $\mathcal{D}$. Nieustannie muszą być natomiast wycofywane zmiany na strukturach danych reprezentujących sieć. W systemie *RENEG* rozwiązano ten problem nieco inaczej niż opisano w części teoretycznej: nie wycofuje się zmian każdorazowo po obliczeniu funkcji dopasowania, a raczej zapamiętuje się stan grafu przed dokonaniem zmian i przywraca się go zawsze, gdy zachodzi potrzeba. Takie podejście jest prostsze do zaprogramowania i prawdopodobnie nieco szybsze (zwłaszcza w systemach z dobrym zarządzaniem pamięcią dynamiczną).

W każdym obiegu swojej pętli algorytm ramowy rozszerza rozwiązanie o dokładnie jedną nową krawędź (ewentualnie orientując dodatkowo jedną z krawędzi uprzednio niezorientowanych). W oryginalnym sformułowaniu dodawana krawędź jest krawędzią, której dodanie spowodowało najlepszy globalny wzrost dopasowania w porównaniu do wszystkich innych krawędzi jeszcze nie należących do sieci. Nie stawia się tu warunku, by sieć wynikowa była

grafem spójnym.

Przeprowadzono testy, czy wymuszenie na algorytmie, by począwszy od drugiej krawędzi poszukiwał kolejnych tylko spośród tych, które sąsiadują z grafem dotąd zbudowanym, spowoduje wzrost skuteczności lub wydajności. Nie stwierdzono znaczącej różnicy między wersją zmodyfikowaną i oryginalną. Ostatecznie zrezygnowano więc z tego rozszerzenia, zwracając szczególną uwagę na fakt, że nie jest ono w stanie wygenerować niespójnej sieci (sieci o kilku spójnych składowych).

Formalny opis algorytmu ramowego nie precyzuje też warunku stopu. W niniejszej implementacji algorytm przerywa swoją pracę, gdy dodanie żadnej krawędzi nie było legalne, lub względny wzrost dopasowania po dodaniu najlepszej krawędzi nie przekracza wartości podanej przez użytkownika jako parametr algorytmu.

Algorytmy FCI, FCI*

Zgodnie z tym co opisano w rozdziale 2.3.9 algorytmy FCI i FCI* są oparte na tej samej zasadzie, co algorytmy z grupy PC. Ponieważ pierwszy etap algorytmu FCI (kroki pierwszy i drugi) przebiegają dokładnie tak samo jak w algorytmie PC, możliwe jest zastosowanie optymalizacji FCI* na identycznej zasadzie jak w PC*. Znaczenie tej optymalizacji jest jednak jeszcze mniejsze niż w przypadku PC*. Wynika to z faktu, że na koszt obliczeń algorytmu FCI wpływają dodatkowe kroki nie mające swoich odpowiedników w PC i nie profitujące z optymalizacji PC*.

W istocie w systemie *RENEG* pierwsze dwa kroki algorytmu FCI wykonywane są przez fragmenty kodu z modułu `pcfamily.cpp` opracowanego wcześniej dla algorytmów PC i PC*. Fakt wykorzystania wspólnego kodu pozwolił skrócić nieco czas implementacji FCI, który i tak pozostaje najbardziej złożonym algorytmem spośród tutaj opracowanych. Dodatkowym zyskiem było automatyczne przeniesienie optymalizacji PC* oraz strategii heurystycznych PC-1, PC-2 i PC-3.

Implementacja algorytmu FCI wyjątkowo intensywnie korzysta z rekurencji. Dwie funkcje rekurencyjne (`PossibleDSEPDig()` i `DSepSetDig()`) poszukują wierzchołków d-separujących, dwie inne (`DDPDig1()` i `DDPDig2()`) poszukują ścieżek wyłączających. Wreszcie poszukiwanie niesprzecznej orientacji krawędzi w wynikowej sieci również odywa się reukrencyjnie. Pomijam w

tym wyliczeniu funkcje o mniejszym znaczeniu: poszukiwanie ścieżek, cykli, etc.

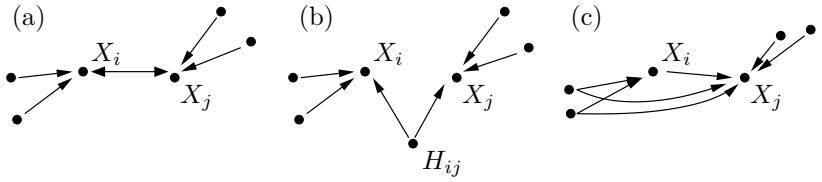
Funkcja rekurencyjna `PossibleDSEPDig()` znajduje zawartość zbioru **Possible-D-SEP**. W implementacji kierowano się zasadą wydzielania funkcjonalnych fragmentów kodu, tak aby nie zwiększać skomplikowania i tak już złożonego algorytmu. Niestety odbyło się to kosztem potencjalnego zmniejszenia szybkości. System najpierw znajduje cały zbiór potencjalnych kandydatów, a następnie szuka w nim podzbioru d-separującego. Wydaje się, że w niektórych przypadkach korzyść mogłaby przynieść lepsza integracja tych funkcji. Należałoby wówczas dynamicznie sprawdzać własności d-separacji dla tych elementów **Possible-D-SEP**, które zostały znalezione do danej chwili.

Operacja poszukiwania podzbioru d-separującego wymaga wykonywania czasochłonnych testów χ^2 na próbie $\mathcal{D}$ podczas, gdy szukanie zbioru kandydatów polega jedynie na przechodzeniu fragmentów grafu przechowywanego w pamięci operacyjnej. Dlatego też algorytm spędza większość czasu wewnątrz funkcji `DSepSet()` poszukującej podzbioru d-separującego `DSepSet()`. Złożoność tej funkcji jest proporcjonalna do silni liczności zbioru kandydatów. Zgodnie z zasadą PC działanie funkcji ogranicza się do sprawdzania w pierwszej kolejności podzbiorów najmniej licznych.

Uwzględniwszy powyższe dochodzimy do wniosku, że wprowadzenie dynamicznego poszukiwania podzbioru d-separującego jednocześnie ze znajdowaniem kolejnych kandydatów **Possible-D-SEP** nie może przynieść znacznej poprawy jakościowej algorytmu (np. zmniejszenie złożoności o rząd wielkości). Uzyskana oszczędność może polegać na ograniczonym zmniejszeniu współczynnika liniowego złożoności, co mogłoby okazać się uzasadnione w oprogramowaniu komercyjnym a wydaje się mieć o wiele mniejsze znaczenie w projekcie akademickim.

Algorytm FCI zwraca częściowo zorientowany graf reprezentujący strukturę zawierającą zmienne ukryte. Wszystkie podwójnie zorientowane krawędzie ($X_i \leftrightarrow X_j$) oznaczają, że ich wierzchołki mają wspólnego ukrytego rodzica. W systemie *RENEG* zastosowano następujący sposób konwersji uzyskanej struktury do sieci Bayesa:

1. Należy usunąć wszystkie podwójnie krawędzie $X_i \leftrightarrow X_j$ i zamiast nich wprowadzić nowe wierzchołki H_{ij} i krawędzie zorientowane $X_i \rightarrow H_{ij} \leftarrow X_j$ (rys. 3.17ab). Jednocześnie należy zapisać na tymczasowej liście



Rysunek 3.17: Wstawianie zmiennych ukrytych.

wszystkie usunięte krawędzie.

2. Zorientować pozostałe niezorientowane krawędzie stosując algorytm *pog-to-dag* (zob. s.23).

3. Usunąć wszystkie wierzchołki dodane w punkcie pierwszym wraz z krawędziami incydentnymi.

4. Do każdej krawędzi zapisanej na liście w punkcie pierwszym należy zastosować algorytm *edge-reversal* (rys. 3.17c). Kolejność wybierania krawędzi z listy i kierunek stosowania *edge-reversal* jest dobierany rekurencyjnie aż zostanie znaleziona globalna orientacja nie zawierająca cykli.

5. Do tekstowego opisu sieci w formacie BNIF dołaczany jest komentarz zawierający listę par wierzchołków utworzoną w punkcie pierwszym z odpowiednia adnotacją o możliwości występowania zmiennych ukrytych.

FCI to ostatni z zaimplementowanych algorytmów. Przedstawione powyżej założenia funkcjonalne, eksploatacyjne i implementacyjne stanowią razem szczegółowy opis zrealizowanego systemu generacji, który cechuje wykorzystanie nowowczesnych technologii, zróżnicowanych algorytmów, otwartość i przenośność. W kolejnym rozdziale opisano testy przeprowadzone po wykonaniu implementacji i przeprowadzono analizę jakościową aplikacji uczących. W końcu wyciągnięto wnioski bardziej ogólne i podsumowano całość projektu.

Rozdział 4

Eksperymenty

W rozdziale zawarto opis testów przeprowadzonych w systemie począwszy od przyjętych zasad przeprowadzania eksperymentu, przez testy wydajności platform programistycznych i systemowych po testy wydajności i skuteczności samych algorytmów uczących.

4.1 Przygotowanie i przebieg eksperymentu

4.1.1 Przygotowanie danych testowych

Testowanie systemu *RENEG* wymagało wygenerowania danych testowych z sieci o znanym rozkładzie. W tym celu powstały niezależne drobne programy-generatory korzystające z systemowego generatora liczb pseudolosowych[1]. Wyniki uzyskane za pomocą generatorów w formacie tekstowym zostały przetłumaczone na format dBase IV przy wykorzystaniu programów *Microsoft Excel* i *Bro*.

Zaprojektowano ponad 20 różnych sieci a dla każdej z nich wygenerowano zestaw prób o wielkościach 500, 1000, 1500, 5000 i 15000 zdarzeń. Próby generowano niezależnie (alternatywnie można obcinać dużą próbę do mniejszych zbiorów). Dla dużych sieci wygenerowano też większe próby złożone

[1]Korzystano tu z funkcji `rand()` języka C. Dokładne testy wymagałyby zapewne zastosowania bardziej zaawansowanego generatora pseudolosowego.

z 30 000, 60 000, 100 000 i 200 000 przypadków. Wszystkie grafy mają nazwy jednoliterowe a odpowiadające im próby mają nazwy złożone z litery oznaczającej graf i ilości zawieranych przez nie przypadków. Przykładowo `n5000.dbf` oznacza próbę z rozkładu o strukturze opisanej w grafie N złożoną z pięciu tysięcy przypadków.

Wszystkie zestawy testowe wraz z odpowiadającymi im strukturami sieci w plikach `.net` dołączono do projektu na jego stronie WWW (pozycja *Download* w menu głównym). Dodatkowo zamieszczono je także na płycie CD-ROM projektu (katalog `test/sample`).

4.1.2 Zasady przeprowadzania testów

Mając próby o znanym pochodzeniu, można już było oceniać wyniki uczenia. Testy przeprowadzano podając na wejścia poszczególnych algorytmów pliki wejściowe o kolejnych strukturach i rozmiarach oraz różne wartości parametrów wejściowych. Rejestrowano czas trwania obliczeń oraz jakość dopasowania uzyskanej sieci do oryginalnego rozkładu.

Ze względu na wykonywanie testów w środowisku wielozadaniowym, istnieje ryzyko zakłócenia pomiarów przez procesy zewnętrzne. Z tego względu testy prowadzono na wyizolowanym serwerze odłączonym od sieci z klientem na konsoli serwera. Podczas eksperymentu nie przeprowadzano na tej stacji żadnych dodatkowych czynności. Każdy program uczący był uruchamiany oddzielnie tak, że nigdy dwa nie funkcjonowały jednocześnie, obniżając wydajność jednostki obliczeniowej.

Programy uczące same zwracają czas generacji w sekundach. Jest on mierzony od momentu startu generacji do zakończenia procesu uczącego, a więc włącznie z opóźnieniami systemu plików, itp. Podawany tu czas to rzeczywisty czas działania, a nie jak często się praktykuje tzw. rzeczywisty czas zajętości procesora. Korzystanie z czasu zajętości procesora pozwala ograniczyć wpływ czynników zewnętrznych na wyniki pomiarów, ale jednocześnie prowadzi do mylących wniosków, gdyż sugeruje, że czasy są znacznie niższe niż rzeczywiście odczuwane przez operatora.

Wszystkie te względy każą traktować zaprezentowane poniżej wyniki jedynie poglądowo. Aby otrzymać porównanie bardziej statystycznie wiarygodne należałoby powtarzać testy wielokrotnie dla oddzielnie generowanych prób tej samej wielkości i uśredniać wyniki.

4.1.3 Środowisko testowe

Testy przeprowadzano na komputerze Pentium-Celeron 533Mhz ze 128MB pamięci operacyjnej[2]. Ta sama fizycznie stacja była uruchamiana z systemami: Microsoft Windows 98, RedHat Linux 6.1 i 4.4FreeBSD.

Pomiary rozpoczęto od wszystkich trzech dostępnych autorowi systemów operacyjnych. Wyniki wstępnych testów potrzebne były do wyboru platformy, na której przeprowadzano wszelkie dalsze eksperymenty. Tabela 4.1 pokazuje czasy generacji uzyskane w trzech środowiskach przy wykorzystaniu identycznych danych wejściowych i parametrów.

Dużym zaskoczeniem jest znacząca różnica szybkości tych samych algorytmów w systemie Windows i w systemach uniksowych. Wersja windows była w zależności od algorytmu od trzech do siedmiu razy wolniejsza (sic!). Przyczyną takiego stanu rzeczy jest prawdopodonie od dawna krytykowany menedżer pamięci firmy Microsoft – powszechnie uważany za wolny i mało skuteczny z tendencją do nadmiernego stronicowania. Wszystkie programy systemu *RENEG* alokują i zwalniają bowiem bardzo dużo niewielkich bloków pamięci – zadanie zdecydowanie najtrudniejsze dla menedżera alokacji. Innym prawdopodonym powodem mogłobyć niejawne przydzielania przez Windows 98 niższych priorytetów wszystkim procesom systemu za wyjątkiem pracującego na pierwszym planie. W tym wypadku na pierwszym planie pozostawała przeglądarka WWW a sam proces mógł otrzymać niższy priorytet.

Jądra systemów uniksowych słyną ze skutecznego zarządzania pamięcią, oszczędnego korzystania z plików wymiany i bardzo dobrze określonej hierarchii priorytetów niezależnych od tego, czy aplikacja znajduje się na pierwszym planie, jest na drugim planie, czy w ogóle nie posiada widocznego interfejsu tak, jak jest w tym przypadku (strumienie `stdout` i `stderr` są zamykane zanim program przystąpi do obliczeń). Niezależnie od domniemanych przyczyn niska wydajność najpopularniejszego na świecie systemu operacyjnego jest bulwersująca.

Bardzo bliskie (w granicach błędu) wyniki dla Linux i FreeBSD mogą wynikać z zastosowania tego samego kompilatora GNU C.[3] Jednocześnie,

[2]Gwoli sprawiedliwości trzeba zaznaczyć, że systemy Microsoft Windows i FreeBSD korzystały z całej dostępnej pamięci natomiast Linux był skonfigurowany w sposób udostępniający jedynie pierwsze 64MB

[3]Jest duża szansa, że zastosowanie kompilatora GNU C/C++ zamiast Microsoft Visual C/C++ mogłoby istotnie podnieść wydajność implementacji dla Windows. Zestaw narzę-

Algorytm (parametr)		Windows 98	Linux (2.2.13)	4.4FreeBSD
Pearl	(0.001)	4 s.	1 s.	1 s.
SGS	(0.05)	141 s.	33 s.	33 s.
PC-1	(0.02)	21 s.	3 s.	4 s.
K2	(13246768,7)	13 s.	3 s.	2 s.
Ramowy	(0.001)	107 s.	28 s.	26 s.

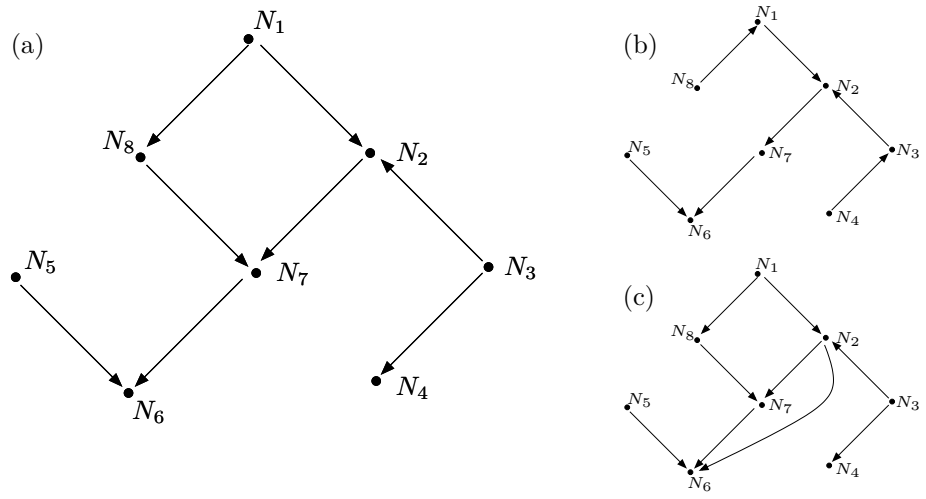
Tabela 4.1: Test porównawczy wydajności dla różnych systemów operacyjnych. Badania przeprowadzono na próbie `n5000.dbf` złożonej z 5000 wierszy i 8 kolumn. Wszystkie algorytmy (poza Pearl) zwróciły w tym przypadku strukturę sieci identyczną ze strukturą oryginalnej sieci `n.net`, z której pochodziła próba – zob. rys. 4.1.

zważywszy na fakt, że jądra obu systemów są inaczej zbudowane i zupełnie niezależnie zaimplementowane, sugerują one, że osiągnięcie lepszego wyniku w systemie wielozadaniowym jest niemożliwe lub bardzo trudne. Autor podejrzewa, że ewnetualne kompilacje w innych systemach nie przyniosą zauważalnej poprawy (przy zachowaniu tej samej platformy sprzętowej).

W konsekwencji zdecydowano się na przeprowadzanie testów na platformie Linux, pomimo mniejszej ilości dostępnej pamięci operacyjnej (wynika stąd ryzyko zwiększonego stronicowania pamięci). Linux oferował wysoką wydajność i komfort pracy niedostępny we FreeBSD.[4]

dzi programistycznych GNU dla Windows jest dostępny w Internecie pod nazwą DJGPP.

[4]Uwaga ta dotyczy instalacji autora i nie ma na celu włączania się w ogólną animozję środowisk FreeBSD i Linux.



Rysunek 4.1: (a) struktura sieci n użyta do generacji prób od `n500.dbf` do `n15000.dbf`. Po prawej stronie wynikowe sieci przy próbie wejściowej 1000 elementów: (b) polidrzewo wygenerowane algorytmem Pearla i (c) acykliczny graf skierowany z algorytmu ramowego

4.2 Testy wydajności i efektywności

Zasadniczą fazę testów rozpoczęto od stosunkowo niewielkiej próby z niedużą liczbą atrybutów. Na rys. 4.1 pokazano oryginalną sieć `n.net` i wygenerowane w procesie testów dwie struktury wynikowe. Zaczerpnięto je ze zbioru rezultatów podstawowych testów jakościowych, tzn. testów w których porównywano zarówno jakość jak i szybkość. Wykorzystanie próby o niewielkiej ilości zmiennych ograniczyło znacznie dokładność pomiaru czasu, ale za to pozwoliło na drobiazgową kontrolę jakości wygenerowanych struktur (co jest bardzo kosztowne dla struktur o większej ilości wierzchołków).

Wyniki testów jakościowo-wydajnościowych przedstawiono w tabeli 4.2. W porównaniu nie umieszczono alorytmu FCI, ze względu na inny rodzaj realizowanego przezeń zadania.

W każdej komórce zawarte są dwie liczby: czas działania alorytmu i (po znaku ukośnika) wartość oceny wyniku. W celu porównania jakościowego wyników poszczególnych algorytmów wprowadzono bowiem intuicyjną funkcję oceniającą. Zasada jej obliczania jest następująca: suma ilości krawędzi bra-

Algorytm	n500.dbf	n1000.dbf	n1500.dbf	n5000.dbf	n15000.dbf
Chow-Liu (N_1)	<1s./0.188	<1s./0.188	<1s./0.188	1s./0.188	2s./0.188
Pearl (0.001)	<1s./0.188	1s./0.188	<1s./0.188	2s./0.188	2s./0.188
SGS (0.05)	1s./0.500	4s./0.438	8s./0.188	33s./0.000	107s./0.000
PC-1 (0.02)	1s./0.188	1s./0.188	2s./0.063	3s./0.000	12s./0.000
PC-2 (0.02)	1s./0.188	1s./0.188	1s./0.000	4s./0.000	13s./0.000
PC-3 (0.02)	<1s./0.188	1s./0.188	1s./0.000	4s./0.000	13s./0.000
PC*-1 (0.02)	<1s./0.188	<1s./0.188	1s./0.063	4s./0.000	11s./0.000
PC*-2 (0.02)	1 s./0.188	1 s./0.188	1 s./0.188	5 s./0.000	12 s./0.000
PC*-3 (0.02)	<1s./0.188	<1s./0.188	1s./0.188	4s./0.000	12s./0.000
K2 (13245768,7)	<1s./ 0.000	1s./0.000	1s./0.000	3s./0.000	7s./0.000
Ramowy (0.001)	1s./0.125	2s./0.063	5s./0.000	28s./0.000	93s./0.000

Tabela 4.2: Wyniki testów przeprowadzonych na próbach pochodzących z rozkładu `n.net`

kujących, ilości krawędzi nadmiarowych i ilości krawędzi źle zorientowanych podzielona przez dwukrotność ilości krawędzi w grafie wyjściowym. Przy takiej heurystyce wartość dopasowania równa zero oznacza, że wynik jest identyczny z oryginałem, a wartości powyżej 0.5 świadczą o raczej złym dopasowaniu. Sieć o poprawnej strukturze i wszystkich krawędziach źle zorientowanych ma wartość dopasowania równą właśnie 0.5 i jest to najłagodniejszy przypadek grafu o takiej wartości dopasowania. Sieć, w której brakuje wszystkich krawędzi, która ma dodatkowo kilka krawędzi nie należących do oryginalnego grafu ma już dopasowanie powyżej 0.5.

Wyniki testów pokazują, że zdecydowanie najszybszym algorytmem jest K2, jeżeli wziąć pod uwagę konieczność uzyskania wyniku dokładnego. Następne na liście są algorytmy grupy PC. Charakterystyczne są zbliżone wyniki wszystkich algorytmów tej grupy. Trudno oceniać szybkość przy tak krótkim czasie obliczeń, ale wydaje się, że rozszerzenie PC* nie przynosi w stosunku do PC większego zysku niż 10%. Nieoczekiwanie działa jednak nieco mniej stabilnie od oryginalnego PC (odwrotnie niż podejrzewali autorzy) – zachowanie to zależy jednak od konkretnego przykładu.

Stosowanie heurystyk PC-1, PC-2 i PC-3 również nie przynosi dużych

zmian szykości, daje natomiast nieznaczną poprawę stabilności, co zauważono także dla wielu innych przykładów.

Należy też zwrócić uwagę na wysoką skuteczność algorytmów Pearla i Chow-Liu. Wprawdzie nie generują one właściwej sieci tylko jej drzewiaste przybliżenie, ale są przy tym bardzo szybkie i potrafią uzyskać niezły wynik już dla bardzo niewielkiej próby. Nie widać tego w tabeli, ale wszystkie struktury zwrócone przez Chow-Liu były identyczne niezależnie od wielkości próby. Już dla 500 wierszy osiągnięto maximum informacji. Podobną właściwość wykazał algorytm Pearla. Z tego powodu nadają się one bardzo dobrze do przeprowadzania badań wstępnych (np. w celu ustalenia potencjalnie możliwych porządków wejściowych dla algorytmu K2). Podobną właściwość (niezłą skuteczność dla niewielkieo zioru wejściowego) wykazał też algorytm ramowy. Mając podobnie złe wyniki czasowe co SGS, zwraca wyniki o wiele od niego lepsze (dla prób tej samej wielkości).

W testach jakościowych interesujący wydawał się także wpływ parametrów pomocniczych algorytmu na wyniki. Zgodnie z oczekiwaniami testy wykazały, że podnoszenie poziomu ufności w algorytmach opartych o testy χ^2 prowadzi do zwiększenia liczby włączonych do grafu krawędzi. W skrajnych przypadkach generowane są struktury sprzeczne, złożone ze zbyt wielu krawędzi, w konsekwencji zawierające cykle, których nie da się uniknąć. Dla tak niekorzystnych przypadków algorytmy przerywały pracę.

Podnoszenie poziomu ufności nieznacznie wydłuża też czas obliczeń. Jak pamiętamy, krawędzie, które pozostają w strukturze wynikowej są dla algorytmów bazujących na SGS przypadkami najbardziej czasochłonnymi. Oczywistym jest więc, że zwiększenie grafu wynikowego o dodatkowe krawędzie wydłuża także czas obliczeń.

Dla celów referencyjnych struktury sieci uzyskanych w wyniku testów zamieszczono na stronie WWW projektu (pozycja *download* w menu głównym). Dostępne są one także na płycie CD-ROM projektu w archiwum `test/spool`.

4.3 Testy wydajności

Testy wydajności prowadzono na próbach o większej ilości kolumn i wierszy, poczynając od sieci `z.net` o 30 kolumnach (próby od 500 do 60000 elementów) a kończąc na próbie z rozkładu `y.net` o 210 kolumnach i 65536 wier-

szach. W tym wypadku czasy obliczeń były wielokrotnie dłuższe i nie zawsze było możliwe odizolowanie procesu od zakłóceń zewnętrznych (autor nie miał do dyspozycji oddzielnej stacji testowej). Starano się jednak zminimalizować inne zadania obciążające system i podnieść priorytet procesów uczących tak, że zakłócenie nie powinno wpływać na wynik o więcej niż 10%. Wszystkie wyniki w tej części zostały zaokrąglone.

Sieć `z.net` to sieć o 30 zmiennych i 43 krawędziach posiadająca strukturę odległą od drzewiastej. Sieć tą charakteryzuje wysoka ilość d-połączeń – praktycznie wszystkie krawędzie spotykają się głowami strzałek z inną krawędzia w jakimś wierzchołku. Zatem struktura sieci powinna być w miarę dokładnie uczona przez większość algorytmów, Sieć `y.net` powstała z sieci `z.net` poprzez siedmiokrotne powielenie jej struktury. Ma zatem 210 zmiennych i 301 krawędzi.

Na wykresie przedstawiono zlogarytmowany czas obliczeń dla poszczególnych algorytmów w zależności od wielkości próby. Należy podkreślić, że żaden z algorytmów nie odtworzył próby dokładnie, choć kilka wyników było bardzo blisko rzeczywistej struktury. Wynika to prawdopodobnie z faktu, że 2 krawędzie (krawędzie te były pomijane przez wszystkie procedury) reprezentowały wyraźnie niższy niż pozostałe poziom zależności. Bardzo dobre sieci wynikowe były zwrcane przez algorytm PC dla prób od 15000 wierszy i co znamienne dla 60000 występowało pewne pogorszenie – zjawisko charakterysyczne, gdy zostanie przekroczona krytyczna wielkość próby i słabe zależności zaczynają stawać się statystycznie istotne. Prawie tak samo dobre wyniki algorytm K2 zwracał już przy próbie 1000 elementowej.[5]

Podczas testów na dużych próbach największe trudności sprawił algorytm ramowy. Okazało się bowiem, że warunek stopu (błąd względny) jest zależny od konkretnej próby i trzeba go eksperymentalnie dobierać. Prawdopodobnie należałoby rozważyć zaproponowanie lepszego warunku stopu. Z tego powodu większość wyników z testów algorytmu ramowego było bardzo złych (zawierało bardzo niewiele krawędzi).

Przeprowadzono też testy na próbie zawierającej 200 kolumn i ponad 60000 wierszy. Do tego testu wybrano algorytmy, które okazały się najszybsze w poprzednich. Algorytm PC prowadził obliczenia 12 godzin i zwrócił wynik zawierający nieusuwalne cykle. To samo zadanie było rozwiązywane przez algorytm K2 w czasie 11 godzin. Zwrócony wynik cechowało niestety

[5]Porównanie za pomocą prostej funkcji oceniającej opisanej w poprzednim punkcie.

zwiększanie rzędu zależności wzraz ze wzrostem rangi wierzchołka w zadanym porządku (tzn. wierzchołki bliżej końca porządku miały więcej rodziców niż powinny mieć). Algorytm Pearla zwrócił przybliżenie drzewiaste dla tych samych danych wejściowych już po 40 minutach.

Dla celów referencyjnych struktury sieci uzyskanych w wyniku testów zamieszczono na stronie WWW projektu (pozycja *download* w menu głównym). Dostępne są one także na płycie CD-ROM projektu w archiwum `test/spool`.

Rozdział 5

Podsumowanie i wnioski

W niniejszej pracy rozważono zagadnienie zdalnej generacji sieci Bayesa. Przedstawiono pojęcia podstawowe z zakresu sieci przyczynowo-skutkowych i zasady funkcjonowania poszczególnych algorytmów uczących. Następnie zrealizowano demonstracyjny system uczący o nazwie *RENEG*, który pozwolił wykonać testy porównawcze dla dziewięciu różnych algorytmów generacji.

Zastosowano twórcze podejście do implementacji istniejących algorytmów, niejednokrotnie stosując własne usprawnienia w postaci równoważnych przekształceń oryginalnych sformułowań funkcji i metod postępowania, inwersję reprezentacji, wymianę używanych przez autorów algorytmów podstawowych (niższego poziomu) na alternatywne skuteczniejsze algorytmy, ingerowanie w kolejność podawania podzbiorów do testów niezależności, oraz stosując inżynierskie usprawnienia w rodzaju zamiany licznych mnożeń zmiennoprzecinkowych na sumy logarytmów, czy tablicując funkcje pomocnicze. W niektórych algorytmach modyfikowano nieznacznie warunki stopu, inne z kolei wymagały doszczegółowienia elementów mniej kluczowych, ale niezbędnych do funkcjonowania i nieokreślonych przez autorów. Zaproponowano też własny mechanizm zliczania wartościowań zmiennych i grup zmiennych w próbie oparty o zagnieżdżane drzewa pozycyjne. Wszystkie wykonane usprawnienia i modyfikacje zebrano w paragrafie 3.3.3.

W wyniku testów porównawczych zaobserwowano, że do uczenia niedużych sieci (do kilkunastu zmiennych) można z powodzeniem stosować algorytmy PC, PC*, K2 i Ramowy, uzyskując szybko zadowalająco dokładne odwzorowanie rozkładu, podczas gdy dla sieci o dużej ilości wierzchołków algorytm Ramowy sprawuje się dużo gorzej. W sytuacjach, gdy nie jest możliwe

bardzo długie oczekiwanie na wyniki dla dużej próby, wstępne oszacowanie można otrzymać relatywnie szybko korzystając z algorytmu Pearla.

Zaimplementowany system został zbudowany w oparciu o architekturę klient/serwer, gdzie serwer WWW pełni rolę serwera aplikacji. Strona klienta oparta jest na przeglądarce WWW a części systemu komunikują się przez Internet za pośrednictwem technologii CGI. System cechuje też prosty i łatwy w użyciu graficzny interfejs użytkownika oraz bogata dokumentacja: pomoc kontekstowa, instrukcja użytkownika i instrukcja eksploatacyjna.

Dzięki zastosowaniu technologii internetowych system *RENEG* zyskuje wiele na elastyczności. Praktyczny brak wymagań co do systemu operacyjnego i platformy sprzętowej klienta rozszerza bardzo zakres użytkowników. Jest to szczególnie istotne przy zastosowaniach akademickich, bowiem środowiska akademickie cechuje wciąż jeszcze korzystanie z bardzo różnych platform sprzętowych i systemowych. Zastosowane technologie internetowe (http, html, Java i JavaScript) cechuje daleko posunięta przenośność między platformami.

Nie ograniczając się do zapewnienia przenośności strony klienta, autor posunął się krok dalej. Zastosowano techniki programowania przenośnego w językach C/C++, co zaowocowało utworzeniem systemu przenośnego także ze względu na platformę serwera. Dzięki temu stało się możliwe porównywanie własności obliczeniowych różnych systemów operacyjnych.

Nieoczekiwanym rezultatem pracy jest demonstracja niskiej przydatności obliczeniowej systemu Microsoft Windows 98 i kompilatora Microsoft Visual C/C++, w którego środowisku aplikacje funkcjonowały od trzech do siedmiu razy wolniej niż w systemach uniksowych (dotyczy tej samej platformy sprzętowej). Zdecydowanie więc odradza się stosowania tego systemu i kompilatora wszystkim, którzy zamierzają wykorzystywać swój komputer do celów obliczeniowych.

Wyniki prac nad systemem *RENEG* wskazują na dalsze możliwości rozwoju. W tej perspektywie rysują się między innymi problemy związane z usuwaniem cykli zwracanych niekiedy przez algorytmy grupy PC, czy z unormowaniem lub zaproponowaniem alternatywnego warunek stopu algorytmu ramowego. Pierwszy z tych problemów należałoby rozwiązywać eliminując z cyklu krawędzie najmniej istotne dla jakości dopasowania całej struktury, drugi – prowadząc dalsze testy i analizy własności tego algorytmu, co mogłoby doprowadzić do sformułowania warunku zupełnie nowego rodzaju.

Kolejnym punktem w rozwoju projektu mogłoby też być rozszerzenie o możliwość uczenia na podstawie informacji niepełnej, czyli tabeli posiadającej niewypełnione niektóre komórki.

Spis literatury

[1] Acid S. Campos de L.M. *BENEDICT: An Algorithm for Learning Probailistic Belief Networks.* Departamento de Ciencias de la Computacion e I.A. Universidad de Granada. 180771-Granada. Spain.

[2] Acid S. Campos de L.M. *An algorithm for Finding Minimum d-Separating Sets in Belief Networks.* Departamento de Ciencias de la Computacion e I.A. Universidad de Granada. 180771-Granada. Spain.

[3] Banachowski L., Diks K., Rytter W. *Algorytmy i struktury danych.* Wydawnictwo Naukowo-Techniczne. Warszawa 1996. s.126-134.

[4] Bouckaert R.R., *Properties of Bayesian Belief network learning algorithms.* w Proceedings of 10th Conf. on Uncertainty in Artificial Intelligence. Seattle. Washington. Morgan Kaufmann. s. 102-109.

[5] Chow C.K., Liu C.N. *Approximating discrete probability distributions with dependence trees.* IEEE Transactions on Information Theory 14(3), 1968. s.462-467.

[6] Cooper G.F., Herskovits E. *A Bayesian Method for the Induction of Probabilistic Networks from Data.* Machine Learning vol. 9, no.4. Kluwer Academic Publishers. Boston 1992. s. 309-347.

[7] Cowell R.G. Dawid A.Ph. Lauritzen S.L. Spiegelhalter D.J. *Probabilistic Networks and Expert Systems.* Statistics for Engineering and Information Science Springer Verlag Series. Springer Verlag 1999.

[8] Heckerman D. Horovitz E. *Inferring Informational Goals from Free Text Queries: A Bayesian Approach.* Microsoft Research. Redmond. WA

98052-6399. In *Proceedings of the Fourtheenth Conference on Uncertainty in Artificial Intelligence.* Madison, WI. July 1998. Morgan Kaufman Publishers. s. 230-237.

[9] Jensen Finn V. *Introduction to Bayesian Networks.* Springer Verlag. New York 1996.

[10] Klonecki W. *Statystyka dla inżynierów.* Wydawnictwo Naukowe PWN. Warszawa-Wrocław 1999.

[11] Kłopotek M.A, Wierzchoń S. T. *Discovery of Bayesian Networks from Data with Maintainance of Partially Oriented Graphs.* Intelligent Infoprmation Systems. Advances in Soft Computing Series of Physica-Verlag/Springer Verlag, Heidelberg/New York 2000, s. 277- 288.

[12] Kullback S., Leibler RA. *On information and sufficiency.* Annals of Mathematical Statistics,22: s. 79-86.

[13] Lam W., Bacchus F. *Learning Bayesian Belief Networks: an approach based on MDL principle.* Computational Intelligence, Volume 10. Number 3, 1994. s. 269-293.

[14] Pearl J., *Probabilistic reasoning in intelligent systems.* Morgan Kaufman. San Mateo, CA 1988.

[15] Praca zbiorowa. *Numerical Recipes.* Cambridge University Press, New York 1988. s.155-165, 469-472.

[16] Ross K.A., Wright Ch.R.B. *Matematyka Dyskretna.* Wydawnictwo Naukowe PWN, Warszawa 1999

[17] Shachter R.D. *Evaluating influence diagrams.* Operations Research 34 (1986). s.871-882

[18] Spirtes P., Glymour C. Scheines R. *Causation, Prediction and Search.* Lecture Notes in Statistics 81. Springer Verlag. New York 1993. s. 112-124, 163-191.

[19] Sysło M.M., Deo N. Kowalik J.S. *Algorytmy optymalizacji dyskretnej.* Wydawnictwo Naukowe PWN. Warszawa 1995. s. 208-217.

[20] *Proposal of a BayEsian Network InterOperability Format Standard for Knowledge Interchange (BENIOFSKI).* Koordynator: Maciej Michalewicz. `http://www.ipipan.waw.pl/~ klopotek/crit2/index.html` Data ostatniej modyfikacji: 15-05-1998.

[21] *dBASE III,IV,5 DBF file format.* `http://www.apptools.com/dbase/faq/qformt.htm` Data ostatniej modyfikacji: 5-07-2000.