



**POLITECHNIKA WROCŁAWSKA**  
**WYDZIAŁOWY ZAKŁAD INFORMATYKI**  
**WYDZIAŁ INFORMATYKI I ZARZĄDZANIA**

**Rozprawa doktorska**

**METODY OBLICZANIA NIEZAWODNOŚCI SIECI *K*-TERMINALI**

**Lech J. Madeyski**

**Promotor:** dr hab. Zygmunt Mazur, prof. P.Wr.

**Recenzenci:** dr hab. Krystyna Balińska,  
Politechnika Poznańska

dr hab. Adam Grzech, prof. P.Wr.  
Politechnika Wrocławska

Wrocław 1999

## Spis treści

|                                                                                             |           |
|---------------------------------------------------------------------------------------------|-----------|
| <b>1. WSTĘP</b>                                                                             | <b>5</b>  |
| 1.1. PODSTAWOWE DEFINICJE TEORII GRAFÓW                                                     | 6         |
| 1.2. MODEL SIECI                                                                            | 8         |
| 1.3. MIARY OCENY NIEZAWODNOŚCI SIECI                                                        | 10        |
| 1.3.1. Wybór odpowiedniej miary oceny niezawodności sieci                                   | 12        |
| 1.4. ZAKRES ZASTOSOWAŃ MIAR NIEZAWODNOŚCI BAZUJĄCYCH NA SPÓJNOŚCI SIECI                     | 13        |
| 1.4.1. Poziom kręgosłupowy sieci z przełączaniem pakietów                                   | 13        |
| 1.4.2. Poziom kręgosłupowy sieci z przełączaniem kanałów                                    | 14        |
| 1.4.3. Inne zastosowania                                                                    | 14        |
| 1.5. ZŁOŻONOŚĆ OBLICZENIOWA                                                                 | 15        |
| 1.6. CEL I ZAKRES PRACY                                                                     | 15        |
| <b>2. PRZEGLĄD WAŻNIEJSZYCH METOD OBLICZANIA MIAR NIEZAWODNOŚCI SIECI</b>                   | <b>18</b> |
| 2.1. ALGORYTMY ZNAJDUJĄCE DOKŁADNE ROZWIĄZANIE                                              | 18        |
| 2.1.1. Efektywne algorytmy dla pewnych klas sieci i zachowujące niezawodność redukcje grafu | 18        |
| 2.1.2. Metoda wyliczania przestrzeni stanów (state space enumeration)                       | 19        |
| 2.1.3. Metody bazujące na zbiorach sprawności ( $K$ -drzewach)                              | 20        |
| 2.1.3.1. Metoda zawierania-wykluczania ( <i>inclusion-exclusion formula</i> )               | 20        |
| 2.1.3.2. Metoda rozłącznych iloczynów (sum of disjoint products)                            | 21        |
| 2.1.4. Algorytm faktoryzacji                                                                | 21        |
| 2.1.5. Algorytm dekompozycji grafu                                                          | 22        |
| 2.2. METODY MONTE CARLO                                                                     | 23        |
| 2.3. METODY KONSTRUOWANIA OGRANICZEŃ NA NIEZAWODNOŚĆ SIECI                                  | 26        |
| 2.3.1. Ograniczenia na niezawodność wszystkich terminali                                    | 27        |
| 2.3.2. Ograniczenia na niezawodność dwóch terminali                                         | 28        |
| 2.3.3. Ograniczenia na niezawodność $K$ -terminali                                          | 29        |
| 2.4. INNE METODY                                                                            | 29        |
| 2.5. PODSUMOWANIE STANU BADAŃ                                                               | 30        |
| <b>3. ALGORYTM FAKTORYZACJI</b>                                                             | <b>32</b> |
| 3.1. ZACHOWUJĄCE NIEZAWODNOŚĆ REDUKCJE I DEKOMPOZYCJE GRAFU                                 | 35        |
| 3.2. STRATEGIA WYBIERANIA KRAWĘDZI GRAFU DO FAKTORYZACJI                                    | 41        |

|                                                                                                                 |           |
|-----------------------------------------------------------------------------------------------------------------|-----------|
| 3.3. SZKIELET ALGORYTMU FAKTORYZACJI                                                                            | 42        |
| 3.4. PRZYKŁAD ZASTOSOWANIA ALGORYTMU                                                                            | 43        |
| 3.5. PODSUMOWANIE ROZDZIAŁU                                                                                     | 44        |
| <b>4. ANALIZA PESYMISTYCZNEJ ZŁOŻONOŚCI OBLICZENIOWEJ ALGORYTMÓW FAKTORYZACJI</b>                               | <b>45</b> |
| 4.1. ZŁOŻONOŚĆ CZASOWA ALGORYTMÓW FAKTORYZACJI                                                                  | 46        |
| 4.1.1. Niezmienniki grafu a twierdzenie faktoryzacji                                                            | 47        |
| 4.1.2. Wykorzystanie niezmienników grafów do analizy złożoności niektórych algorytmów faktoryzacji              | 47        |
| 4.2. ZŁOŻONOŚĆ PAMIĘCIOWA ALGORYTMÓW FAKTORYZACJI                                                               | 52        |
| 4.3. PODSUMOWANIE ROZDZIAŁU                                                                                     | 52        |
| <b>5. ALGORYTM FAKTORYZACJI <i>FACT</i></b>                                                                     | <b>54</b> |
| 5.1. ANALIZA PESYMISTYCZNEJ ZŁOŻONOŚCI OBLICZENIOWEJ ALGORYTMÓW FAKTORYZACJI Z WYKORZYSTANIEM GRAFÓW PEŁNYCH    | 54        |
| 5.1.1. Idea wykorzystania grafów pełnych                                                                        | 54        |
| 5.1.2. Złożoność czasowa algorytmu <i>Fact</i> i jego modyfikacji                                               | 55        |
| 5.1.2.1. Niezawodność wszystkich terminali                                                                      | 56        |
| 5.1.2.2. Niezawodność <i>K</i> -terminali                                                                       | 62        |
| 5.1.3. Złożoność pamięciowa algorytmów faktoryzacji                                                             | 67        |
| 5.2. PODSUMOWANIE ROZDZIAŁU                                                                                     | 71        |
| <b>6. ALGORYTM FAKTORYZACJI <i>FACT&amp;CACHE</i> Z WYKRYWANIEM RÓWNOWAŻNYCH PODPROBLEMÓW</b>                   | <b>72</b> |
| 6.1. ZMODYFIKOWANA TECHNIKA PROGRAMOWANIA DYNAMICZNEGO                                                          | 74        |
| 6.2. WYKRYWANIE RÓWNOWAŻNYCH PODPROBLEMÓW                                                                       | 75        |
| 6.2.1. Metoda kodowania podproblemów                                                                            | 76        |
| 6.3. STRATEGIA DEKOMPOZYCJI PROBLEMU                                                                            | 78        |
| 6.3.1. Heurystyka przyporządkowywania priorytetów                                                               | 78        |
| 6.4. PODSUMOWANIE ROZDZIAŁU                                                                                     | 80        |
| <b>7. PORÓWNANIE EFEKTYWNOŚCI RÓŻNYCH ALGORYTMÓW OBLICZANIA NIEZAWODNOŚCI <i>K</i>-TERMINALI</b>                | <b>81</b> |
| 7.1. ZŁOŻONOŚĆ OBLICZENIOWA ALGORYTMU <i>FACT</i>                                                               | 81        |
| 7.2. ZŁOŻONOŚĆ OBLICZENIOWA ALGORYTMU <i>FACT&amp;CACHE</i>                                                     | 81        |
| 7.3. PORÓWNANIE MIAR PESYMISTYCZNEJ ZŁOŻONOŚCI CZASOWEJ ALGORYTMÓW OBLICZANIA NIEZAWODNOŚCI <i>K</i> -TERMINALI | 83        |

|                                                                                                                                      |            |
|--------------------------------------------------------------------------------------------------------------------------------------|------------|
| 7.3.1. Porównanie określonych analitycznie miar pesymistycznej złożoności czasowej różnych algorytmów faktoryzacji                   | 84         |
| 7.3.2. Porównanie otrzymanych eksperymentalnie miar pesymistycznej złożoności czasowej algorytmu <i>Fact</i> i <i>Fact&amp;Cache</i> | 86         |
| 7.4. PORÓWNANIE EFEKTYWNOŚCI ALGORYTMÓW FAKTORYZACJI NA PRZYKŁADZIE GRAFÓW LOSOWYCH Z UWZGLĘDNIENIEM GĘSTOŚCI GRAFÓW                 | 87         |
| 7.5. EFEKTYWNOŚĆ ALGORYTMU <i>FACT&amp;CACHE</i> NA TLE AKTUALNIE NAJEFEKTYWNIJSZYCH ALGORYTMÓW OBLICZANIA NIEZAWODNOŚCI K-TERMINALI | 91         |
| 7.5.1. Porównanie czasów obliczeń                                                                                                    | 92         |
| 7.5.2. Porównanie liczby liści binarnego drzewa obliczeń algorytmów faktoryzacji                                                     | 95         |
| 7.6. WSPÓŁCZYNNIK TRAFIENIA ALGORYTMU <i>FACT&amp;CACHE</i>                                                                          | 97         |
| 7.7. PODSUMOWANIE ROZDZIAŁU                                                                                                          | 99         |
| <b>8. PODSUMOWANIE PRACY</b>                                                                                                         | <b>100</b> |
| 8.1. WNIOSKI KOŃCOWE I PODSUMOWANIE REZULTATÓW                                                                                       | 100        |
| 8.2. PROJEKTOWANIE NIEZAWODNYCH SIECI                                                                                                | 102        |
| 8.3. PROPOZYCJE DALSZYCH BADAŃ I MODYFIKACJI ZAPROPONOWANYCH ALGORYTMÓW                                                              | 103        |
| 8.3.1. Zastosowanie miar zdolności do prawidłowego działania ( <i>performability measures</i> )                                      | 103        |
| 8.3.2. Modyfikacja techniki wykrywania równoważnych podproblemów                                                                     | 104        |
| 8.3.2.1. Wykorzystanie osiągnięć w dziedzinie testowania izomorfizmu i kanonicznego etykietowania grafów                             | 105        |
| 8.3.3. Modyfikacja funkcji haszującej wykorzystywanej do wyszukiwania równoważnych podproblemów                                      | 107        |
| 8.3.4. Faktoryzacja i redukcje na sieciach o zawodnych węzłach                                                                       | 107        |
| 8.3.5. Faktoryzacja z aproksymacją                                                                                                   | 108        |
| 8.3.6. Wykorzystanie zmodyfikowanego algorytmu <i>Fact&amp;Cache</i> przy projektowaniu niezawodnych sieci                           | 109        |
| <b>BIBLIOGRAFIA</b>                                                                                                                  | <b>111</b> |
| <b>INDEKS OZNACZEŃ, SKRÓTÓW I STAŁYCH NUMERYCZNYCH</b>                                                                               | <b>122</b> |
| <b>WYKAZ RYSUNKÓW</b>                                                                                                                | <b>123</b> |
| <b>WYKAZ TABEL</b>                                                                                                                   | <b>124</b> |
| <b>ZAŁĄCZNIK 1. – ZESTAWIENIE TYPOWYCH SIECI WYKORZYSTYWANYCH DO TESTOWANIA EFEKTYWNOŚCI ALGORYTMÓW</b>                              | <b>125</b> |
| <b>SKOROWIDZ</b>                                                                                                                     | <b>131</b> |



## 1. WSTĘP

Niezawodność złożonych systemów – począwszy od sieci komputerowych czy telekomunikacyjnych, poprzez sieci transportowe, elektryczne, systemy mechaniczne po układy elektroniczne dużej skali integracji – jest od dłuższego czasu tematem intensywnych badań. Ponadto niezawodność tak oprogramowania, jak i sprzętu komputerowego stała się niezwykle istotna szczególnie w aplikacjach komercyjnych i wojskowych. Lata siedemdziesiąte zapoczątkowały gwałtowny rozwój badań dotyczących niezawodności sieci komputerowych (Arpanet, Internet). We wszystkich tych zastosowaniach można zauważyć dwa główne cele stosowanej teorii niezawodności: określenie niezawodności danego systemu (analiza) i projektowanie możliwie najbardziej niezawodnego systemu z danych komponentów (synteza) [Shier91].

W pracy tej rozpatrywany jest, powszechnie podejmowany przez badaczy, problem analizy niezawodności sieci poprzez obliczanie bazujących na spójności miar ich niezawodności. Jest to o tyle usprawiedliwione, że w większości sieci stosowana jest technika dynamicznego wyznaczania tras (*dynamic routing*). W razie uszkodzenia niektórych łączy można wykorzystać inne trasy pod warunkiem, że takowe istnieją (sieć jest spójna).

Sieć reprezentowana jest poprzez probabilistyczny graf nieskierowany  $G = (V, E)$  z wyróżnionym podzbiorem wierzchołków  $K$  ( $2 \leq |K| \leq |V|$ ), gdzie  $V$  jest zbiorem wierzchołków a  $E$  jest zbiorem krawędzi. Krawędzie grafu reprezentują łącza komunikacyjne, które ulegają wzajemnie niezależnie losowym uszkodzeniom ze znanym prawdopodobieństwem. Spośród miar oceny niezawodności sieci, najbardziej uniwersalną i powszechnie stosowaną miarą jest niezawodność  $K$ -terminali (*K-terminal network reliability*), definiowana jako prawdopodobieństwo, iż wszystkie węzły znajdujące się w zbiorze  $K$  są połączone za pomocą nie uszkodzonych łączy (mogą się komunikować).

Problem niezawodności  $K$ -terminali należy do klasy problemów NP-trudnych i #P-zupełnych [Valiant79b, Ball80, Ball83, Ball86]. Oznacza to, że najprawdopodobniej nie istnieje algorytm, który będzie znajdował dokładne rozwiązanie dla wszystkich możliwych wystąpień problemu w czasie wielomianowym w funkcji rozmiaru problemu. Z tego względu z jednej strony podejmowane są poszukiwania algorytmów aproksymacyjnych, z drugiej strony (efektywnych w wielu przypadkach) algorytmów znajdujących dokładne rozwiązanie.

Spośród algorytmów dokładnych rozwiązujących problem niezawodności  $K$ -terminali szczególnie dużo uwagi poświęcono algorytmom faktoryzacji. Wniesiono wielki wkład w

udoskonalenie pierwotnej wersji algorytmu faktoryzacji [Choi95, Deo92, Lai94, Page88, Page89, ResendeM86, ResendeL88, Satyanarayana83, Theologou91, Wood85, Wood86, Wood89]. Spośród algorytmów faktoryzacji najmniejszą pesymistyczną złożonością czasową charakteryzuje się algorytm opisany przez Wooda [Wood85, Wood86]. W przypadku tego algorytmu maksymalna liczba liści binarnego drzewa obliczeń  $(|V| - 2)!$  jest zagwarantowana tylko dla przypadków granicznych, gdy  $2 \leq |K| \leq 5$  oraz  $|V| - 2 \leq |K| \leq |V|$ . W przypadku algorytmów analizowanych przez Wooda, Satyanarayana i Changa [Satyanarayana83, Wood86] maksymalna liczba liści binarnego drzewa obliczeń  $(|V| - 1)!$  jest osiągalna dla dowolnego zbioru  $K$  ( $2 \leq |K| \leq |V|$ ). Efektywność tych algorytmów jest jednak niewystarczająca. Ze względu na konieczność analizy niezawodności coraz bardziej złożonych sieci, potrzebne są efektywniejsze algorytmy oceny ich niezawodności. W niniejszej pracy podjęto więc próbę modyfikacji algorytmu faktoryzacji w celu zwiększenia jego efektywności.

Realizując ten cel zaproponowano dwa nowe, konkurencyjne w stosunku do znanych dotąd algorytmów faktoryzacji, algorytmy *Fact* i *Fact&Cache* obliczania niezawodności  $K$ -terminali.

## 1.1. PODSTAWOWE DEFINICJE TEORII GRAFÓW

Teoria grafów udostępnia wygodny aparat formalny do modelowania różnego rodzaju systemów. Rozdział ten definiuje niektóre, podstawowe terminy z teorii grafów<sup>1</sup> użyteczne z punktu widzenia tej pracy.

Graf  $G = (V, E)$  składa się ze skończonych zbiorów wierzchołków  $V = \{v_1, \dots, v_n\}$  i krawędzi  $E = \{e_1, \dots, e_b\}$ . Graf  $G$  jest planarny, jeśli istnieje pewna geometryczna reprezentacja tego grafu, którą można tak narysować na płaszczyźnie, że żadne dwie jej krawędzie nie przecinają się. Graf  $G$ , który zawiera wyróżniony (niepusty) podzbiór wierzchołków  $K \subseteq V$  oznaczamy przez  $G_K$ . Dwa wierzchołki  $u$  i  $v$  są połączone, jeżeli istnieje sekwencja wierzchołków i krawędzi z wierzchołka  $u$  do wierzchołka  $v$ . Sekwencja taka jest ścieżką (*path*). Ścieżka ta jest cyklem, jeśli pierwszy i ostatni wierzchołek są

---

<sup>1</sup> Por. [Wood86], s.271-272.

identyczne. Zbiór wierzchołków  $K \subseteq V$  jest spójny lub inaczej mówiąc połączony (*connected*), jeśli istnieje ścieżka pomiędzy wszystkimi parami wierzchołków znajdującymi się w zbiorze  $K$ . Graf  $G$  jest spójny (połączony), jeśli jego zbiór  $V$  jest spójny (lub inaczej wierzchołki ze zbioru  $V$  są połączone).

Graf  $G' = (V', E')$  jest podgrafem grafu  $G = (V, E)$  jeżeli  $V' \subseteq V$  i  $E' \subseteq E$ . Przez  $V_0$  oznaczmy podzbiór wierzchołków  $V$  w grafie  $G$  a przez  $G - V_0$  oznaczmy podgraf grafu  $G$  otrzymany z tegoż grafu  $G$  przez usunięcie wszystkich wierzchołków  $v \in V_0$  i wszystkich krawędzi incydentnych do tych wierzchołków. Jeżeli  $V_0$  będzie najmniejszym zbiorem wierzchołków takim, że  $G - V_0$  jest rozłączony lub  $|V - V_0| = 1$ , to graf  $G$  jest  $k$ -spójny ( $k$ -*connected*), gdy  $|V_0| \geq k$  [Tutte66]. Inaczej mówiąc spójność (zwana też spójnością wierzchołkową) grafu  $G$  definiuje się jako minimalną liczbę wierzchołków, których usunięcie z  $G$  czyni pozostały graf niespójnym [Deo80].

Jeśli graf  $G$  jest spójny, a graf  $G - v$  jest niespójny, wtedy wierzchołek  $v$  jest wierzchołkiem rozdzielającym (*cutvertex*). Graf spójny  $G$  jest nierozdzielny (*nonseparable*), jeżeli nie zawiera wierzchołków rozdzielających. Maksymalny nierozdzielny podgraf jest blokiem (*block*). Graf spójny jest rozdzielny, jeżeli jest 1-spójny. Jeżeli graf  $G$  jest spójny, a  $G - e$  jest niespójny, to krawędź  $e$  jest mostem (*bridge*).

Jeżeli graf  $G$  można podzielić na dwie części tak, że  $G = G_1 \cup G_2$ ,  $E_1 \cap E_2 = \emptyset$ ,  $V_1 \cap V_2 = \{u, v\}$ ,  $|E_1| \geq 2$ ,  $|E_2| \geq 2$ , to para wierzchołków  $\{u, v\}$  jest parą rozdzielającą (*separating pair*).

Krawędzie równoległe (*parallel edges*) mają te same wierzchołki końcowe. Dwie nierównoległe krawędzie są przyległe, gdy mają wspólny wierzchołek. Krawędziami szeregowymi nazywamy dwie przyległe krawędzie, jeżeli ich wierzchołek wspólny jest stopnia drugiego. Zastępowanie pary krawędzi szeregowych (równoległych) przez pojedyncze krawędzie nazywamy redukcją szeregową (równoległą). Graf  $G$  jest grafem szeregowo-równoległym, jeżeli może być zredukowany do pojedynczej krawędzi poprzez redukcje szeregowo-równoległe.

Łańcuch  $\chi$  w grafie jest sekwencją wyróżnionych wierzchołków i krawędzi:  $v_1, (v_1, v_2), \dots, (v_{k-1}, v_k), v_k$  taką, że wszystkie wierzchołki wewnętrzne są stopnia 2, a wierzchołki końcowe  $v_1$  i  $v_k$  są stopnia wyższego niż 2. Łańcuch nie musi zawierać krawędzi wewnętrznych, ale musi zawierać co najmniej jedną krawędź i dwa wierzchołki końcowe.

Jeżeli dwa łańcuchy  $\chi_1$  i  $\chi_2$  mają wspólne wierzchołki końcowe  $u$  i  $v$ , to  $\chi_1 \cup \chi_2$  tworzą wielokąt (*polygon*).

$K$ -drzewo (*K-tree*) grafu  $G_K$  jest minimalnym podgrafem, który łączy wszystkie wierzchołki ze zbioru  $K$  grafu  $G_K$ . Krawędź lub wierzchołek są nieistotne (*irrelevant*), jeśli nie występują w żadnym  $K$ -drzewie. Każda krawędź lub wierzchołek, które nie są nieistotne, są istotne (*relevant*).  $G_K$  jest koherentny (*coherent*), jeżeli  $|K| > 1$  i nie zawiera nieistotnych krawędzi lub wierzchołków.  $G_K$  jest spójny, jeżeli wierzchołki z jego zbioru  $K$  są połączone.

Formacją grafu  $G_K$  jest zbiór  $K$ -drzew, których unią jest  $G_K$ . Graf  $G_K$  nie ma formacji, jeśli zawiera krawędź nieistotną lub jest niespójny.  $N_o$  jest liczbą formacji o liczności nieparzystej a  $N_e$  jest liczbą formacji o liczności parzystej. Dominacja (*domination*)  $D(G_K) = |N_o - N_e|$ . Minimalna dominacja (*minimum domination*)  $\mu(G) = \min_{K: |K|=2} D(G_K)$ .

Rozpatrywany w niniejszej pracy graf probabilistyczny  $(G_K, P)$  składa się z nieskierowanego grafu  $G_K$  z wyróżnionym podzbiorem wierzchołków  $K$  oraz funkcji  $P$ , która przypisuje krawędziom (i ewentualnie wierzchołkom) prawdopodobieństwo poprawnego funkcjonowania. W przypadku, gdy istnienie funkcji  $P$  będzie wynikało z kontekstu, rozważany graf probabilistyczny będziemy oznaczać przez  $G_K$ .

W związku z tym, że graf jest modelem sieci terminy używane w przypadku grafów (np. wierzchołki czy krawędzie) będą niekiedy używane zamiennie z terminami odnoszącymi się do sieci (np. węzły czy łącza).

## 1.2. MODEL SIECI

Ze względu na złożoność rozpatrywanych problemów i niezdolność do modelowania mechanizmów uszkodzeń, w analizie niezawodności sieci wykorzystuje się zwykle, niezależne od czasu, dyskretne modele probabilistyczne<sup>2</sup>. Struktura topologiczna sieci jest na

---

<sup>2</sup> Ball, Colbourn i Provan zwracają uwagę, że chociaż w większości klasycznych zastosowań analizy niezawodnościowej, mechanizmy i przyczyny uszkodzeń są relatywnie dobrze znane, to w przypadku analizy niezawodności sieci głównymi powodami powszechnego wykorzystywania niezależnego od czasu modelu probabilistycznego są niezdolność do modelowania mechanizmów uszkodzeń (np. uszkodzenia sieci

ogół modelowana za pomocą grafu. Najczęściej stosowany jest probabilistyczny model sieci, w którym łącza ulegają wzajemnie niezależnie losowym uszkodzeniom z określonym prawdopodobieństwem. Sieć reprezentowana jest poprzez probabilistyczny graf nieskierowany  $G_K = (V, E)$  z wyróżnionym podzbiorem węzłów  $K$  ( $2 \leq |K| \leq |V|$ ). Wierzchołki grafu reprezentują węzły (centra) komunikacyjne. Zakłada się, że są one całkowicie niezawodne. Krawędzie grafu  $e_i$  reprezentują łącza komunikacyjne, które ulegają wzajemnie niezależnie losowym uszkodzeniom ze znanym prawdopodobieństwem  $1 - p_i$ . Każde łącze komunikacyjne umożliwia dwukierunkową komunikację pomiędzy węzłami pod warunkiem, że jest ono sprawne. Model ten, wykorzystywany w większości cytowanych prac, przyjęto również w niniejszej pracy.

Czasami przyjmuje się dodatkowe założenia zawężające model (np. że niezawodności łączy są takie same), bądź go rozszerzające (np. że węzły sieci również ulegają uszkodzeniom). Argumentem przeciwko użyciu bardziej skomplikowanego modelu sieci jest fakt, iż nawet dla przyjętego, tak prostego modelu, rozważane problemy należą do klasy problemów NP-trudnych i #P-zupełnych [Valiant79b, Ball80, Ball83, Ball86]<sup>3</sup> (jest więc mało prawdopodobne, że można je rozwiązać w czasie wielomianowym).

W przyjętym dwustanowym modelu, prawdopodobieństwo działania komponentu (łącza, węzła) często określane słowem niezawodność może mieć jedną kilku interpretacji. Najczęściej<sup>4</sup> przyjmowane interpretacje to:

1. gotowość komponentu do działania (*component's availability*)<sup>5</sup>;
2. niezawodność komponentu (*component's reliability*);

Gotowość do działania jest używana w kontekście systemów naprawialnych (*repairable systems*) i jest definiowana jako prawdopodobieństwo, że w losowym punkcie

światłowodowych są powodowane przez czynniki naturalne takie jak ogień czy błąd ludzki) i właściwa rozważanym problemom trudność obliczania niezawodności sieci. Por. [Ball95], s.8-9.

<sup>3</sup> Dotyczy to również grafów planarnych [Provan86].

<sup>4</sup> Por. [Ball95] s.9.

<sup>5</sup> Jest określana również jako dyspozycyjność lub gotowość operacyjna. Por. *Słownik naukowo-techniczny angielsko-polski*, red. Sergiusz Czerni i Maria Skrzyńska, WNT 1983, s.51.

czasu komponent jest sprawny:  $\frac{MTTF}{MTTF + MTTR}$ <sup>6</sup>. Niezawodność komponentu jest zwykle definiowana jako prawdopodobieństwo, że komponent nie ulegnie uszkodzeniu w określonym czasie. Inne interpretacje prawdopodobieństwa działania komponentu są również możliwe<sup>7</sup>. Oczywiście interpretacja prawdopodobieństwa działania komponentu determinuje właściwą interpretację obliczonych miar niezawodności. W niniejszej pracy wykorzystując terminy prawdopodobieństwa działania czy niezawodności komponentu nie przypisuje się im konkretnej interpretacji. Zwykle wykorzystywane są konkretne wartości prawdopodobieństwa działania łączy szacowane dla określonych typów łączy. W przypadku sieci testowych wykorzystywanych w literaturze (jak również w tej pracy) do eksperymentalnego testowania algorytmów obliczania miar niezawodności, dla uproszczenia przyjęto wartości prawdopodobieństwa działania łączy równe 0,9.

W przypadku sieci komputerowych wierzchołki grafu reprezentują na przykład *switche*, *routery* czy *hosty*. Krawędzie natomiast reprezentują bezpośrednie połączenia pomiędzy węzłami (mogą to być linie transmisyjne lub podsieci) [Zegura96, Calvert97]. Dobre modele topologicznych struktur sieci są bardzo istotne ze względu na rozwój i analizę technologii sieciowych. Z tego względu pojawia się coraz więcej prac na temat sposobu wykorzystania modeli opartych na grafach do reprezentowania topologii dużych sieci, w szczególności sieci Internet [Thomas94, Zegura96, Calvert97].

Innym choć znacznie rzadziej stosowanym jest deterministyczny model sieci<sup>8</sup>. W modelu tym najczęściej badany jest najgorszy przypadek, w którym przeciwnik inteligentnie wybiera pewne elementy sieci do zniszczenia tak, aby sieć nie mogła realizować swoich funkcji. Model deterministyczny znajduje zastosowanie głównie w projektowaniu systemów i sieci wojskowych.

### 1.3. MIARY OCENY NIEZAWODNOŚCI SIECI

Termin niezawodność jest rozumiany jako prawdopodobieństwo, że element lub system funkcjonuje. Najczęściej formułowane są następujące miary oceny niezawodności sieci modelowanych przez grafy probabilistyczne:

---

<sup>6</sup> MTTF(Mean Time to Failure) to średni czas do zaobserwowania następnego uszkodzenia a MTTR(Mean Time to Repair) to średni czas w którym system będzie naprawiony po zaobserwowaniu awarii.

<sup>7</sup> Por. [Ball95] s.9.

<sup>8</sup> Por. [Shier91], s.3-5 oraz [Boesch86], s.240-245.



- niezawodność dwóch terminali (*2-terminal network reliability*), (przypadek gdy  $|K|=2$ ) określa prawdopodobieństwo, że pomiędzy dwoma węzłami może być przesłana wiadomość lub inaczej prawdopodobieństwo, iż istnieje ścieżka łącząca określone dwa węzły [AboElFotoh89, Beichelt91b, Brecht85, Cancela95, Chari90, Fratta78, Harms93, Shier91, Strayer95b, Torrieri94];
- niezawodność wszystkich terminali (*All-terminal network reliability*), (przypadek gdy  $|K|=|V|$ ) jest definiowana jako prawdopodobieństwo, że wszystkie węzły są połączone (mogą się komunikować) [Assous84, Ball82, Beichelt91a, Brown93, Carrasco85, Colbourn88, Harms83, Jacobs59, Karger95a, Karger95b, Karger97, Lomonosov71, Lomonosov72, Polesskii71, Strayer95b, Strayer98, Van Slyke72, Zemel82];
- najbardziej interesująca, bo będąca uogólnieniem dwóch powyższych miar, jest niezawodność  $K$ -terminali (*K-terminal network reliability*), definiowana jako prawdopodobieństwo, iż wszystkie węzły znajdujące się w zbiorze  $K$  ( $2 \leq |K| \leq |V|$ ) są połączone za pomocą nie uszkodzonych łączy (mogą się komunikować) [Cancela94, Carlier91, Carlier96, Page88, Satyanarayana83, Satyanarayana85, Strayer95a, Theologou91, Wood85, Wood86, Wood89].

Dla sieci skierowanych zaproponowano analogiczne miary niezawodności sieci (*source-to-terminal network reliability*, *source-to-all-terminal network reliability*, *source-to-K-terminal network reliability*).

Chociaż powyższe miary są najczęściej stosowane, to niekiedy używane są także inne miary oceny niezawodności sieci probabilistycznych. Przykładem może być oczekiwana liczba par węzłów, które mogą się komunikować [Kershenbaum73, Amin88, Colbourn87].

Czasami stosuje się inne miary specyficzne dla przyjętego modelu sieci komputerowej. Na przykład dla modelu w postaci drzewa źródłowego (*rooted tree*) [Kershenbaum73] jako miary niezawodności przyjęto oczekiwaną liczbę węzłów komunikujących się z węzłem centralnym, czy oczekiwaną liczbę par węzłów komunikujących się poprzez węzeł centralny. Dla modelu niezawodnościowo-funkcjonalnego [Jóźwiak79, Jóźwiak92] zaproponowano względną przepustowość sieci. Metoda oceny względnej przepustowości sieci komputerowych o strukturach rozgałęzionych uwzględnia aspekty niezawodnościowe (uszkodzenia, przekłamania), funkcjonalne (przetwarzanie, transmisja) oraz strukturalne (liczba użytkowników, lokalnych mikrokomputerów, terminali) badanej sieci.

W przypadku deterministycznego modelu sieci stosowane miary wrażliwości na uszkodzenia (*vulnerability measures*) zwykle określają, jak wiele komponentów musi ulec uszkodzeniu, aby sieć nie mogła realizować swoich funkcji, żeby wymienić chociażby spójność wierzchołkową  $\kappa(G)$ , spójność krawędziową  $\lambda(G)$ , minimalny stopień wężła  $\delta(G)$  (które łączy zależność  $\kappa(G) \leq \lambda(G) \leq \delta(G) \leq \lfloor 2e/n \rfloor$ ), czy trwałość (*persistence*) definiowaną jako minimalna liczba łączy, które muszą być usunięte, aby zwiększyć średnicę sieci (maksymalny dystans pomiędzy węzłami) lub rozłączyć sieć [Boesch86].

Jakkolwiek niezawodność jest zwykle wyrażana za pomocą miar bazujących na spójności sieci i wysiłek badawczy w większości skoncentrował się na konsekwencjach uszkodzeń w postaci rozłączenia sieci, to jednak często konsekwencje są mniej katastrofalne i pojawiają się w postaci zredukowanej wydajności (*performance*)<sup>9</sup>. Ponadto nie zawsze miary bazujące na spójności sieci są najodpowiedniejsze. W takich przypadkach rozpatruje się miary zdolności systemu do prawidłowego działania<sup>10</sup> (*performability measures*).

Dla sieci o niezawodnych węzłach  $V$  i zawodnych łączach  $E$ , każdy stan  $S \subseteq E$  ma przypisane prawdopodobieństwo  $\Pr[S]$  wystąpienia i wartość  $\psi(S)$  pewnej miary wydajności  $\psi$ . Oczekiwana wydajność (*expected performance*) określona wzorem:

$$Perf(G_K) = \sum_{S \subseteq E} \Pr[S] \psi(S),$$

jest jedną z najpowszechniej wykorzystywanych miar zdolności systemu do prawidłowego działania<sup>11</sup>.

### 1.3.1. Wybór odpowiedniej miary oceny niezawodności sieci

Istotną kwestią jest, którą ze znanych miar oceny niezawodności sieci wybrać w konkretnej sytuacji. Miary bazujące na spójności są stosowane we wszystkich sytuacjach, w których wydajność sieci wydaje się być wystarczająca tak długo, jak długo sieć jest spójna. W szczególności sieci światłowodowe spełniają ten warunek. Miary spójności są użyteczne

<sup>9</sup> Por. [Colbourn98], s.2.

<sup>10</sup> *Performability* to zdolność prawidłowego działania. Por. *Słownik naukowo-techniczny angielsko-polski*, red. Sergiusz Czerni i Maria Skrzyńska, WNT 1983, s.563. Termin *performability* powstał ze złożenia terminów *performance* i *reliability*. Por. [Colbourn98], s.2.

<sup>11</sup> Por. [Colbourn98], s.2.



także w przypadku pomiaru prawdopodobieństwa, że sieć jest zdolna zagwarantować pewien minimalny poziom usług. Miary wydajności są stosowane tam, gdzie uszkodzenia elementów mogą obniżyć wydajność systemu do niedopuszczalnie niskich poziomów, podczas gdy sieć pozostaje spójna.

Wybór pomiędzy niezawodnością dwóch terminali,  $K$ -terminali i wszystkich terminali zależy od perspektywy, z której obserwujemy system. Niezawodność dwóch terminali określa zdolność sieci do spełnienia potrzeb komunikacji pomiędzy dwoma węzłami mierząc prawdopodobieństwo, że wiadomość pomiędzy tymi wyspecyfikowanymi węzłami może być przesłana (istnieje ścieżka łącząca oba węzły). Nie ma znaczenia czy węzły te, w przypadku sieci komputerowej, reprezentują *hosty*, *switche* czy *routery*. Niezawodność wszystkich terminali umożliwia ocenę systemu nie z punktu widzenia użytkownika, lecz dostawcy usług sieciowych. Niezawodność  $K$ -terminali jest najbardziej elastyczna, gdyż zawiera w sobie oba skrajne przypadki, umożliwiając ocenę systemu z punktu widzenia różnych grup użytkowników.



## 1.4. ZAKRES ZASTOSOWAŃ MIAR NIEZAWODNOŚCI BAZUJĄCYCH NA SPÓJNOŚCI SIECI

Miary niezawodności bazujące na spójności sieci znajdują zastosowanie w analizie niezawodności różnego rodzaju sieci (w tym także komputerowych) [Ball95, Frank72].

### 1.4.1. Poziom kręgosłupowy sieci z przełączaniem pakietów

W rozległych sieciach komputerowych stosuje się technikę komutacji pakietów lub jej modyfikacje (komutacja ramek lub komórek) dostosowane do coraz bardziej nowoczesnych środków transmisji danych [Kasprzak97, Tanenbaum88]. Rozległa sieć komputerowa z komutacją pakietów udostępnia wiele różnych, możliwych do wykorzystania tras, którymi pakiet może być przesłany od węzła źródłowego do węzła docelowego. Do najbardziej rozpowszechnionych protokołów określających sposób wymiany informacji w sieci pakietowej<sup>12</sup> można zaliczyć protokoły X.25, Frame Relay, TCP/IP oraz ATM. Sieć ARPANET, przedstawiona w załączniku 1, była pierwszą dużą siecią z komutacją pakietów

<sup>12</sup> Por. [Kasprzak97], s.25.

łączącą ośrodki uniwersyteckie w Stanach Zjednoczonych. Gwałtowny rozwój badań nad niezawodnością sieci od początku lat siedemdziesiątych był motywowany powstaniem tej sieci. Większość wykorzystywanych miar niezawodności to miary bazujące na spójności grafu reprezentującego badaną sieć. Jest to o tyle usprawiedliwione, że w większości sieci stosowana jest technika dynamicznego wyznaczania tras (*dynamic routing*). W razie uszkodzenia któregoś łącza pakiety mogą być przesłane wykorzystując inne możliwe trasy pod warunkiem, że takowe istnieją (tzn., że sieć jest spójna).

#### 1.4.2. Poziom kręgosłupowy sieci z przełączaniem kanałów

Pionierskie prace nad niezawodnością sieci dotyczyły sieci z komutacją kanałów [Lee55]. W tego typu sieciach przesłanie danych pomiędzy dwoma użytkownikami jest możliwe dopiero po ustanowieniu połączenia. Kanały komunikacyjne są dedykowane parze użytkowników na czas wymiany informacji. Oczywiście komunikacja pomiędzy użytkownikami jest niemożliwa jeżeli sieć przestanie być spójna. Ponadto może zaistnieć sytuacja, w której żądanie połączenia jest blokowane (np. sygnał „zajętości” w sieci telefonicznej), ponieważ nie można ustanowić nowego połączenia. Łącza sieciowe definiowane są jako uszkodzone jeżeli są zablokowane. W analizie tego typu sieci miary bazujące na spójności są używane razem z powyższą definicją uszkodzenia<sup>13</sup>.

#### 1.4.3. Inne zastosowania

Wydaje się, że bazujące na spójności miary niezawodności sieci można wykorzystać także przy analizie niezawodności systemów komputerowych tolerujących uszkodzenia, systemów o architekturze równoległej, a także sieci telekomunikacyjnych (szczególnie światłowodowych)<sup>14</sup>. Światłowodowe sieci telekomunikacyjne ze względu na wiele zalet są coraz częściej wprowadzane w miejsce tradycyjnych sieci. Ze względu na bardzo dużą pojemność łączy światłowodowych sieci realizowane na ich bazie wymagają dużo mniejszej liczby łączy niż sieci tradycyjne. Konsekwencją tego jest rosnąca konieczność uwzględnienia zagadnień niezawodnościowych przy projektowaniu takich sieci.

---

<sup>13</sup> Por. [Ball95], s.5.

<sup>14</sup> Por. [Ball95], s.7-8.

Jakkolwiek miary niezawodności bazujące na spójności sieci są powszechnie używanymi surogatami niezawodności sieci, to jednak w niektórych przypadkach odpowiedniejsze będzie wykorzystanie miar zdolności do prawidłowego działania (*performability*). Zaproponowany w niniejszej pracy algorytm obliczania niezawodności  $K$ -terminali można w stosunkowo prosty sposób przystosować do obliczania oczekiwanej wydajności, będącej jedną z najpopularniejszych miar zdolności do prawidłowego działania<sup>15</sup>.

## 1.5. ZŁOŻONOŚĆ OBLICZENIOWA

Naturalnym sposobem oceny złożoności obliczeniowej problemów jest wykorzystanie teorii NP-zupełności. Ponieważ jednak wymaga to sztucznego przeformułowania problemów obliczania miar niezawodności na problemy decyzyjne, Valiant [Valiant79a] zaproponował analogiczną teorię #P-zupełności. Znajduje ona zastosowanie w odniesieniu do problemów polegających na wyliczeniu rozwiązań (*counting problems*). Wykazano, że problemy obliczania miar niezawodności sieci (niezawodność  $K$ -terminali, niezawodność dwóch terminali i niezawodność wszystkich terminali) należą do klasy problemów NP-trudnych i #P-zupełnych [Valiant79b, Ball80, Ball83, Ball86] nawet w przypadku grafów planarnych [Provan86]. Oznacza to, że najprawdopodobniej nie istnieje algorytm, który będzie znajdował dokładne rozwiązanie, dla wszystkich możliwych wystąpień problemu w czasie wielomianowym w funkcji rozmiaru problemu. Dla problemów #P-zupełnych nie są znane deterministyczne schematy aproksymacyjne<sup>16</sup>. Są natomiast znane probabilistyczne wersje takich schematów [Karger95b, Karger97]. Z drugiej strony algorytm znajdujący dokładne rozwiązanie – mający w najgorszym przypadku złożoność wykładniczą – w wielu przypadkach może być efektywny.

## 1.6. CEL I ZAKRES PRACY

Spośród algorytmów dokładnych rozwiązujących problem niezawodności  $K$ -terminali szczególnie dużo uwagi poświęcono algorytmom faktoryzacji. Wniesiono wielki wkład w udoskonalenie pierwotnej wersji algorytmu faktoryzacji poprzez zastosowanie zachowujących

---

<sup>15</sup> Zob. rozdział 9 zawierający propozycje dalszych badań.

<sup>16</sup> Por. [Motwani95], s.308 i 307.

niezawodność redukcji, czy efektywnych strategii selekcji krawędzi grafu [Choi95, Deo92, Lai94, Page88, Page89, ResendeM86, ResendeL88, Satyanarayana83, Theologou91, Wood85, Wood86, Wood89]. W wielu przypadkach ze względu na trudność analizy złożoności obliczeniowej zaproponowanych algorytmów, ich efektywność jest badana jedynie eksperymentalnie. W przypadku niektórych algorytmów możliwa jest jednak analiza pesymistycznej złożoności obliczeniowej. Najmniejszą pesymistyczną złożonością czasową charakteryzuje się algorytm opisany przez Wooda [Wood85, Wood86]. W przypadku tego algorytmu maksymalna liczba liści binarnego drzewa obliczeń równa  $(|V| - 2)!$  jest osiągalna dla przypadków granicznych, gdy  $2 \leq |K| \leq 5$  oraz  $|V| - 2 \leq |K| \leq |V|$ . Znany są również algorytmy faktoryzacji [Satyanarayana83, Wood86], których maksymalna liczba liści binarnego drzewa obliczeń jest równa  $(|V| - 1)!$  dla dowolnego zbioru  $K$  ( $2 \leq |K| \leq |V|$ ). Jednakże ze względu na konieczność analizy coraz bardziej skomplikowanych sieci, efektywność tych algorytmów jest wciąż niewystarczająca. **Z tego względu celem niniejszej pracy jest zaproponowanie modyfikacji algorytmu faktoryzacji, które zwiększałyby jego efektywność.**

Konstruując układ niniejszej rozprawy doktorskiej skorzystano z propozycji redagowania rozpraw naukowych przedstawionych przez prof. Chinnecka [Chinneck97].

Praca zawiera osiem rozdziałów. W rozdziale pierwszym przedstawiono uzasadnienie wyboru tematu pracy, podstawowe pojęcia z teorii grafów, stosowany model sieci, miary oceny niezawodności sieci i zakres ich zastosowań, złożoność badanych problemów oraz cel i zakres pracy.

W rozdziale drugim przedstawiono przegląd ważniejszych metod obliczania miar niezawodności sieci i aktualny stan badań.

Dokładne omówienie algorytmu faktoryzacji jak również często wykorzystywanych zachowujących niezawodność redukcji można znaleźć w rozdziale trzecim.

Rozdział czwarty dotyczy problemu analizy pesymistycznej złożoności obliczeniowej algorytmów faktoryzacji. Przedstawiono dotychczasowe osiągnięcia w dziedzinie analizy pesymistycznej złożoności algorytmów faktoryzacji, uzyskane z wykorzystaniem niezmienników grafów.

W rozdziale piątym przedstawiono algorytm faktoryzacji *Fact* o maksymalnej liczbie liści binarnego drzewa obliczeń równej  $(|V| - 2)!$  dla dowolnego zbioru  $K$  ( $2 \leq |K| \leq |V|$ ). W

rozdziale tym zaprezentowano również odpowiednią metodę analizy pesymistycznej złożoności obliczeniowej algorytmu *Fact* i jego modyfikacji.

Zaproponowana przez autora modyfikacja algorytmu faktoryzacji *Fact&Cache* wykorzystująca wykrywanie równoważnych podproblemów została omówiona w rozdziale szóstym.

W rozdziale siódmym dokonano porównania efektywności różnych algorytmów obliczania niezawodności  $K$ -terminali wykorzystując zarówno rezultaty uzyskane na drodze analitycznej, jak i eksperymentalnej. Porównano otrzymane analitycznie i eksperymentalnie miary pesymistycznej złożoności obliczeniowej algorytmów faktoryzacji. Przedstawiono również uzyskane eksperymentalnie (z wykorzystaniem rozpatrywanych w literaturze sieci) wyniki pomiarów efektywności zaproponowanego algorytmu *Fact&Cache* na tle wyników uzyskanych przez inne, aktualnie najefektywniejsze, algorytmy obliczania niezawodności  $K$ -terminali.

Wnioski i podsumowanie rezultatów pracy, zagadnienia związane z projektowaniem niezawodnych sieci z wykorzystaniem algorytmów faktoryzacji (w szczególności algorytmu *Fact&Cache*), a także propozycje dalszych badań przedstawiono w rozdziale ósmym.

Indeks oznaczeń, skrótów i stałych numerycznych, zestawienie typowych sieci rozważanych w literaturze, jak również w niniejszej pracy zaprezentowano na końcu pracy.

## 2. PRZEGLĄD WAŻNIEJSZYCH METOD OBLICZANIA MIAR NIEZAWODNOŚCI SIECI

W rozdziale tym zostanie przedstawiony stan badań dotyczący algorytmów obliczania miar niezawodności sieci począwszy od metod znajdujących dokładne rozwiązanie, poprzez symulacje Monte Carlo, metody szacowania niezawodności sieci poprzez konstruowanie ograniczeń na niezawodność sieci, po inne – rzadziej stosowane – metody np. probabilistyczne algorytmy aproksymacyjne. Szczegółowe omówienie zagadnień analizy niezawodności sieci – za wyjątkiem probabilistycznych algorytmów aproksymacyjnych – zostało zaprezentowane w pracy Balla, Colbourn i Provana [Ball95].

### 2.1. ALGORYTMY ZNAJDUJĄCE DOKŁADNE ROZWIĄZANIE

Jak pokazano w rozdziale 1.5, rozważane problemy obliczania miar niezawodności sieci (zaprezentowanych w rozdziale 1.3) należą do klasy problemów #P-zupełnych. Z tego względu spośród algorytmów znajdujących dokładne rozwiązania można wyodrębnić dwie kategorie: algorytmy o wykładniczej (w najgorszym przypadku) złożoności czasowej, umożliwiające obliczanie niezawodności sieci o dowolnych strukturach i algorytmy o wielomianowej złożoności czasowej, umożliwiające obliczanie niezawodności tylko pewnych klas sieci. Omówienie niektórych metod można znaleźć w pracach [Agrawal84, Satyanarayana83, Shier91]. Przegląd i szersze omówienie różnych metod można znaleźć w pracy [Ball95].

#### 2.1.1. Efektywne algorytmy dla pewnych klas sieci i zachowujące niezawodność redukcje grafu

Idealnie byłoby znać efektywne (wielomianowe) algorytmy obliczania miar niezawodności dowolnych sieci. Jednak ze względu na złożoność rozpatrywanych problemów, mało prawdopodobne jest otrzymanie efektywnych algorytmów dla dowolnych sieci. Znane są natomiast wielomianowe algorytmy dla pewnych klas grafów. Klasy grafów, dla których znane są efektywne, dokładne algorytmy obliczania miar niezawodności sieci są jednak nieliczne. W większości przypadków algorytmy te wykorzystują zachowujące

niezawodność redukcje grafu. Dokładne omówienie najbardziej znanych zachowujących niezawodność redukcji grafu można znaleźć w rozdziale 3.1.

Satyanarayana i Wood [Satyanarayana85] zaproponowali algorytm o złożoności wielomianowej do obliczania niezawodności  $K$ -terminali sieci szeregowo-równoległych. Wykorzystali oni redukcje szeregowę, równoległą, stopnia drugiego i redukcje wielokąta do łańcucha. Ponadto Politof i Satyanarayana [Politof86] zaproponowali wielomianowy algorytm obliczania niezawodności pewnych podklas sieci planarnych.

### 2.1.2. Metoda wyliczania przestrzeni stanów (state space enumeration)

Najbardziej fundamentalną metodą obliczania niezawodności, wykorzystywaną już przez Moora i Shannona [Moore56], jest zliczanie przestrzeni stanów (*state space enumeration*). Stan sieci reprezentowanej przez graf  $G = (V, E)$  definiuje podzbiór  $S \subseteq E$  funkcjonujących krawędzi grafu. Niezawodność  $K$ -terminali sieci reprezentowanej poprzez graf  $G$  można wyrazić następująco:

$$R(G_K) = \sum_{S \in D} I_K(S) \Pr(S),$$

gdzie:

$D$  jest zbiorem wszystkich stanów sieci;

$I_K(S)$  jest zmienną zero-jedynkową przyjmującą wartość jeden tylko wtedy, gdy podzbiór  $S$  funkcjonujących krawędzi grafu gwarantuje, że węzły ze zbioru  $K$  są połączone.

$\Pr(S) = \prod_{e \in S} p_e \prod_{e \notin S} (1 - p_e)$  jest prawdopodobieństwem wystąpienia stanu  $S$  (pod warunkiem, że uszkodzenia krawędzi są niezależne).

Ponieważ istotne są tylko te stany, dla których  $I_K(S) = 1$  nie jest konieczne sprawdzanie wszystkich możliwych stanów sieci  $D$ . Dlatego kolejne metody wykorzystują zbiory sprawności (*success sets*) ewentualnie niesprawności (*failure sets*) [Satyanarayana83] sieci reprezentowanej przez graf  $G$  i podzbiór węzłów  $K$ . Zbiór sprawności jest minimalnym zbiorem krawędzi grafu  $G$ , który zapewnia, że węzły ze zbioru  $K$  są połączone. Zbiór ten jest w tym sensie minimalny, że usunięcie dowolnej krawędzi sprawia, iż węzły ze zbioru  $K$  przestają być połączone. Zbiór niesprawności jest definiowany analogicznie.



Niezawodność można wyrazić wzorem:

$$R(G_K) = \Pr\{f_1 \cup f_2 \cup \dots \cup f_p\} = 1 - \Pr\{\overline{f_1} \cup \overline{f_2} \cup \dots \cup \overline{f_p}\}, \quad (2.1)$$

gdzie:

$f_1, f_2, \dots, f_p$  reprezentują zbiory sprawności, a

$\overline{f_1}, \overline{f_2}, \dots, \overline{f_p}$  reprezentują zbiory niesprawności.

Topologicznie, zbiór sprawności odpowiada minimalnemu drzewu grafu  $G$  zawierającemu wszystkie węzły ze zbioru  $K$ , czyli  $K$ -drzewu<sup>17</sup> (w przypadku obliczania niezawodności dwóch terminali zbiorami sprawności/niesprawności będą minimalne ścieżki/cięcia grafu  $G$ ).

Ponieważ zdarzenia  $f_i$  w ogólnym przypadku niestety nie są rozłączne, kolejne dwie metody umożliwiają w praktyce obliczenie prawdopodobieństwa sumy zdarzeń.

### 2.1.3. Metody bazujące na zbiorach sprawności ( $K$ -drzewach)

#### 2.1.3.1. Metoda zawierania-wykluczania (*inclusion-exclusion formula*)

Metoda zawierania-wykluczania umożliwia praktyczne wyliczenie niezawodności w następujący sposób:

$$R(G_K) = \sum_i \Pr\{f_i\} - \sum_{i < j} \Pr\{f_i \cap f_j\} + \sum_{i < j < m} \Pr\{f_i \cap f_j \cap f_m\} - \dots \quad (2.2)$$

W równaniu (2.2) jest aż  $2^p - 1$  wyrażeń, gdzie  $p$  jest liczbą  $K$ -drzew. Metoda ta okazuje się więc nieefektywna już dla sieci o średnim rozmiarze<sup>18</sup>. Istotnym problemem związanym z zastosowaniem tej metody jest także pojawianie się par identycznych wyrażeń o przeciwnych znakach, które można odrzucić. Satyanarayana, Prabhakar i Hagstrom [Satyanarayana78, Satyanarayana80, Satyanarayana81a, Satyanarayana81b, Satyanarayana82b] zaproponowali specjalną formę zredukowanego wyrażenia zawierania-wykluczania, w której nie występują

---

<sup>17</sup> Por. rozdział 1.1.

<sup>18</sup> Por. [Satyanarayana83], s.109.

identyczne wyrażenia o przeciwnych znakach. Efektywność tej metody jest jednak również ograniczona<sup>19</sup>.

#### 2.1.3.2. Metoda rozłącznych iloczynów (sum of disjoint products)

Inną metodą obliczania prawdopodobieństwa sumy zdarzeń we wzorze (2.1) jest metoda rozłącznych iloczynów. Metoda ta dekomponuje sumę zdarzeń ze wzoru (2.1) na sumę zdarzeń rozłącznych:

$$R(G_K) = \Pr\{f_1\} + \Pr\{\overline{f_1} \cap f_2\} + \dots + \Pr\{\overline{f_1} \cap \overline{f_2} \cap \dots \cap \overline{f_{p-1}} \cap f_p\} \quad (2.3)$$

Chociaż metoda ta wymaga tylko dodania  $p$  wyrażen (prawdopodobieństw), to czas potrzebny na obliczenie wartości każdego wyrażenia może być wykładniczy w funkcji  $p$ <sup>20</sup>. Większość metod wykorzystujących algebrę boolowską do obliczania niezawodności  $R(G_K)$  stosuje tę metodę [Fratta73, Aggarwal75, Abraham79].

#### 2.1.4. Algorytm faktoryzacji

Omówione dotychczas metody bazują na koncepcji zbiorów stanów, w których graf  $G$  jest sprawny bądź niesprawny. Ponieważ zbiory te są określone poprzez kolekcje łączy, mamy do czynienia ze zdarzeniami złożonymi. W przeciwieństwie do tych metod twierdzenie faktoryzacji wykorzystuje zdarzenia elementarne sprawności lub niesprawności pojedynczego łącza. Metoda faktoryzacji została najprawdopodobniej po raz pierwszy użyta przez Moskowitza [Moskowitz58]. Spośród metod znajdujących dokładne rozwiązanie problemu niezawodności  $K$ -terminali, szczególnie dużo uwagi poświęcono właśnie tej metodzie. Algorytm faktoryzacji, jak i próby udoskonalenia pierwotnej wersji algorytmu (np. poprzez zastosowanie zachowujących niezawodność redukcji, czy efektywnych strategii selekcji krawędzi grafu, czyli dekompozycji problemu), stały się tematem wielu prac [Choi95, Deo92, Lai94, Page88, Page89, ResendeM86, ResendeL88, Satyanarayana83, Theologou91, Wood85, Wood85, Wood89]. W niektórych przypadkach algorytm faktoryzacji stał się częścią składową bardziej rozbudowanych (hybrydowych) algorytmów [Beichelt91a, Ng91].

<sup>19</sup> Tamże s.109.

<sup>20</sup> Por. [Agrawal84], s.482.

Dokładne omówienie algorytmu faktoryzacji jak i znanych z literatury propozycji jego usprawnienia można znaleźć w rozdziale 3. Algorytm faktoryzacji wykorzystujący zachowujące niezawodność redukcje grafu, choć uważany za jeden z najefektywniejszych, okazuje się niewystarczający ze względu na konieczność analizy coraz bardziej skomplikowanych sieci. Dlatego w kolejnych rozdziałach zostaną przedstawione nowe, zaproponowane przez autora, propozycje usprawnienia tego algorytmu.

### 2.1.5. Algorytm dekompozycji grafu

Technika dekompozycji została zaproponowana przez Rosenthala [Rosenthal77]. Uogólnił on technikę 2-spójnej dekompozycji (*biconnected decomposition*), która polega na tym, że jeżeli graf  $G$  ma węzeł rozdzielający, to graf ten można zdekomponować na dwa grafy  $H$  i  $L$  tak, że  $R(G) = R(H)R(L)$ .

Jeżeli  $H = (V', E')$  będzie dowolną podsiecią  $G = (V, E)$  taką, że  $V' \subseteq V$  i  $E' = E \cap (V' \times V')$  a ponadto  $L = (V'', E'')$  będzie dopełnieniem grafu  $H$  w  $G$  ( $V' \cup V'' = V$ ;  $E' \cup E'' = E$  oraz  $E' \cap E'' = \emptyset$ ), to zbiór węzłów  $F = V' \cap V''$  będzie nazywany zbiorem granicznym<sup>21</sup> (*boundary set*) bądź zbiorem rozdzielającym<sup>22</sup> (*separating node set*). Zbiór ten rozdziela grafy  $H$  i  $L$ . Rosenthal zaproponował przypisać grafom  $H$  i  $L$  pewne klasy (kolekcje stanów), których liczba zależy od licznosci zbioru granicznego  $F$ , rozpatrywanej miary niezawodności (licznosci zbioru  $K$ ) i zastosowanego modelu sieci (ewentualnego rozszerzenia modelu na uszkodzenia węzłów). Stany wchodzące w skład dowolnej klasy nazywane są stanami równoważnymi. Niezawodność jest obliczana przez skojarzenie klas grafu  $H$  i  $L$ .

Rosenthal twierdzi, że metoda dekompozycji jest efektywna pod warunkiem, że sieć nie jest silnie spójna<sup>23</sup>. W istocie decydującym o efektywności parametrem jest rozmiar (licznosc) wykorzystywanych zbiorów rozdzielających, ponieważ liczba klas rośnie wykładniczo z rozmiarem zbiorów rozdzielających<sup>24</sup>. Beichelt i Tittman<sup>25</sup> [Beichelt91a]

---

<sup>21</sup> Por. [Carlier96], s.144.

<sup>22</sup> Por. [Beichelt91a], s.198 i [Beichelt91b], s.870.

<sup>23</sup> Por. [Rosenthal77], s.391.

<sup>24</sup> Por. [Carlier96], s.149-151.

zaproponowali algorytm będący kombinacją algorytmu dekompozycji i faktoryzacji [Wood86] do obliczania niezawodności wszystkich terminali. Zaimplementowany przez nich algorytm okazał się dla rozpatrywanej przez nich sieci PLG znacznie efektywniejszy niż algorytm faktoryzacji Wooda. Ci sami autorzy zaproponowali metodę dekompozycji także dla obliczania niezawodności dwóch terminali [Beichelt91b]. Carlier<sup>26</sup> i Lucet<sup>27</sup> [Carlier96] zaprezentowali algorytm dekompozycji do obliczania niezawodności  $K$ -terminali. Algorytm ten wykorzystuje wielokrotnie heurystyczną procedurę numerowania wierzchołków grafu, gdyż rozmiar największego zbioru rozdzielającego (a więc również efektywność algorytmu) zależy od numeracji wierzchołków. Istotnym ograniczeniem metody dekompozycji jest duża ilość pamiętanych danych<sup>28</sup>. Algorytmy dekompozycji, choć złożone implementacyjnie, wydają się być aktualnie najszybszymi znanymi algorytmami obliczania niezawodności  $K$ -terminali. Dlatego efektywność tych algorytmów zostanie porównana z efektywnością algorytmu zaproponowanego przez autora.

## 2.2. METODY MONTE CARLO

Metody Monte Carlo cechuje szczególnie szeroki zakres zastosowań. Można je wykorzystać do rozwiązania większości problemów związanych z niezawodnością sieci. Atrakcyjność metod Monte Carlo wynika z faktu, że można oszacować niezawodność poprzez zaprojektowanie relatywnie prostego (szczególnie w przypadku standardowej metody Monte Carlo) planu próbkowania. Standardowa metoda Monte Carlo jest jednak bardzo nieefektywna. Wykorzystując bardziej zaawansowane techniki próbkowania (np. *antithetic variate*, *control variate*, *conditional sampling*, *importance sampling* i *stratified sampling*) można generować oszacowania ze znacząco mniejszym błędem przy tym samym lub mniejszym koszcie niż w metodzie standardowej. Ta obserwacja stała się motywem wielu prac, które można podzielić na kilka nurtów:

---

<sup>25</sup> Professor at the Department of Mathematics / Physics, and Computer Science of the University of Technology and Economics Mittweida (Hochschule für Technik und Wirtschaft Mittweida), e-mail: peter@htwm.de

<sup>26</sup> Professor Jacques Carlier, Universite de Technologie de Compiègne, e-mail: Jacques.Carlier@utc.fr

<sup>27</sup> Assistant professor at the Department of Information Processing Engineering, Universite de Technologie de Compiègne, e-mail: clucet@hds.univ-compiegne.fr lub Corinne.Lucet@utc.fr.

<sup>28</sup> Por. [Carlier96] s.153.

## 1. Metody szacowania niezawodności bazujące na konstruowaniu górnych i dolnych ograniczeń.

Fishman [Fishman86a], Van Slyke i Frank [Van Slyke72] oraz Kumamoto, Tanaka i Inoue [Kumamoto77] zaproponowali metody szacowania niezawodności poprzez konstruowanie górnych i dolnych ograniczeń na niezawodność. Ograniczenia skonstruowane w pierwszej z tych prac zawierają w sobie, jako specjalne przypadki, ograniczenia skonstruowane w pozostałych dwóch pracach<sup>29</sup>. Ponadto Fishman [Fishman89] wykorzystał próbkowanie *importance sampling* i *stratified sampling* do szacowania niezawodności dwóch terminali.

## 2. Metoda *dagger sampling*.

Kumamoto, Tanaka, Inoue i Henley [Kumamoto80] zaproponowali metodę redukcji wariancji *dagger sampling*. Metoda ta zapewnia redukcję wariancji, poprzez generację negatywnie skorelowanych wektorów stanów. Technika ta może być widziana jako „wielowymiarowe”<sup>30</sup> rozszerzenie techniki nazywanej w terminologii metod Monte Carlo *antithetic variates*. Metodę Monte Carlo wykorzystującą negatywne korelacje zaproponował również Khadiri i Rubino [Khadiri92a].

## 3. Metoda sekwencyjnej konstrukcji/destrukcji (*sequential construction/destruction*).

Easton i Wong [Easton80] zaproponowali metodę sekwencyjnej konstrukcji/destrukcji wykorzystującą właściwości permutacji stanów sprawności i niesprawności łączy. Na początku wszystkie łączy są uszkodzone/sprawne, a następnie sukcesywnie „naprawiane/uszkodzane”, jedno po drugim w określonym porządku, aż do momentu, kiedy system okaże się sprawny/niesprawny. Oszacowanie niezawodności jest funkcją długości procesu konstrukcji/destrukcji, po którym system okaże się sprawny/niesprawny. Metodę destrukcji preferuje się wtedy, gdy system może ulec uszkodzeniu przy relatywnie małej ilości uszkodzonych łączy. Metoda ta została później udoskonalona i jest znana jako metoda ewolucji grafu lub metoda *MP(merge process)* (por. punkt 5).

## 4. Metoda zbiorów niesprawności (*failure sets*).

Karp i Luby [Karp83] wykorzystali zbiory niesprawności sieci (*failure sets*) do szacowania niezawodności systemu (w istocie metoda ta wykorzystuje wyliczanie przekrojów grafu).

---

<sup>29</sup> Por. [Fishman86b], s.149.

<sup>30</sup> Por. [Ball95], s.43.

## 5. Metoda modeli ewolucji grafu zwana niekiedy *MP(merging process)*.

Elperin, Gertsbackh i Lomonosov [Elperin91] wykorzystują symulację Monte Carlo bazującą na modelach ewolucji grafu wprowadzając sztuczny czas ewolucji. Niezawodność sieci jest wyrażana w terminach dwóch różnych procesów Markowa (*DP-destruction process* i *CP-creation process*).

## 6. Metoda rekurencyjnej redukcji wariancji.

Spośród różnych metod redukcji wariancji do szacowania niezawodności sieci komunikacyjnych szczególnie interesującą grupę stanowią zaproponowane przez Cancela i Khadiri rekurencyjne algorytmy redukcji wariancji do szacowania niezawodności  $K$ -terminali [Cancela94] oraz niezawodności 2-terminali [Cancela95].

## 7. Metody szacowania współczynników wielomianu niezawodności<sup>31</sup>.

Niekiedy istotniejsze jest oszacowanie funkcjonalnej formy wielomianu niezawodności niż oszacowanie niezawodności dla konkretnego wektora prawdopodobieństw uszkodzenia łączy. Szczególnie istotne jest to w przypadku, gdy prawdopodobieństwa uszkodzenia łączy są takie same. Wtedy niezawodność systemu można zapisać w jednej z dwóch form wielomianowych:

$$R(G_K) = \sum_{i=0}^b F_i p^{b-i} (1-p)^i$$

$$= p^l \sum_{i=0}^{b-l} H_i (1-p)^i,$$

gdzie  $l = n - 1$  jest rozmiarem minimalnego drzewa rozpinającego. Wielomiany te są nazywane wielomianami niezawodności (*reliability polynomial*). Van Slyke i Frank [Van Slyke72] oraz Fishman [Fishman87] zaproponowali metody szacowania współczynników  $F_i$  dla problemu niezawodności  $K$ -terminali. Van Slyke i Frank użyli próbkowania *stratified sampling*, podczas gdy Fishman zaproponował rozszerzenie metody sekwencyjnej konstrukcji. Nel i Colbourn [Nel90] zaproponowali schemat szacowania współczynników  $H_i$  dla problemu niezawodności wszystkich terminali.

## 8. Inne metody.

Zaprezentowany przez Kargera [Karger95a, Karger95b, Karger97] probabilistyczny w pełni wielomianowy schemat aproksymacyjny FPTAS do obliczania niezawodności wszystkich

---

<sup>31</sup> Por. [Ball95], s.48-50.

terminali wykorzystywał między innymi metodę Monte Carlo (więcej informacji na temat tego algorytmu można znaleźć w rozdziale 2.4.).

Fishman [Fishman86b] dokonał porównania efektywności metod Monte Carlo wymienionych w punktach 1 do 4 zarówno pod kątem złożoności czasowej, jak i pamięciowej. Fishman uwzględnił również wrażliwość porównywanych metod na niezawodność elementów sieci. Prezentując również swoje rozwiązania, podobne porównania zamieścili także Elperin, Gertsbackh i Lomonosov [Elperin91] (metody 1 do 5) oraz Cancela i Khadiri [Cancela94] (metody 1 do 6). Z porównania dokonanego w ostatniej z tych prac wynikają następujące wnioski:

1. Metoda rekurencyjnej redukcji wariancji zdecydowanie przewyższa metody wymienione w punktach 1, 2, 3 i 5 bez względu na niezawodność elementów sieci.
2. Metoda zbiorów niesprawności wydaje się jednak konkurencyjna dla dużych niezawodności łączy (np.  $p=0.95$ ). Wadą tej metody jest jednak bardzo duża złożoność pamięciowa algorytmu (jak podano w [Fishman86b] lista przekrojów dla sieci „dodecahedron” o 20 węzłach i 30 łączach wymaga ponad 2 MB pamięci). Ponadto metoda ta nie gwarantuje redukcji wariancji.
3. Dla małych wartości niezawodności łączy (np.  $p=0.5$ ), jedynie metody rekurencyjnej redukcji wariancji i *dagger sampling* spisują się nieco lepiej niż standardowa metoda Monte Carlo.

Generalnie efektywność metod Monte Carlo zależy od niezawodności elementów sieci i mimo zastosowania metod redukcji wariancji, jest często niezadowalająca. Większość problemów związanych z niezawodnością doprowadza bowiem w efekcie do symulacji zdarzeń występujących niezwykle rzadko. Ponadto chociaż aproksymacja jest poprawna z dużym prawdopodobieństwem, to nie jest możliwa jej weryfikacja.

### 2.3. METODY KONSTRUOWANIA OGRANICZEŃ NA NIEZAWODNOŚĆ SIECI

Odrębną grupę stanowią metody szacowania niezawodności sieci poprzez konstruowanie górnych i dolnych ograniczeń na rozważaną miarę niezawodności sieci. Zwykle przyjmowane jest dodatkowe ograniczenie modelu podanego w rozdziale 1.2, że uszkodzenia wszystkich łączy są równie prawdopodobne.



Do konstruowania ograniczeń wykorzystywane są zarówno metody kombinatoryczne, jak i algebraiczne. W przypadku metod algebraicznych zamiast operować wielkościami numerycznymi, wykorzystuje się obiekty algebraiczne (wielomiany) i zdefiniowane na nich odpowiednie transformacje<sup>32</sup>. Metody algebraiczne szacowania niezawodności sieci są dobrze opisane w pracy Shiera [Shier91]. Kombinatoryczne metody szacowania niezawodności sieci poprzez konstruowanie ograniczeń na niezawodność sieci są szczegółowo opisane w książce Colbourn [Colbourn87]. Przegląd metod konstruowania ograniczeń można znaleźć również w pracy Balla, Colbourn i Provana [Ball95].

Wygodnie jest podzielić metody konstruowania ograniczeń w zależności od stosowanej miary oceny niezawodności sieci.

### 2.3.1. Ograniczenia na niezawodność wszystkich terminali

Zaproponowane ograniczenia na niezawodność wszystkich terminali można podzielić na następujące grupy<sup>33</sup>.

Pierwsza grupa ograniczeń znajduje zastosowanie nawet wtedy, gdy nie zakładamy statystycznej niezależności i równych prawdopodobieństw uszkodzenia łączy. Tego typu ograniczenia przedstawili Zemel [Zemel82] i Assous [Assous84]. Ograniczenia te dla większości sieci nie dostarczają żadnych informacji o ich niezawodności (dolne ograniczenie wynosi 0, górne ograniczenie wynosi 1). Umożliwiają jednak specyfikacje zależności statystycznych. Doprowadza to do uzyskania użytecznych ograniczeń, kiedy informacje o korelacji uszkodzeń par łączy są znane [Assous84, Carrasco85]. Tak skonstruowane ograniczenia są jednak słabsze od tych, które przyjmują statystyczną niezależność.

Druga grupa ograniczeń wykorzystuje pewne właściwości z teorii grafów. Polesskii [Polesskii71] zaproponował dolne ograniczenie, które zostało później poprawione przez Lomonosova i Polesskii'ego [Lomonosov72]. Używając innej strategii proponowali oni także górne ograniczenie [Lomonosov71].

Trzecia grupa ograniczeń wykorzystuje techniki numeryczne do liczenia działających podsieci. Jacobs [Jacobs59] zaproponował proste ograniczenia. Frank i Van Slyke [Van

---

<sup>32</sup> Por. [Shier91], s.22.

<sup>33</sup> Por. [Colbourn88], s.2-3.

Slyke72] znacznie je poprawili, wykorzystując kombinatoryczne rezultaty Kruskala [Kruskal63] i Katona [Katona66] (często nazywane ograniczeniami Kruskala-Katony). Ostatecznie jeszcze bardziej ulepszyli je Ball i Provan [Ball82]<sup>34</sup>.

Harms [Harms83] z kolei zaimplementował wszystkie wymienione ograniczenia i przeprowadził porównanie. Ograniczenia Balla-Provana były zawsze lepsze od ograniczeń Kruskala-Katony. Podobnie te ostatnie były zawsze lepsze od ograniczeń Jacobsa. Okazało się więc, że spośród ograniczeń zakwalifikowanych do trzeciej grupy, wystarczy brać pod uwagę jedynie ograniczenia Balla-Provana. Nieoczekiwanym rezultatem pracy Harmsa było spostrzeżenie, że proste ograniczenia Lomonosova-Polesskii'ego są w niektórych przypadkach lepsze niż zaproponowane przez Balla-Provana<sup>35</sup>. Zdarzało się to rzadko, niemniej jednak oznaczało, że ograniczenia Balla-Provana można poprawić. Dokonali tego Colbourn i Harms [Colbourn88]. Połączyli oni oba ograniczenia wykorzystując technikę programowania liniowego. Uzyskane ograniczenie jest co najmniej tak dobre, jak użyte ograniczenia składowe. Zaproponowana technika pozwala również dołączać inne na przykład nowo powstałe ograniczenia. Podejmowane są więc próby znalezienia dalszych możliwie dobrych ograniczeń [Brown93, Strayer95b, Strayer98].

### 2.3.2. Ograniczenia na niezawodność dwóch terminali

Jest wiele metod konstruowania ograniczeń na niezawodności dwóch terminali, jednak tylko kilka może być efektywnie wyliczonych. Ograniczenia Kruskala-Katony [Kruskal63, Katona66] znajdują zastosowanie zarówno w przypadku szacowania niezawodności wszystkich terminali, jak i niezawodności dwóch terminali. Brecht i Colbourn [Brecht85] zaproponowali dolne ograniczenia, bazujące na wzajemnie rozłącznych ścieżkach<sup>36</sup>. Podobnie jak w przypadku niezawodności wszystkich terminali tak i w przypadku niezawodności dwóch terminali można zastosować metodę uzyskania lepszych ograniczeń poprzez połączenie ograniczeń uzyskanych na bazie różnych technik [Colbourn88]. Harms i Colbourn w swojej pracy [Harms93] zaprezentowali efektywną metodę konstruowania górnego ograniczenia, jak również technikę poprawy górnych ograniczeń, uzyskanych za pomocą

---

<sup>34</sup> Por. [Brecht86], s.370.

<sup>35</sup> Por. [Colbourn88], s.3.

<sup>36</sup> Por. [Brecht86], s.370.

innych metod. Dolne ograniczenie na niezawodność dwóch terminali przedstawili także Chari i Provan [Chari90]. AboElFotouh i Colbourn [AboElFotouh89] zaproponowali dolne ograniczenie (szeregowo-równoległe) na niezawodność dwóch terminali. Ich strategia polegała na przekształceniu (wykorzystując rozłączne ścieżki i cięcia) rozpatrywanej sieci  $G_K$  na dwie inne sieci  $G'_{K'}$  i  $G''_{K''}$  w taki sposób, aby  $R(G'_{K'}) \leq R(G_K) \leq R(G''_{K''})$ .

Zastosowanie struktury algebraicznej właściwej dla szacowania niezawodności dwóch terminali można znaleźć w pracach Shiera [Shier85, Shier87]. W jednej ze swoich prac [Shier91] w rozdziale 3 przedstawił on sposób algebraicznego sformułowania problemu niezawodności dwóch terminali, a w rozdziale 4.2 i 4.3 sposób konstruowania ograniczeń i obliczania niezawodności dwóch terminali wykorzystując techniki algebraiczne. Shier jest również autorem bądź współautorem innych prac dotyczących algebraicznych metod konstruowania ograniczeń [Shier88a, Shier88b, Shier89].

### 2.3.3. Ograniczenia na niezawodność $K$ -terminali

Ograniczenia na niezawodność  $K$ -terminali można uzyskać poprzez modyfikację ograniczeń Kruskala-Katony. Brecht i Colbourn [Brecht86, Brecht89] zaproponowali nierówności addytywne i multiplikatywne, które umożliwiały uzyskanie ograniczeń na niezawodność  $K$ -terminali. Odmienne rozwiązanie zaproponowali Strayer i Colbourn [Strayer95a].

## 2.4. INNE METODY

Li i Silvester [Li84] przedstawili ideę generowania najbardziej prawdopodobnych stanów. Metoda ta, nazywana niekiedy metodą aproksymacji przestrzeni stanów (*state-space approximation*), była rozwijana między innymi przez Shiera<sup>37</sup>.

Karger [Karger95a, Karger95b, Karger97] zaprezentował probabilistyczny w pełni wielomianowy schemat aproksymacyjny FPTAS<sup>38</sup> do obliczania niezawodności wszystkich

---

<sup>37</sup> Por. [Shier91], rozdział 8.

<sup>38</sup> Więcej informacji na temat algorytmów aproksymacyjnych można znaleźć w raporcie opracowanym przez Motwaniego [Motwani92], na bazie którego przygotowana jest książka. Ten sam autor pisze na temat algorytmów probabilistycznych w książce [Motwani95].

terminali. Jakkolwiek powyższa technika tego nie wymusza, autor poczynił dodatkowe założenie, że uszkodzenia wszystkich łączy są równie prawdopodobne ( $\forall i \in E \quad q_i = q$ ). Dla dużych wartości  $q_i$  algorytm szacuje niezawodność sieci poprzez symulację Monte Carlo. Dla małych wartości  $q_i$  autor przyjmuje, że tylko niewielkie cięcia (*small cuts*) w grafie ulegają uszkodzeniu. Wylicza te cięcia używając zaproponowanego przez siebie algorytmu RCA(*Recursive Contraction Algorithm*) [Karger95c] i znajduje prawdopodobieństwo, że jedno z nich ulega uszkodzeniu używając algorytmu SACA(*Self Adjusting Coverage Algorithm*) [Karp89].

## 2.5. PODSUMOWANIE STANU BADAŃ

Ze względu na fakt, iż rozpatrywane problemy niezawodnościowe należą do klasy problemów #P-zupełnych, podejmowane są próby zastosowania algorytmów, które dostarczają niekoniecznie dokładnych rozwiązań, ale są efektywne. Dwa główne nurty to estymacja niezawodności poprzez zastosowanie metod Monte Carlo i konstruowanie dolnych i górnych ograniczeń na niezawodność. Metody Monte Carlo w celu otrzymania maksymalnie dokładnego oszacowania rozważanej miary niezawodności sprawdzają niewielką część stanów. Stany te są wybierane losowo w odróżnieniu od aktualnie stosowanych technik konstruowania ograniczeń. Nie zawsze jednak klasyfikacja metod jest prosta, gdyż na przykład wiele metod Monte Carlo wykorzystuje techniki konstruowania ograniczeń w celu zmniejszenia rozmiaru przestrzeni stanów<sup>39</sup>, a probabilistyczne algorytmy aproksymacyjne wykorzystują metody Monte Carlo [Karger95b, Karger97].

Metody Monte Carlo cechuje szeroki zakres zastosowań, ale ich efektywność pomimo zastosowania metod redukcji wariancji, jest często niezadowalająca. Większość problemów związanych z niezawodnością doprowadza bowiem w efekcie do symulacji zdarzeń występujących niezwykle rzadko. Efektywność metod Monte Carlo jest również uzależniona od niezawodności elementów sieci. Ponadto chociaż oszacowanie jest poprawne z pewnym określonym prawdopodobieństwem, to nie jest możliwa jego weryfikacja.

Z kolei jak zauważa Colbourn<sup>40</sup> metody konstruowania dolnych i górnych ograniczeń na niezawodność sieci istnieją tylko dla pewnych klas problemów, a jakość konstruowanych

---

<sup>39</sup> Por. [Ball95], s.26 oraz [Nel90].

<sup>40</sup> Por. [Colbourn98], s.10.

ograniczeń i efektywność tych algorytmów zależy od klasy problemu. Ponadto, jeżeli w wyniku zastosowania tych metod nie uzyskamy wystarczającej dokładności oszacowania niezawodności, to na ogół nie ma możliwości jej poprawy na przykład kosztem dłuższego czasu obliczeń<sup>41</sup>.

Zaproponowany niedawno przez Kargera [Karger95b, Karger97] probabilistyczny schemat aproksymacyjny FPTAS do szacowania niezawodności wszystkich terminali charakteryzuje teoretyczna złożoność czasowa  $O(n^{4.5})$ . Chociaż algorytm ten należy do klasy algorytmów wielomianowych, to jego efektywność (nawet w przypadku sieci o prostej strukturze) wydaje się w praktyce niezbyt wysoka. Świadczą o tym zamieszczone w pracy [Karger97] duże czasy realizacji<sup>42</sup> zaimplementowanego algorytmu zarówno dla sieci o skomplikowanej jak i dla tych o bardzo prostej strukturze (np. cykli). Ponadto, ze względu na wykorzystanie metody Monte Carlo, chociaż aproksymacja jest z dużym prawdopodobieństwem poprawna, to nie jest możliwa jej weryfikacja<sup>43</sup>.

Ze względu na powyższe ograniczenia wiele prac poświęconych jest algorytmom znajdującym dokładne rozwiązanie. W chwili obecnej najefektywniejsze wydają się być algorytmy wykorzystujące metodę dekompozycji [Beichelt91a, Beichelt91b, Carlier96]. Uważany dotąd za najefektywniejszy<sup>44</sup> algorytm faktoryzacji jest jednak znacznie prostszy zarówno koncepcyjnie jak i implementacyjnie. Z powyższych względów, a także z uwagi na dostrzeżone możliwości jego dalszych modyfikacji, algorytm faktoryzacji zostanie dokładniej omówiony.

---

<sup>41</sup> Por. [Colbourn98], s.6.

<sup>42</sup> Pomimo użycia wydajnej stacji roboczej SPARC 20. Por. [Karger97], s.334.

<sup>43</sup> Por. [Karger97], s.334.

<sup>44</sup> Por. [Shier91], s.18, [Theologou91], s.210 i [Wood85], s.175.

### 3. ALGORYTM FAKTORYZACJI

Algorytm faktoryzacji wykorzystuje zdarzenia elementarne sprawności lub niesprawności pojedynczej krawędzi. Stany grafu można podzielić na dwa zbiory ze względu na dwa możliwe stany krawędzi  $e_i$  o niezawodności  $p_i$ . Stąd niezawodność  $K$ -terminali można wyrazić w postaci prostej formuły niezawodności warunkowej, jako:

$$R(G_K) = p_i R(G_K | e_i \text{ funkcjonuje}) + (1 - p_i) R(G_K | e_i \text{ nie funkcjonuje}) \quad (3.1)$$

Równanie (3.1) jest znane jako dekompozycja osiowa (*pivotal decomposition*) i zostało po raz pierwszy zastosowane przez Moskowitza [Moskowitz58]<sup>45</sup>. Dla grafów nieskierowanych, których węzły są niezawodne, formuła ta ulega następującym uproszczeniom:

$R(G_K | e_i \text{ funkcjonuje})$  pokrywa się z  $R(G_{K'} * e_i)$ , gdzie  $G_{K'} * e_i$  jest grafem otrzymanym z grafu  $G_K$  przez usunięcie krawędzi  $e_i$  oraz połączenie jego wierzchołków końcowych,

$R(G_K | e_i \text{ nie funkcjonuje})$  odpowiada  $R(G_K - e_i)$ , gdzie  $G_K - e_i$  jest grafem otrzymanym z grafu  $G_K$  przez usunięcie krawędzi  $e_i$ , ale pozostawienie wierzchołków końcowych.

Twierdzenie faktoryzacji jest topologiczną interpretacją formuły (3.1) dla grafów nieskierowanych.

#### Twierdzenie 1 (Twierdzenie faktoryzacji)

Niezawodność  $K$ -terminali sieci probabilistycznej reprezentowanej poprzez graf  $G$  z wyróżnionym podzbiorem węzłów  $K$  można wyrazić następująco:

$$R(G_K) = p_i R(G_{K'} * e_i) + (1 - p_i) R(G_K - e_i), \quad (3.2)$$

gdzie:

$e_i$  – dowolna krawędź grafu  $G_K$ ;

---

<sup>45</sup> Por. [Satyanarayana83], s.109.

$$\begin{aligned}
G_{K'} * e_i &= (V - u - v + w, E - e_i), \quad w = u \cup v; \\
K' &= \begin{cases} K & \text{jeżeli } u, v \notin K; \\ K - u - v + w & \text{jeżeli } u \in K \text{ lub } v \in K; \end{cases} \\
G_K - e_i &= (V, E - e_i).
\end{aligned}$$

Obliczenia realizowane na skutek rekurencyjnego stosowania równania (3.1) lub (3.2) na każdym podproblemie można zaprezentować za pomocą binarnego drzewa obliczeń. Binarne drzewo obliczeń zawiera  $2^b$  liści<sup>46</sup>, co odpowiada wyliczaniu wszystkich możliwych stanów z przestrzeni stanów. Mine [Mine59], Wing i Demetriou [Wing64] oraz Brown [Brown71] wykorzystywali do obliczania niezawodności sieci probabilistycznych równanie (3.1). Metoda ta jest bardzo nieefektywna, gdyż spośród  $2^b$  liści wiele odpowiada stanom w których węzły zawarte w podzbiorze węzłów  $K$  są rozłączone. Można uniknąć generowania takich stanów, stosując odpowiednią strategię selekcji krawędzi, na których dokonywana jest faktoryzacja, czyli dekompozycji problemu. Sposób uzyskania optymalnego drzewa obliczeń można znaleźć w [Satyanarayana83, Wood85].

Wykorzystując twierdzenie faktoryzacji niezawodność  $K$ -terminali sieci reprezentowanej poprzez graf  $G_K$  można wyrazić w funkcji niezawodności dwóch prostszych sieci. Graf będący reprezentacją pierwszej sieci zawiera o jedną krawędź mniej niż graf  $G_K$ , zaś graf będący reprezentacją drugiej sieci zawiera o jedną krawędź i o jeden wierzchołek mniej niż graf  $G_K$ . Istotną zaletą równania (3.2) w stosunku do równania (3.1) jest możliwość występowania szeregowych i równoległych krawędzi w obu rozpatrywanych grafach  $G_{K'} * e_i$  oraz  $G_K - e_i$ . W takich sytuacjach można zastosować, wymagające jedynie wielomianowego czasu wykonania, zachowujące niezawodność redukcje grafu uzyskując znaczne zmniejszenie binarnego drzewa obliczeń.

Niezawodność  $K$ -terminali może być obliczona, dla dowolnej sieci, poprzez rekurencyjne wywoływanie formuły (3.2) i dokonywanie zachowujących niezawodność redukcji. Grafy rozpatrywane w wyniku rekurencyjnego wywoływania tej formuły są coraz prostsze. Procedura ta odpowiada konstruowaniu drzewa binarnego. Otrzymywane w ten sposób grafy mogą również podlegać zachowującym niezawodność redukcjom, co wpływa na znaczne zwiększenie efektywności algorytmu. W końcu grafy są redukowane do prostych struktur (np. pojedynczych krawędzi), dla których niezawodność można bardzo prosto

---

<sup>46</sup> Gdzie  $b$  jest liczbą krawędzi grafu.



obliczyć lub wierzchołki należące do zbioru  $K$  stają się rozłączone (niezawodność  $K$ -terminali sieci reprezentowanej poprzez taki graf wynosi zero). W ten sposób można obliczyć niezawodność  $K$ -terminali dowolnej sieci.

Od początku lat siedemdziesiątych wielokrotnie wykorzystano zarówno twierdzenie faktoryzacji, jak i zachowujące niezawodność redukcje grafu do efektywnego obliczania niezawodności sieci [Carlier91, Choi95, Deo92, Lai94, Page88, Page89, ResendeM86, ResendeL88, Satyanarayana83, Theologou91, Wood85, Wood86, Wood89]. Opracowano i udoskonalono zachowujące niezawodność redukcje grafu, w szczególności redukcje wielokąta do łańcucha (*polygon-to-chain reductions*) [Satyanarayana82a, Wood82, ResendeM86, Wood85, Lai94, Choi95]. Carlier i Theologou wykorzystywali dwuspójną dekompozycję [Carlier91, Theologou91]. Wood zaproponował technikę trójspójnej dekompozycji (*triconnected decomposition*) [Wood89] w celu dalszego zwiększenia efektywności algorytmu faktoryzacji. Deo i Medidi [Deo92] zaprojektowali i zaimplementowali zrównoleglone wersje najefektywniejszych ich zdaniem algorytmów sekwencyjnych w tym algorytmu faktoryzacji opisanego przez Page i Perry [Page88].

Theologou i Carlier [Theologou91] przedstawili modyfikację algorytmu faktoryzacji do efektywnego obliczania niezawodności  $K$ -terminali sieci, których nie tylko łączy, lecz także węzły są zawodne. Zaproponowana przez nich modyfikacja algorytmu faktoryzacji jest o tyle istotna, że stosunkowo niskim kosztem zwiększa obszar zastosowań algorytmów faktoryzacji. Zaprezentowany algorytm wykorzystywał dwuspójną dekompozycję oraz proste redukcje grafu (szeregowę, równoległą i stopnia drugiego) bez redukcji wielokąta do łańcucha, których autorzy nie potrafili zastosować do sieci o zawodnych węzłach. Sposób wykorzystania redukcji wielokąta do łańcucha również w przypadku zawodnych węzłów zaproponował kilka lat później Lai [Lai94].

Technika aproksymacji na bazie algorytmu faktoryzacji została najprawdopodobniej po raz pierwszy zaproponowana przez Balla i Van Slyke'a [Ball77]<sup>47</sup> i wykorzystywana później przez Carliera i Theologou [Carlier91] oraz Feo i Johnsona [Feo90]. Może ona znaleźć zastosowanie w trudnych obliczeniowo przypadkach analizy niezawodności złożonych sieci. Zamiast określenia dokładnej wartości niezawodności  $K$ -terminali, technika ta umożliwia szybsze określenie górnego i dolnego ograniczenia na wartość niezawodności  $K$ -terminali.

---

<sup>47</sup> Por. [Feo90] s.291.

Algorytm faktoryzacji uzupełniony o funkcje redukcji grafu można opisać w postaci następującej trójki  $(\mathbf{F}_0, \mathbf{R}, \mathbf{S})$ , gdzie:

$\mathbf{F}_0$  – szkielet algorytmu obliczania niezawodności wykorzystujący twierdzenie faktoryzacji,

$\mathbf{R}$  – zbiór zachowujących niezawodność redukcji i dekompozycji grafu,

$\mathbf{S}$  – strategia wybierania krawędzi grafu do faktoryzacji (dekompozycji problemu).

### 3.1. ZACHOWUJĄCE NIEZAWODNOŚĆ REDUKCJE I DEKOMPOZYCJE GRAFU

Aby zredukować rozmiar badanych grafów, a co za tym idzie także rozmiar przestrzeni stanów rozpatrywanego problemu niezawodności  $K$ -terminali, czyli złożoność problemu, można zastosować zachowujące niezawodność redukcje grafu (*reliability preserving reductions*) ze zbioru  $\mathbf{R}$ . Redukcje te wymagają tylko wielomianowego czasu wykonania<sup>48</sup> zmniejszając wykładniczą przestrzeń stanów problemu. Redukcje zmieniają graf  $G_K$  topologicznie i probabilistycznie przekształcając go do postaci  $G'_{K'}$  tak, że:

$$R(G_K) = \Omega_1 + \Omega_2 R(G'_{K'}). \quad (3.3)$$

$\Omega_1$  i  $\Omega_2$  są stałymi uzyskanymi wyłącznie z oryginalnego grafu  $G_K$ . Ta ogólna formuła jest często upraszczana do postaci, która zawiera jedynie stałą multiplikatywną  $\Omega = \Omega_2$ . Ponieważ większość redukcji wykorzystuje jedynie stałą multiplikatywną równanie (3.3) przyjmuje następującą postać:

$$R(G_K) = \Omega R(G'_{K'}) \quad (3.4)$$

Dekompozycja grafu, podobnie jak redukcja, zastępuje podgraf grafu  $G_K$  prostszym podgrafem zachowując niezawodność. Różnica w stosunku do redukcji polega na tym, że niezawodności krawędzi w podgrafie jak i stała multiplikatywna  $\Omega$  są obliczane przez wywołanie algorytmu obliczania niezawodności<sup>49</sup>.

---

<sup>48</sup> Por. [Wood85], s.175 i [Satyanarayana83], s.114.

<sup>49</sup> Por. [Wood89], s.207.

W dalszej części rozdziału zostaną przedstawione wykorzystywane w praktyce redukcje grafu<sup>50</sup> i najprostsza dwuspójna dekompozycja (*biconnected decomposition*) grafu. Redukcje R1, R2 i R3 są najczęściej stosowane. Zwykle określane są one mianem redukcji prostych lub standardowych. Najmniej znane<sup>51</sup> i najrzadziej stosowane są redukcje R7 i R8.

### R0 redukcja stopnia 1 (*degree-1 reduction*)

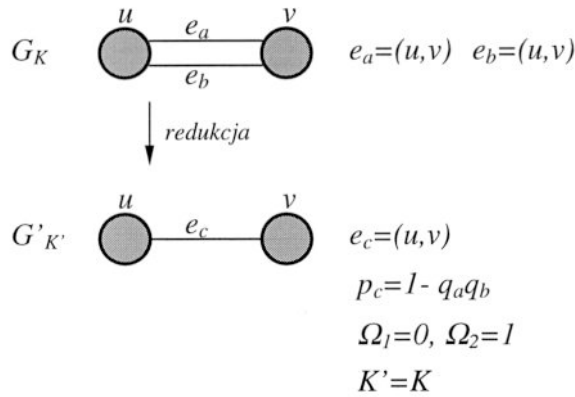
Jeśli  $e_i = (u, v)$  będzie krawędzią w grafie  $G_K$ , a wierzchołek  $v$  będzie stopnia jeden tzn.  $\deg(v) = 1$ , to wtedy:

$$G'_{K'} = G_K * e_i = (V - u - v + w, E - e_i), \quad w = u \cup v,$$

$$K' = \begin{cases} K & \text{jeżeli } u, v \notin K \\ K - u - v + w & \text{jeżeli } u \in K \text{ lub } v \in K. \end{cases}$$

$$\Omega_1 = 0, \Omega_2 = \begin{cases} 1 & \text{jeżeli } v \notin K \\ p_a & \text{jeżeli } v \in K \end{cases}$$

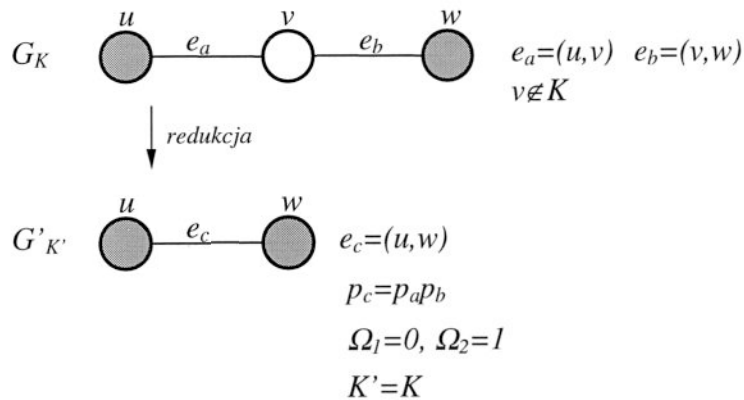
### R1 Redukcja równoległa (*parallel reduction*)



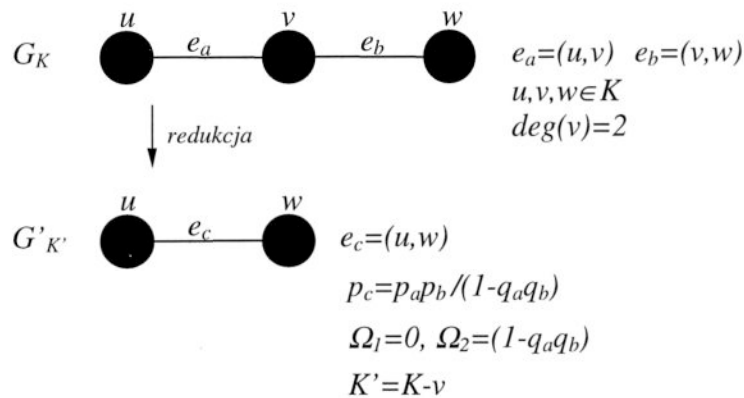
<sup>50</sup> Oznaczenia wierzchołków na rysunkach opisujących redukcje są zgodne z indeksem zamieszczonym na końcu pracy. Opis redukcji można znaleźć także w pracy [Wood86], s.272-273 oraz [Wood85], s.177-178.

<sup>51</sup> Por. [Wood86], s.272.

### R2 Redukcja szeregową (*series reduction*)

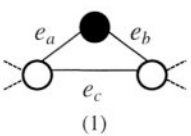
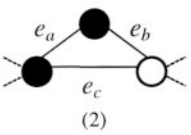
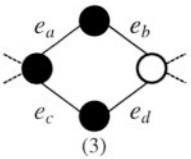
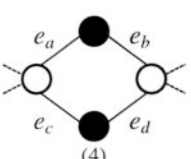
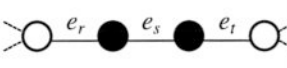
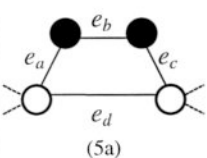
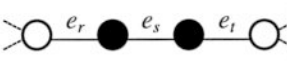
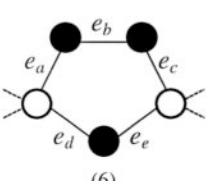
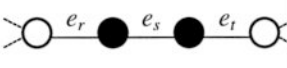
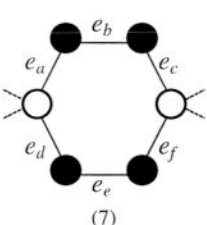
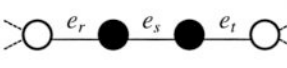


### R3 Redukcja stopnia 2 (*degree-2 reduction*)

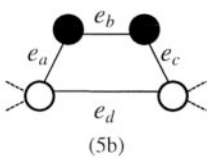
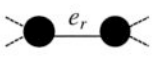


### R4 Redukcje wielokąta do łańcucha (*polygon-to-chain reduction*)

Satyanarayana i Wood [Satyanarayana82a, Wood82, Wood85] zaproponowali redukcje R4, wielokąta do łańcucha (*polygon-to-chain reductions*). Jeżeli po zastosowaniu redukcji R1, R2, R3 graf zawiera wielokąt, to przybiera on jedną z siedmiu postaci i można go zastąpić przez łańcuch składający się z 1, 2 lub 3 krawędzi. Redukcje te mogą zmniejszyć wysiłek obliczeniowy wymagany do obliczenia niezawodności za pomocą algorytmu faktoryzacji. Poniższa tabela zawiera wszystkie siedem redukcji wielokąta do łańcucha:

| Wielokąt                                                                                        | Łańcuch                                                                                          | Formuły redukcji                                                                                                                                                                                                                                                                                                                          | Niezawodności łączy łańcucha                                                                                  |
|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
|  <p>(1)</p>    |                 | $\alpha = q_a p_b q_c$<br>$\beta = p_a q_b q_c$                                                                                                                                                                                                                                                                                           | $p_r = \delta / (\alpha + \delta)$<br>$p_s = \delta / (\beta + \delta)$                                       |
|  <p>(2)</p>    |                 | $\delta = p_a p_b p_c [1 + (q_a/p_a) + (q_b/p_b) + (q_c/p_c)]$                                                                                                                                                                                                                                                                            | $\Omega_2 = (\alpha + \delta)(\beta + \delta) / \delta$<br>$\Omega_1 = 0$                                     |
|  <p>(3)</p>    |                 | $\alpha = p_a q_b q_c p_d + q_a p_b p_c q_d + q_a p_b q_c p_d$<br>$\beta = p_a q_b p_c q_d$<br>$\delta = p_a p_b p_c p_d [1 + (q_a/p_a) + (q_b/p_b) + (q_c/p_c) + (q_d/p_d)]$                                                                                                                                                             |                                                                                                               |
|  <p>(4)</p>    |                 | $\alpha = q_a p_b q_c p_d$<br>$\beta = p_a q_b q_c p_d + q_a p_b p_c q_d$<br>$\delta = p_a q_b p_c q_d$<br>$\gamma = p_a p_b p_c p_d [1 + (q_a/p_a) + (q_b/p_b) + (q_c/p_c) + (q_d/p_d)]$                                                                                                                                                 | $p_r = \gamma / (\alpha + \gamma)$<br>$p_s = \gamma / (\beta + \gamma)$<br>$p_t = \gamma / (\delta + \gamma)$ |
|  <p>(5a)</p> | $ K  > 2$<br> | $\alpha = q_a p_b p_c q_d$<br>$\beta = p_a q_b p_c q_d$<br>$\delta = p_a p_b q_c q_d$<br>$\gamma = p_a p_b p_c p_d [1 + (q_a/p_a) + (q_b/p_b) + (q_c/p_c) + (q_d/p_d)]$                                                                                                                                                                   | $\Omega_2 = (\alpha + \gamma)(\beta + \gamma)(\delta + \gamma) / \gamma^2$<br>$\Omega_1 = 0$                  |
|  <p>(6)</p>  |               | $\alpha = q_a p_b p_c q_d p_e$<br>$\beta = p_a q_b p_c (p_d q_e + q_d p_e) + p_b (q_a p_c p_d q_e + p_a q_c q_d p_e)$<br>$\delta = p_a p_b q_c p_d q_e$<br>$\gamma = p_a p_b p_c p_d p_e [1 + (q_a/p_a) + (q_b/p_b) + (q_c/p_c) + (q_d/p_d) + (q_e/p_e)]$                                                                                 |                                                                                                               |
|  <p>(7)</p>  |               | $\alpha = q_a p_b p_c q_d p_e p_f$<br>$\beta = p_a q_b p_c (q_d p_e p_f + p_d q_e p_f + p_d p_e q_f) + p_a p_b q_c p_f (p_d q_e + q_d p_e) + q_a p_b p_c p_d (q_e p_f + p_e q_f)$<br>$\delta = p_a p_b q_c p_d p_e p_f$<br>$\gamma = p_a p_b p_c p_d p_e p_f [1 + (q_a/p_a) + (q_b/p_b) + (q_c/p_c) + (q_d/p_d) + (q_e/p_e) + (q_f/p_f)]$ |                                                                                                               |

c.d. tabeli na następnej stronie

| Wielokąt                                                                                      | Łańcuch                                                                                                       | Formuły redukcji                                                                                                                                                        | Niezawodności łączy łańcucha                                                                   |
|-----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
|  <p>(5b)</p> |  <p><math> K  = 2</math></p> | $\alpha = q_a p_b p_c q_d$<br>$\beta = p_a q_b p_c q_d$<br>$\delta = p_a p_b q_c q_d$<br>$\gamma = p_a p_b p_c p_d [1 + (q_a/p_a) + (q_b/p_b) + (q_c/p_c) + (q_d/p_d)]$ | $p_r = (p_b + p_a q_b p_c p_d) / \Omega_2$<br>$\Omega_2 = p_b + p_a q_b p_c$<br>$\Omega_1 = 0$ |

**Tabela 1. Redukcje wielokąta do łańcucha (*polygon-to-chain reductions*) zaproponowane przez Satyanarayane i Wooda**

### R5 Ściąganie mostu (*bridge contraction reduction*)

Jeśli krawędź  $e_i = (u, v)$  będzie istotnym mostem (*relevant bridge*)<sup>52</sup>, to wtedy:

$$G'_{K'} = G_K * e_i = (V - u - v + w, E - e_i), \quad w = u \cup v,$$

$$K' = \begin{cases} K & \text{jeżeli } u, v \notin K \\ K - u - v + w & \text{jeżeli } u \in K \text{ lub } v \in K. \end{cases}$$

$$\Omega_1 = 0, \Omega_2 = p_a$$

Redukcja ta jest w istocie uogólnieniem redukcji R0. Jeżeli krawędź  $e_i$  jest zakończona wierzchołkiem  $v \in K$  o  $\deg(v) = 1$ , to ma miejsce redukcja R0.

### R6. Usunięcie nieistotnego składnika (*irrelevant component deletion*)

Jeśli  $G^0 = (V^0, E^0)$  będzie spójnym składnikiem grafu  $G$  połączonym z resztą grafu poprzez węzeł rozdzielający  $v^0$  oraz jeżeli  $K \cap (V^0 - v^0) = \emptyset$ , to składnik  $G^0$  jest nieistotny i redukcja polega na usunięciu tego składnika:

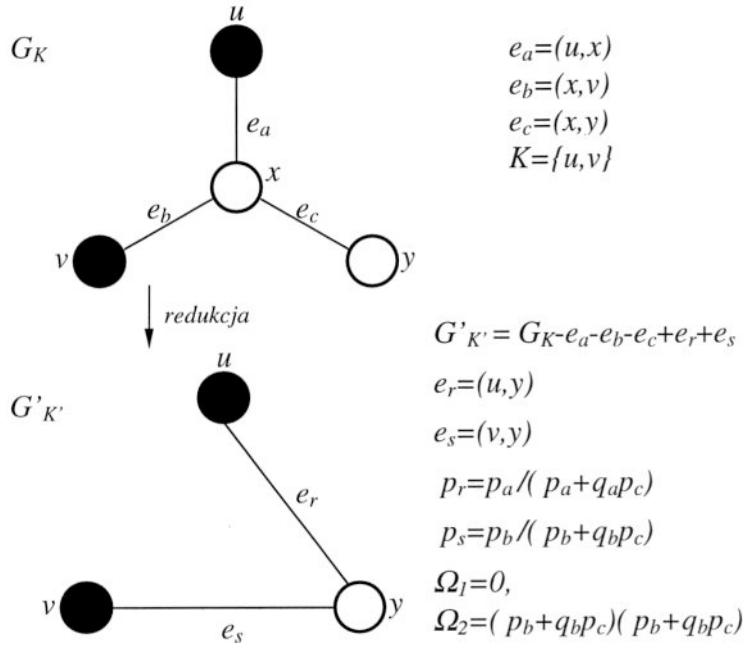
$$G'_{K'} = G_K - (V^0 - v^0),$$

$$K' = K,$$

$$\Omega_1 = 0, \Omega_2 = 1.$$

<sup>52</sup> Definicje terminów w rozdziale 1.1.

### R7. Redukcja stopnia 3 (degree-3 reduction)



### R8. Redukcja trywialna (trivial reduction)

Jeśli  $e_a = (u, v)$  będzie jedną z krawędzi grafu  $G_K$ , gdzie  $K = \{u, v\}$ , to wtedy:

$$G'_{K'} = G - e_a, \quad K' = K, \quad \Omega_1 = p_a, \quad \Omega_2 = q_a.$$

Redukcja ta nosi nazwę redukcji trywialnej, gdyż jest wynikiem faktoryzacji na krawędzi  $e_a$ .

Jest to przykład redukcji z niezerowym wyrażeniem addytywnym  $\Omega_1$ .

### D1. Dwuspójna dekompozycja (biconnected decomposition)

Dekompozycja grafu na dwuspójne części nosi nazwę dwuspójnej dekompozycji. Niezawodność grafu rozdzielnego (czyli grafu spójnego, którego spójność wierzchołkowa wynosi jeden) jest iloczynem niezawodności dwuspójnych części.

Przykładowo niech  $G_K$  składa się z dwóch dwuspójnych części tak, że  $G_K = G'_{K'} \cup G''_{K''}$ ,  $V' \cap V'' = \{v\}$ ,  $E' \cap E'' = \emptyset$ ,  $K' - v \neq \emptyset$  i  $K'' - v \neq \emptyset$ . Wtedy  $R(G_K) = R(G'_{K'}) R(G''_{K''})$  co można zapisać podobnie jak w przypadku redukcji  $R(G_K) = \Omega R(G''_{K''})$ , ale  $\Omega$  będzie obliczana poprzez wywołanie algorytmu obliczania niezawodności.



Techniki redukcji jak i dekompozycji grafu redukują złożoność obliczeniową algorytmu faktoryzacji. Redukcje wymagają tylko wielomianowego czasu wykonania zmniejszając wykładniczą przestrzeń stanów problemu, podczas gdy dekompozycje mogą wymagać wykładniczego czasu wykonania. Z tego względu użycie dekompozycji nie zawsze jest użyteczne (użycie dwuspójnej dekompozycji jest zawsze użyteczne)<sup>53</sup>.

### 3.2. STRATEGIA WYBIERANIA KRAWĘDZI GRAFU DO FAKTORYZACJI

Strategia **S** wybierania krawędzi grafu do faktoryzacji, czyli strategia dekompozycji problemu, jest zbiorem reguł wskazujących, które krawędzie mogą być wybrane do faktoryzacji. Strategia taka jest dobrze zdefiniowana tylko wtedy, gdy w każdym stanie, w którym może się znaleźć algorytm faktoryzacji, istnieje co najmniej jedna krawędź, która spełnia reguły strategii.

Przykładowo strategia **S** może być określona następująco:

$S = \{ \text{Dowolna krawędź } e=(u, v) \in E \text{ może być wybrana, jeżeli } u \in K \text{ lub } v \in K \}.$

Jeżeli realizacja strategii dekompozycji problemu jak i zachowujących niezawodność redukcji dla dowolnego grafu  $G_K$  jest ograniczona wielomianowo w funkcji rozmiaru grafu, to obliczenia reprezentowane za pomocą binarnego drzewa obliczeń są proporcjonalne do liczby liści tego drzewa. W takiej sytuacji optymalne binarne drzewo obliczeń otrzymane na skutek rekurencyjnego stosowania równania (2) to takie drzewo, które ma minimalną liczbę liści.

Optymalna strategia selekcji krawędzi do faktoryzacji, czyli dekompozycji problemu zależy od zastosowanych zachowujących niezawodność redukcji grafu. Satyanarayana i Chang [Satyanarayana83] badali algorytm faktoryzacji wykorzystujący redukcje szeregowo-równoległe do obliczania niezawodności  $K$ -terminali. Pokazali, że dla  $K$ -grafu  $G_K$ , wybór dowolnej krawędzi  $e_i$  takiej, że zarówno  $G_K * e_i$ , jak i  $G_K - e_i$  są  $K$ -grafami, jest warunkiem koniecznym i wystarczającym do uzyskania optymalnego drzewa obliczeń, czyli minimalizacji czasu obliczeń dla tak skonstruowanego algorytmu. Również Wood w swojej

---

<sup>53</sup> Por. [Wood89], s.207.

pracy [Wood86] rozważał optymalne strategie dekompozycji problemu dla algorytmów faktoryzacji wykorzystujących różne zbiory zachowujących niezawodność redukcji grafu.

### 3.3. SZKIELET ALGORYTMU FAKTORYZACJI

Parametry wejściowe: Graf nieskierowany  $G_K$  z odpowiednimi niezawodnościami krawędzi i wyróżnionym podzbiorem wierzchołków  $K$ . Graf nie zawiera krawędzi równoległych i pętli własnych.

Rezultat: Niezawodność  $K$ -terminali grafu  $G_K$ .

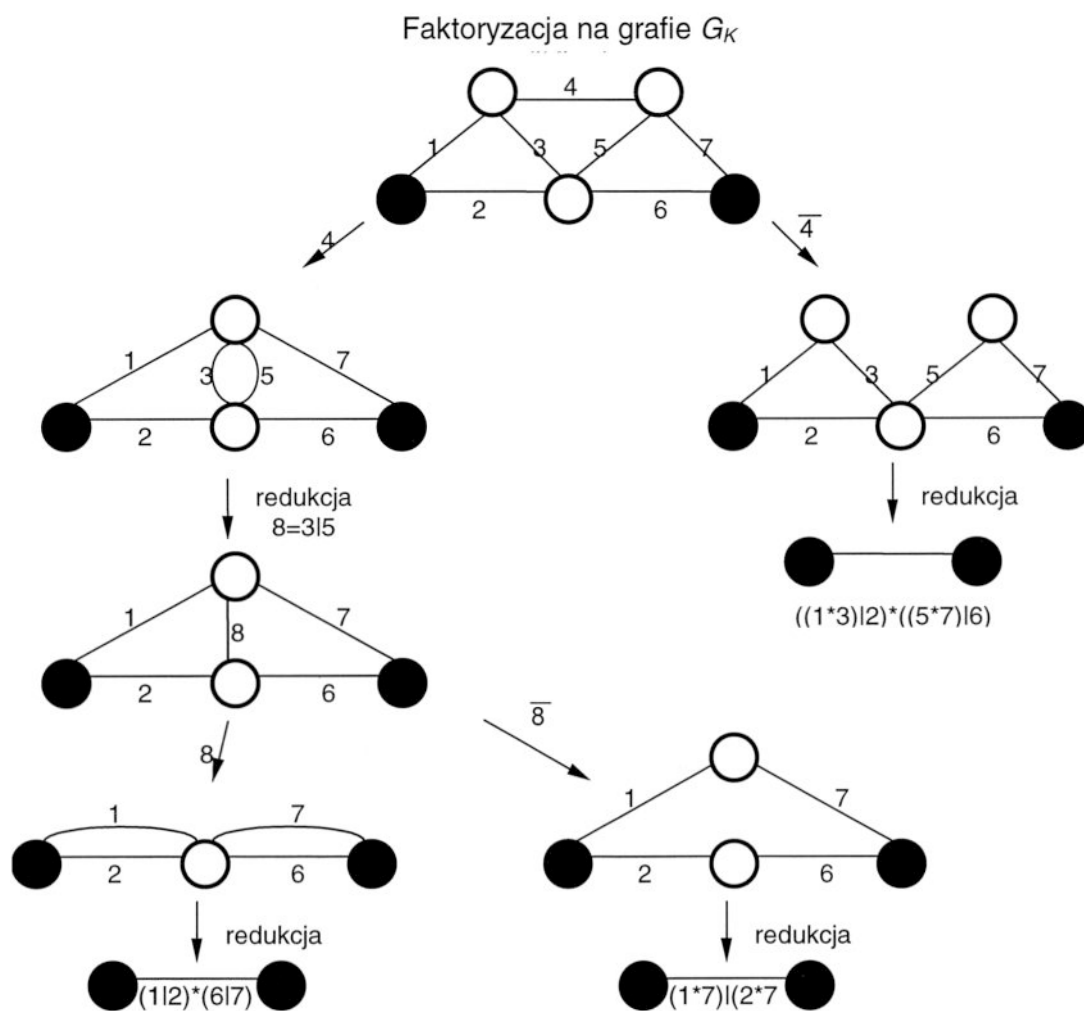
```

Prob( $G_K$ )
{
    if ( $|K| == 1$ )
        return(1);
    else
    {
        if(na grafie  $G_K$  można przeprowadzić redukcje ze zbioru  $R$ )
        {
            Zredukuj  $G_K$  do postaci  $G'_{K'}$ ;
            return( $\Omega_2 * \text{Prob}(G'_{K'})$ ); //gdzie  $\Omega_2$  jest odpowiednim wsp.
        }
        else
        {
            Wybierz krawędź  $e$  używając strategii  $S$ ;
            Usuń krawędź  $e$  tworząc  $G_K - e$ ;
            if(węzły w  $K$  są rozłączone)
                 $P1 = 0$ 
            else
                 $P1 = \text{Prob}(G_K - e)$ ;
            Połącz wierzchołki na końcach krawędzi  $e$  tworząc  $G_K * e$ ;
             $P2 == \text{Prob}(G_K * e)$ ;
            return(( $1 - p$ ) *  $P1$  +  $p * P2$ ); //gdzie  $p$  oznacza niezawodność
                                     //krawędzi  $e$ 
        }
    }
}

```

### 3.4. PRZYKŁAD ZASTOSOWANIA ALGORYTMU

Poniżej zaprezentowano przykład zastosowania algorytmu faktoryzacji do obliczenia niezawodności dwóch terminali ( $|K| = 2$ ) prostej sieci o pięciu wierzchołkach i siedmiu krawędziach reprezentowanej poprzez graf  $G_K$ . Binarne drzewo obliczeń (BDO) algorytmu faktoryzacji wygląda następująco:



gdzie:

\* redukcja szeregową;

| redukcja równoległą;

● węzły należące do zbioru  $K$ ;

○ węzły nie należące do zbioru  $K$ ;

Niezawodność  $K$ -terminali sieci reprezentowanej poprzez graf  $G_K$ :

$$R(G_K) = p_4 [(1 - q_3 q_5) (1 - q_1 q_2) (1 - q_6 q_7) + q_3 q_5 (p_1 p_7 + p_2 p_6 - p_1 p_2 p_6 p_7)] + \\ q_4 (p_1 p_3 + p_2 - p_1 p_2 p_3) (p_5 p_7 + p_6 - p_5 p_6 p_7).$$

### 3.5. PODSUMOWANIE ROZDZIAŁU

Wniesiono wielki wkład w udoskonalenie pierwotnej wersji algorytmu faktoryzacji. Zastosowano, wymagające jedynie wielomianowego czasu wykonania, zachowujące niezawodność redukcje grafu, uzyskując znaczne zmniejszenie binarnego drzewa obliczeń. Opracowano również efektywne strategie selekcji krawędzi do faktoryzacji. Jednakże ze względu na konieczność analizy coraz bardziej skomplikowanych sieci, efektywność zaproponowanych algorytmów faktoryzacji jest wciąż niewystarczająca. Istnieje więc nagleca potrzeba dalszych modyfikacji w celu uzyskania jeszcze efektywniejszych algorytmów.

## 4. ANALIZA PESYMISTYCZNEJ ZŁOŻONOŚCI OBLICZENIOWEJ ALGORYTMÓW FAKTORYZACJI

Na złożoność obliczeniową algorytmu składa się złożoność czasowa i pamięciowa. Złożoność pamięciowa jest zwykle wyrażana w słowach pamięci maszyny. Złożoność czasową wyraża się poprzez, charakterystyczne dla rozpatrywanego algorytmu, operacje dominujące – takie, że łączna ich liczba jest proporcjonalna do liczby wykonań wszystkich operacji jednostkowych.

Złożoność obliczeniową traktuje się zwykle jako funkcję rozmiaru danych wejściowych  $n$ . W zależności od charakteru problemu może to być liczba elementów w ciągu wejściowym, całkowita liczba bitów czy, jak to ma miejsce w przypadku grafów, liczba krawędzi i/lub wierzchołków. Rozpatruje się:

- złożoność pesymistyczną, która jest definiowana jako ilość zasobów komputerowych, potrzebnych przy „najgorszych” danych wejściowych rozmiaru  $n$ ;
- złożoność średnią, która jest definiowana jako ilość zasobów komputerowych, potrzebnych przy „typowych” danych wejściowych rozmiaru  $n$ ;
- złożoność optymistyczną, która jest definiowana jako ilość zasobów komputerowych, potrzebnych przy „najlepszych” danych wejściowych rozmiaru  $n$ .

W istocie nie wszystkie problemy o tym samym rozmiarze wymagają tego samego wysiłku obliczeniowego. Istotny wpływ na złożoność obliczeniową ma struktura problemu. Zwykle w takim przypadku jako miarę złożoności obliczeniowej algorytmu wykorzystuje się funkcję złożoności pesymistycznej  $f(n)^{54}$ , która określa maksymalną ilość zasobów komputerowych (czy to czasu procesora, czy rozmiaru pamięci) dla instancji problemu o rozmiarze  $n$ . Również w niniejszej pracy rozważana będzie pesymistyczna złożoność obliczeniowa algorytmów faktoryzacji.

W rozdziale tym zaprezentowano znane z literatury metody analizy pesymistycznej złożoności niektórych algorytmów faktoryzacji.

---

<sup>54</sup> Por. [Shier91], s.18.

#### 4.1. ZŁOŻONOŚĆ CZASOWA ALGORYTMÓW FAKTORYZACJI

Chcąc oszacować złożoność czasową algorytmów faktoryzacji, należy uwzględnić złożoność operacji dokonywanych w poszczególnych węzłach BDO, jak również liczbę węzłów (liści) BDO. W poszczególnych węzłach BDO realizowane są typowe dla większości algorytmów faktoryzacji procedury redukcji grafu i selekcji krawędzi do faktoryzacji. Ponieważ operacje dokonywane w poszczególnych węzłach BDO wymagają jedynie wielomianowej (w funkcji rozmiaru problemu) liczby operacji [Satyanarayana83], znacznie istotniejsza z punktu widzenia złożoności czasowej algorytmów faktoryzacji jest liczba węzłów BDO.

Przez  $C(n, b)$  oznaczmy złożoność czasową operacji wykonywanych w dowolnym węźle binarnego drzewa obliczeń. Niech  $N(G_K)$ ,  $I(G_K)$  i  $L(G_K)$  oznaczają odpowiednio liczbę węzłów, liczbę węzłów wewnętrznych (czyli nie będących liśćmi) i liczbę liści BDO, które łączy zależność  $N(G_K) = I(G_K) + L(G_K)$ . Całkowita liczba wywołań funkcji rekurencyjnej *Prob* algorytmu faktoryzacji równa jest liczbie węzłów  $N(G_K)$  binarnego drzewa obliczeń. Ponieważ jednak drzewo obliczeń jest drzewem binarnym, liczbę liści  $L(G_K) = \frac{N(G_K) + 1}{2}$  zwykle używa się jako miary złożoności czasowej algorytmów faktoryzacji [Satyanarayana83, Wood86, Carlier91, Theologou91]. Oczywiście zamiennie mogą być również używane takie miary, jak liczba węzłów wewnętrznych  $I(G_K) = \frac{N(G_K) - 1}{2}$  i całkowita liczba węzłów  $N(G_K)$  binarnego drzewa obliczeń<sup>55</sup>.

Graf sieci, dla której obliczamy niezawodność  $K$ -terminali, jest korzeniem binarnego drzewa obliczeń, a grafy indukowane powstające w wyniku faktoryzacji i redukcji są węzłami tego drzewa.

---

<sup>55</sup> Miary te mogą być używane zamiennie, gdyż liczba liści BDO jest dokładnie o jeden większa niż liczba węzłów wewnętrznych, zaś liczba węzłów BDO jest równa sumie liczby liści i liczby węzłów wewnętrznych BDO.

#### 4.1.1. Niezmienniki grafu a twierdzenie faktoryzacji

Niezmiennikiem (*invariant*) grafu  $G$  jest liczba, która jest identyczna dla wszystkich grafów izomorficznych do grafu  $G$ . Właściwości pewnych niezmienników grafów przyjmują formę twierdzenia faktoryzacji. Twierdzenie faktoryzacji zdefiniowane na grafie  $G_K$  z wykorzystaniem dowolnej krawędzi  $e_i = (u, v) \in E$  i funkcji  $f(G_K)$  przyjmującej wartości rzeczywiste definiuje następującą relację:

$$f(G_K) = g_1(e_i)f(G_{K'} * e_i) + g_2(e_i)f(G_{K''} - e_i),$$

gdzie:

$g_1(e_i)$  i  $g_2(e_i)$  są funkcjami rzeczywistymi,

$G * e_i = (V - u - v + u \cup v, E - e_i)$  i  $G - e_i = (V, E - e_i)$  są grafami indukowanymi (*induced graphs*) przez faktoryzację,

$K'$  i  $K''$  są podzbiorami węzłów.

Funkcja  $f(G)$  odpowiada zwykle pewnym niezmiennikom grafu<sup>56</sup>.

#### 4.1.2. Wykorzystanie niezmienników grafów do analizy złożoności niektórych algorytmów faktoryzacji

Bodaj najbardziej znanym niezmiennikiem grafu, którego jedna z własności przyjmuje formę twierdzenia faktoryzacji, jest liczba drzew rozpinających grafu  $r(G)$ .

*Drzewa rozpinające*

Niezmiennik:  $r(G)$  jest liczbą drzew rozpinających grafu  $G$

**Własność 1 ([Wood86])**

- (a)  $r(G) = r(G * e_i) + r(G - e_i)$  – twierdzenie faktoryzacji;
- (b)  $r(G) = r(G * e_i)$ , jeżeli  $e_i$  jest mostem;
- (c)  $r(G) = 0$ , jeżeli  $G$  jest niespójny.

---

<sup>56</sup> Por. [Wood86], s.274.



Liczba drzew rozpinających jest wykorzystywana do analizy złożoności czasowej obliczania problemu niezawodności wszystkich terminali przez prosty algorytm faktoryzacji w postaci  $(F_0, R_1=\{R5, R6\}, S=S_1)$ ,

gdzie:  $S_1$  jest dowolną strategią selekcji krawędzi grafu.

Wood pokazał, że algorytm ten produkuje liczbę liści binarnego drzewa obliczeń  $L(G_v)$  równą liczbie drzew rozpinających  $r(G)$ .

**Twierdzenie 2 ([Wood86])**

Dla algorytmu w postaci  $(F_0, \{R5, R6\}, S)$  stosowanego do rozwiązania problemu niezawodności wszystkich terminali, dowolna strategia selekcji krawędzi  $S=S_1$  jest optymalna i produkuje  $L(G_v)=r(G)$ .

Wykorzystując to twierdzenie można oszacować złożoność czasową algorytmu  $(F_0, \{R5, R6\}, S)$ .

**Twierdzenie 3**

Pesymistyczna złożoność czasowa algorytmu  $(F_0, \{R5, R6\}, S)$  stosowanego do rozwiązania problemu niezawodności wszystkich terminali wynosi  $O(C(n,b)r(G))$ , czyli  $O(C(n,b)n^{n-2})$ , gdzie  $n$  jest liczbą wierzchołków,  $b$  jest liczbą krawędzi grafu, a  $C(n,b)$  oznacza złożoność czasową operacji wykonywanych w dowolnym węźle binarnego drzewa obliczeń, natomiast  $r(G)$  jest liczbą drzew rozpinających.

Dowód:

Złożoność czasową algorytmu można oszacować przez  $O(C(n,b)L(G_v))$ . Z twierdzenia 2 wynika, że  $L(G_v)=r(G)$ . Oczywiście jest, że spośród wszystkich grafów o danej liczbie wierzchołków  $n$ , graf pełny charakteryzuje się największą liczbą drzew rozpinających. Dla zaetykietowanego grafu pełnego liczba drzew rozpinających jest liczbą różnych drzew zaetykietowanych o  $n$  wierzchołkach. Twierdzenie o liczbie drzew sformułował po raz pierwszy Cayley, stąd nazywa się je twierdzeniem Cayleya.

**Twierdzenie 4 (Cayleya)<sup>57</sup>**

Liczba drzew zaetykietowanych o  $n$  wierzchołkach ( $n \geq 2$ ) wynosi  $n^{n-2}$ .

---

<sup>57</sup> Deo [Deo80] zamieścił na s.313 dowód tego twierdzenia, a na s.81 można znaleźć odsyłacz do literatury zawierającej podsumowanie dziesięciu dowodów wykorzystujących różne podejścia.

Stąd  $r(G) \leq n^{n-2}$ . Chcąc oszacować złożoność czasową należy uwzględnić, że w każdym węźle binarnego drzewa obliczeń algorytmu faktoryzacji wykonywane są pewne operacje (np. redukcje grafu czy selekcja krawędzi do faktoryzacji), których złożoność czasową można oszacować przez  $C(n, b)$ . Stąd pesymistyczną złożoność czasową algorytmu  $(\mathbf{F}_0, \{\mathbf{R}_5, \mathbf{R}_6\}, \mathbf{S}_1)$  można oszacować przez  $O(C(n, b)n^{n-2})$ .

c. n. d.

Innym przykładem niezmiennika grafu, którego jedna z własności przyjmuje formę twierdzenia faktoryzacji, jest dominacja.

*Dominacja*<sup>58</sup>

Niezmiennik:  $D(G_K) = |N_o - N_e|$

#### **Własność 2 ([Wood86])**

- (a) Jeżeli  $D(G_K) > 0$ , to  $D(G_K) = D(G_K * e) + D(G_K - e)$  – twierdzenie faktoryzacji;
- (b)  $D(G_K)$  jest niezmiennikiem ze względu na redukcje R1, R2 i R5 (także R0);
- (c)  $D(G_K) = 0$ , jeżeli  $G_K$  jest niekoherentny;
- (d)  $D(G_K) = 1$ , jeżeli  $G_K$  jest  $K$ -drzewem lub jest redukowalny do  $K$ -drzewa poprzez redukcje szeregowo-równoległe.

Dominacja jest przydatna w analizie złożoności czasowej obliczania problemu niezawodności  $K$ -terminali przez algorytm faktoryzacji w postaci  $(\mathbf{F}_0, \mathbf{R}=\mathbf{R}_2, \mathbf{S}=\mathbf{S}_2)$ , gdzie

$\mathbf{R}_2 = \{\text{Zbiór redukcji, które nie zmieniają } D(G_K) \text{ np. R1, R2, R5}\};$

$\mathbf{S}_2 = \{\text{Wybierz dowolną krawędź } e_i \in E \text{ taką, że zarówno } G_K * e_i, \text{ jak i } G_K - e_i \text{ są koherentne}\}.$

Dla tego algorytmu Wood udowodnił następujące twierdzenie:

#### **Twierdzenie 5 ([Wood86])**

Dla algorytmu w postaci  $(\mathbf{F}_0, \{\mathbf{R}_1, \mathbf{R}_2, \mathbf{R}_5\}, \mathbf{S})$  strategia selekcji krawędzi  $\mathbf{S}=\mathbf{S}_2$  jest optymalna do rozwiązania dowolnego problemu niezawodności  $K$ -terminali na grafie  $G_K$ , i  $L(G_K) = D(G_K)$ .

---

<sup>58</sup> Definicje podstawowych pojęć z teorii grafów znajdują się w rozdziale 1.1.

Wykorzystując to twierdzenie można oszacować pesymistyczną złożoność czasową algorytmu  $(\mathbf{F}_0, \{R1, R2, R5\}, \mathbf{S})$ .

#### Twierdzenie 6

Pesymistyczna złożoność czasowa algorytmu obliczania niezawodności  $K$ -terminali w postaci  $(\mathbf{F}_0, \{R1, R2, R5\}, \mathbf{S}_2)$  wynosi  $O(C(n,b)D(G_K))$ , czyli  $O(C(n,b)(n-1)!)$ , gdzie  $n$  jest liczbą wierzchołków,  $b$  liczbą krawędzi grafu,  $C(n,b)$  oznacza złożoność czasową operacji wykonywanych w dowolnym węźle binarnego drzewa obliczeń, a  $D(G_K)$  oznacza dominację.

Dowód:

Satyanarayana i Chang udowodnili następujące twierdzenie:

#### Twierdzenie 7 ([Satyanarayana83])

Jeżeli graf  $G_K$  jest grafem pełnym o  $n$  wierzchołkach, to  $D(G_K) = (|K| - 1)(n - 2)!$  dla  $|K| \geq 2$ .

Satyanarayana i Chang<sup>59</sup> pokazali również, że prawdziwa jest następująca własność:

#### Własność 3 ([Satyanarayana83])

Maksymalna możliwa wartość  $D(G_K)$  dla dowolnego grafu o  $n$  wierzchołkach wynosi  $(|K| - 1)(n - 2)!$ .

Z twierdzenia 7. i własności 3. wynika, że spośród grafów o określonej liczbie wierzchołków największą wartość dominacji ma graf pełny, którego wszystkie węzły należą do zbioru  $K$ . W tym przypadku  $D(G_K) = (n - 1)!$ . Pesymistyczna złożoność czasowa algorytmu jest liczona dla „najgorszych” danych wejściowych o rozmiarze  $n$ , czyli w tym przypadku dla problemu obliczania niezawodności wszystkich terminali sieci reprezentowanej poprzez graf pełny. Chcąc oszacować złożoność czasową należy uwzględnić złożoność czasową  $C(n,b)$  operacji dokonywanych w poszczególnych węzłach BDO. Z tego wynika, że pesymistyczna złożoność czasowa algorytmu w postaci  $(\mathbf{F}_0, \{R1, R2, R5\}, \mathbf{S}_2)$  wynosi  $O(C(n,b)(n-1)!)$ .

c. n. d.

---

<sup>59</sup> Por. [Satyanarayana83], s.115.

W wyniku zastosowania odpowiedniego zbioru redukcji uzyskano więc znaczne zwiększenie efektywności algorytmu w porównaniu do algorytmu w postaci  $(\mathbf{F}_0, \{R5, R6\}, \mathbf{S}_1)$ . Generalnie zachodzi następująca nierówność  $D(G_K) \leq r(G)$ .

Satayanarayana i Chang pierwsi użyli niezmiennika grafu określanego jako minimalna dominacja do analizy problemu niezawodności wszystkich terminali<sup>60</sup>.

*Minimalna dominacja*<sup>61</sup>

Niezmiennik:  $\mu(G) = \min_{K: |K|=2} D(G_K)$ .

**Własność 4 ([Wood86])**

- (a) Jeżeli  $\mu(G) > 0$ , to  $\mu(G) = \mu(G - e_i) + \mu(G * e_i)$  – twierdzenie faktoryzacji;
- (b) Jeżeli  $\mu(G) > 0$ , to  $\mu(G)$  jest niezmiennikiem ze względu na redukcje szeregowo i równoległe;
- (c)  $\mu(G) > 0$ , jeżeli  $G$  nie ma pętli własnych i jest dwuspójny;
- (d)  $\mu(G) = 1$ , jeżeli  $G$  jest pojedynczą krawędzią lub dwuspójnym szeregowo-równoległym grafem.

Satayanarayana i Chang udowodnili, że w przypadku analizy niezawodności wszystkich terminali, algorytm w postaci  $(\mathbf{F}_0, \{R1, R3\}, \mathbf{S}_3)$  wykorzystujący optymalną strategię selekcji krawędzi  $\mathbf{S}_3$ , produkuje liczbę liści równą minimalnej dominacji, czyli  $L(G_V) = \mu(G)$ . Ponieważ  $\mu(G) = \min_{K: |K|=2} D(G_K)$ , to problem obliczenia niezawodności wszystkich terminali za pomocą algorytmu  $(\mathbf{F}_0, \{R1, R3\}, \mathbf{S}_3)$  jest tak łatwy, jak najłatwiejszy problem obliczenia niezawodności dwóch terminali za pomocą algorytmu  $(\mathbf{F}_0, \{R1, R2, R5\}, \mathbf{S}_2)$ .

Wood [Wood85, Wood86] wykazał, że  $L(G_K) = \mu(G) \leq (n-2)!$  jest osiągalne dla algorytmu  $(\mathbf{F}_0, \{R1, R2, R3, R4\}, \mathbf{S}_4)$  w przypadkach granicznych, gdy  $2 \leq |K| \leq 5$  oraz  $n-2 \leq |K| \leq n$  przy restrykcjach odnośnie stosowanej strategii selekcji krawędzi  $\mathbf{S}_4$  i grafu  $G_K$ . Graf  $G_K$  musi być dwuspójny i bez pętli własnych, a strategia selekcji krawędzi musi być zdefiniowana następująco:

---

<sup>60</sup> Por. [Wood86], s.276.

$S_4 = \{\text{Wybierz dowolną krawędź } e_i \in E \text{ taką, że zarówno } G_K * e_i, \text{ jak i } G_K - e_i \text{ nie mają pętli własnych i są dwuspójne}\}.$

## 4.2. ZŁOŻONOŚĆ PAMIĘCIOWA ALGORYTMÓW FAKTORYZACJI

W odróżnieniu od złożoności czasowej, złożoność pamięciowa algorytmów faktoryzacji jest zdecydowanie rzadziej rozważana. Złożonością pamięciową algorytmów faktoryzacji zajmowali się jedynie Satyanarayana i Chang [Satyanarayana83], analizując złożoność zaproponowanego przez siebie algorytmu faktoryzacji. Algorytm ten wykorzystywał redukcje R1, R2 oraz wykrywał, czy graf jest drzewem. Zastosowana strategia selekcji krawędzi wymagała, by wybrana krawędź  $e_i$  gwarantowała, że  $D(G_K - e_i) \neq 0$  oraz  $D(G_K * e_i) \neq 0$ . Podana przez autorów pesymistyczna złożoność pamięciowa tego algorytmu wynosiła  $O((b - n + 1)|G_K|)$ .

## 4.3. PODSUMOWANIE ROZDZIAŁU

Jak widać problem analizy złożoności obliczeniowej algorytmów faktoryzacji nie został całościowo rozwiązany ze względu na ograniczony zakres zastosowań przedstawionych w tym rozdziale metod analizy pesymistycznej złożoności algorytmów faktoryzacji. W przypadku analizy złożoności czasowej konieczne jest bowiem przyjęcie pewnych specyficznych założeń odnośnie stosowanych zbiorów redukcji, strategii selekcji krawędzi czy grafu  $G_K$ . Muszą one zostać tak wybrane, aby spełniały pewne, ściśle określone własności. Page i Perry [Page88] twierdzą wręcz, że powyższe postępowanie jest co prawda użyteczne z punktu widzenia analizy złożoności obliczeniowej rozpatrywanych algorytmów, ale jest bezproduktywne, gdy chcemy zakończyć rekurencję (czyli otrzymać wynik) tak szybko, jak to tylko możliwe. Poza tym niektóre strategie selekcji krawędzi są skomplikowane (np. wymagają testowania czy rozpatrywane podproblemy są dwuspójne), a stosowane strategie selekcji krawędzi zależą od rodzaju zastosowanych redukcji grafu.

---

<sup>61</sup> Definicje podstawowych pojęć z teorii grafów znajdują się w rozdziale 1.1.

Najefektywniejsze algorytmy faktoryzacji charakteryzuje maksymalna liczba liści binarnego drzewa obliczeń  $(|V|-2)!$  osiągalna dla przypadków granicznych gdy  $2 \leq |K| \leq 5$  i  $|V|-2 \leq |K| \leq |V|$  [Wood85, Wood86] oraz  $(|V|-1)!$  dla dowolnego zbioru  $K$  ( $2 \leq |K| \leq |V|$ ) [Satyanarayana83, Wood86].

Złożoność pamięciowa algorytmów faktoryzacji jest rzadko rozważana. Częściowo wynika to z faktu, że rozmiar wymaganej pamięci można ogólnie oszacować poprzez iloczyn rozmiaru sieci i głębokości rekurencji. Z kolei głębokość rekurencji – ze względu na twierdzenie faktoryzacji – nie przekracza rozmiaru sieci. W rezultacie złożoność pamięciowa algorytmu faktoryzacji jest w najgorszym razie funkcją kwadratową rozmiaru sieci. Złożoność czasowa algorytmu faktoryzacji rośnie więc znacznie szybciej, a co za tym idzie, jest znacznie istotniejsza, niż złożoność pamięciowa.

## 5. ALGORYTM FAKTORYZACJI *FACT*

Zaproponowany w tym rozdziale nowy algorytm *Fact* ( $F_0, \{R_0, R_1, R_2, R_3\}, S''$ ) jest modyfikacją algorytmu faktoryzacji przedstawionego przez Page'a i Perry [Page88]. Wykorzystano bowiem ten sam zbiór zachowujących niezawodność redukcji modyfikując strategię wybierania krawędzi do faktoryzacji. Dzięki nowej strategii  $S''$  selekcji krawędzi do faktoryzacji, odpowiednio dobranemu zbiorowi redukcji  $R$ , jak również nowej metodzie analizy pesymistycznej złożoności obliczeniowej algorytmu, wykazemy, że maksymalna liczba  $(|V|-2)!$  liści binarnego drzewa obliczeń algorytmu faktoryzacji jest osiągalna dla dowolnego zbioru  $K$  ( $2 \leq |K| \leq |V|$ ), a nie tylko dla przypadków granicznych, gdy  $2 \leq |K| \leq 5$  oraz  $|V|-2 \leq |K| \leq |V|$  (jak to ma miejsce w przypadku algorytmu analizowanego przez Wooda [Wood85, Wood86]). Rezultat ten uzyskano przy prostszym niż zaproponowany przez Wooda zbiorze zachowujących niezawodność redukcji grafu i nieskomplikowanej strategii selekcji krawędzi do faktoryzacji. Dokładna analiza procesu faktoryzacji dla algorytmów faktoryzacji wykorzystujących strategię  $S''$ , jak również sama strategia  $S''$  selekcji krawędzi do faktoryzacji, przedstawiona jest na rysunku 5 w rozdziale 5.1.2.2.

### 5.1. ANALIZA PESYMISTYCZNEJ ZŁOŻONOŚCI OBLICZENIOWEJ ALGORYTMÓW FAKTORYZACJI Z WYKORZYSTANIEM GRAFÓW PEŁNYCH

W rozdziale 4. przedstawiono znane z literatury rezultaty analizy pesymistycznej złożoności algorytmów faktoryzacji. Zastosowane tam metody analizy wykorzystywały niezmienniki grafów. W niniejszym rozdziale zaproponowano odmienną metodę [Madeyski99b] analizy pesymistycznej złożoności algorytmu *Fact*, jak również odmian tego algorytmu, wykorzystujących różne zbiory zachowujących niezawodność redukcji grafów. Złożoność czasowa tych algorytmów nie była dotąd analizowana.

#### 5.1.1. Idea wykorzystania grafów pełnych

Mamy dowolną sieć reprezentowaną poprzez graf  $G_K$  o  $n$ -wierzchołkach. Graf  $G_K$  możemy uzupełnić do postaci grafu pełnego krawędziami reprezentującymi łącza o  $p_i = 0$ . W



wyniku tej operacji niezawodność  $K$ -terminali takiej sieci nie ulega zmianie. Dokonując faktoryzacji (wykorzystując określoną strategię selekcji  $S$ ) na krawędziach o  $p_i = 0$ , zamiast dwóch podproblemów (jak ma to miejsce przy faktoryzacji na krawędziach o  $p_i \neq 0$ ) otrzymujemy tylko jeden podproblem. W związku z tym liczba liści BDO algorytmu faktoryzacji wykorzystującego strategię  $S$  przy rozwiązywaniu dowolnych sieci reprezentowanych przez  $G_K$ , nie będzie większa niż  $L(K_n)$ , gdzie  $K_n$  jest grafem pełnym o  $n$  wierzchołkach. Podobnie głębokość rekurencji przy rozwiązywaniu sieci reprezentowanych przez  $G_K$  nie będzie większa niż w przypadku grafu pełnego.

Jeżeli ponadto strategia  $S$  selekcji krawędzi do faktoryzacji będzie tak dobrana by miała miejsce dekompozycja problemu obliczenia niezawodności  $K$ -terminali sieci reprezentowanej przez graf pełny  $K_n$  (o  $n$  wierzchołkach) na pewną liczbę podproblemów niższego rzędu reprezentowanych poprzez grafy pełne  $K_{n-1}$  to określenie pesymistycznej złożoności obliczeniowej rozważanych algorytmów faktoryzacji będzie stosunkowo proste.

### 5.1.2. Złożoność czasowa algorytmu *Fact* i jego modyfikacji

Chcąc oszacować złożoność czasową algorytmów faktoryzacji, należy uwzględnić złożoność operacji dokonywanych w poszczególnych węzłach BDO, jak również liczbę węzłów (liści) BDO. W poszczególnych węzłach BDO realizowane są typowe dla większości algorytmów faktoryzacji procedury redukcji grafu i selekcji krawędzi do faktoryzacji<sup>62</sup>. Ponieważ operacje dokonywane w poszczególnych węzłach BDO wymagają jedynie wielomianowej (w funkcji rozmiaru problemu) liczby operacji, znacznie istotniejsza z punktu widzenia złożoności czasowej algorytmów faktoryzacji jest liczba węzłów BDO. Z tego względu zaimplementowane przez autora procedury redukcji grafu, realizowane w poszczególnych węzłach BDO, nie były specjalnie optymalizowane pod kątem zmniejszenia ich złożoności czasowej. Głównym celem było ograniczenie liczby liści BDO.

---

<sup>62</sup> W przypadku algorytmu *Fact&Cache* (przedstawionego w rozdziale 6) należy również uwzględnić złożoność procedur związanych z wykrywaniem równoważnych podproblemów..

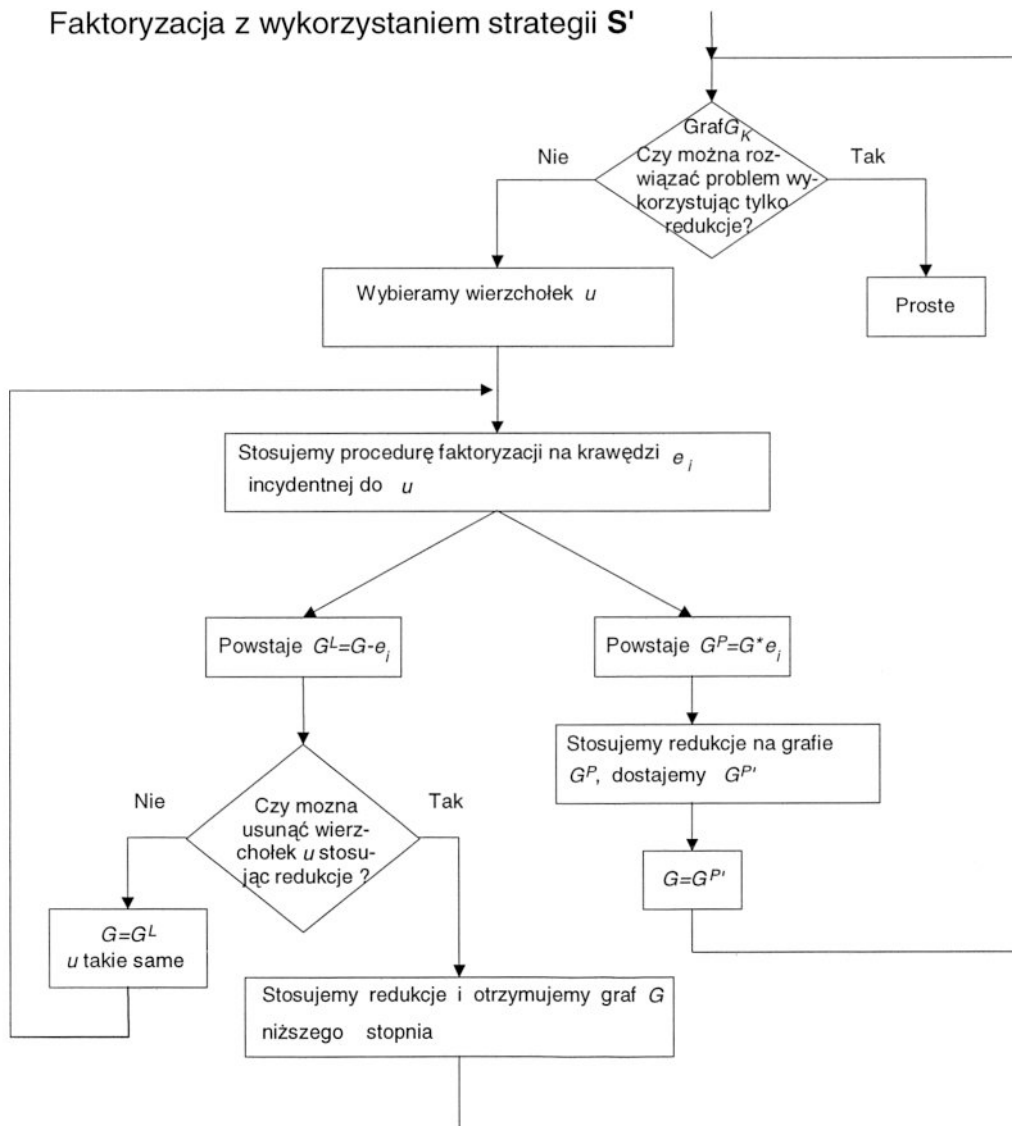
### 5.1.2.1. Niezawodność wszystkich terminali

Dla ułatwienia rozpatrzmy najpierw często rozważany przypadek obliczania niezawodności wszystkich terminali (tzn. gdy  $|K| = |V|$ ).

#### Twierdzenie 8

Liczba liści BDO algorytmu faktoryzacji, wykorzystującego zbiór redukcji  $\mathbf{R}=\{R_0, R_1, R_2, R_3\}$  oraz strategię  $\mathbf{S}=\mathbf{S}'$  (określoną na rysunku poniżej), przy rozwiązywaniu problemu niezawodności wszystkich terminali dowolnej sieci reprezentowanej poprzez graf  $G_v$ , spełnia zależność  $L(G_v) \leq (n-2)!$ , gdzie  $n$  jest liczbą wierzchołków (przy rozwiązywaniu problemu niezawodności wszystkich terminali redukcje  $R_2$  nie są używane).

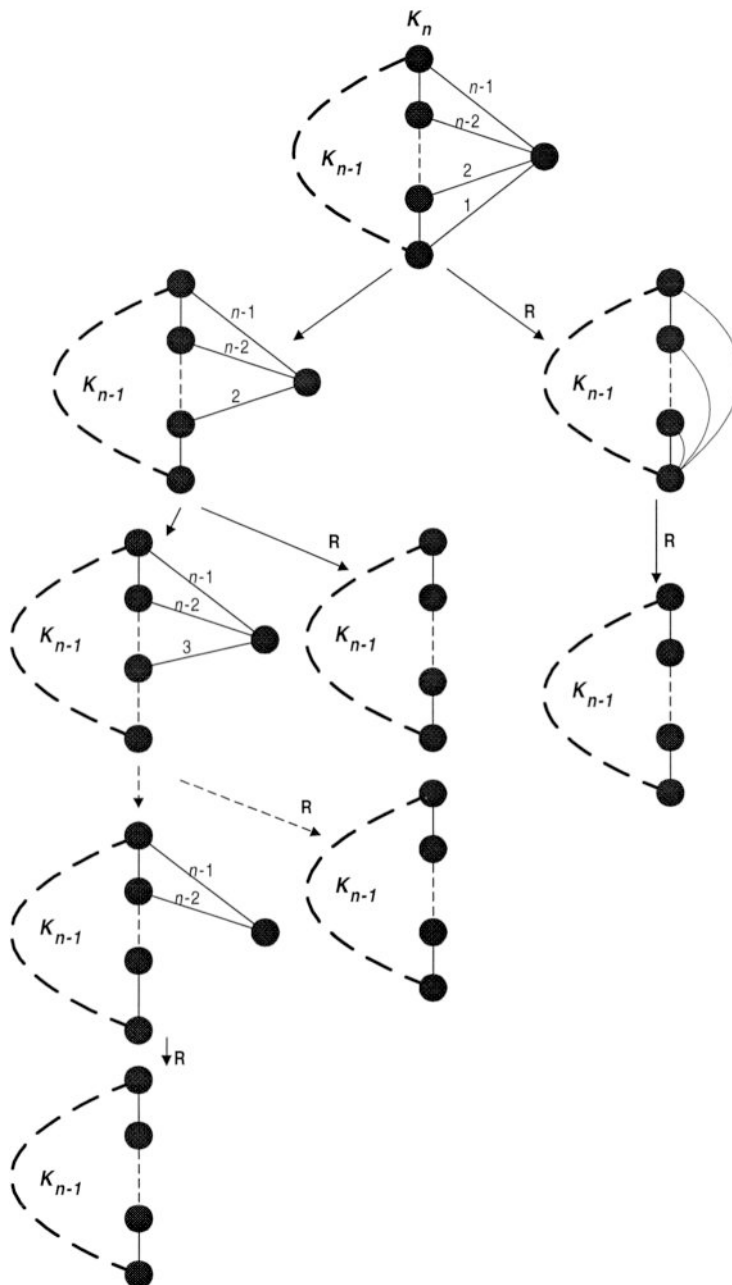
Faktoryzacja z wykorzystaniem strategii  $\mathbf{S}'$



Rysunek 1. Faktoryzacja z wykorzystaniem strategii  $\mathbf{S}'$  selekcji krawędzi grafu do faktoryzacji.

Dowód:

Na poniższym rysunku przedstawiony jest fragment BDO algorytmu faktoryzacji w przypadku obliczania niezawodności wszystkich terminali sieci reprezentowanej przez graf pełny.



**Rysunek 2. Fragment binarnego drzewa obliczeń algorytmu faktoryzacji ( $F_0$ ,  $\{R_0, R_1, R_2, R_3\}$ ,  $S'$ ) przy rozwiązywaniu problemu niezawodności wszystkich terminali sieci reprezentowanych przez graf pełny.**

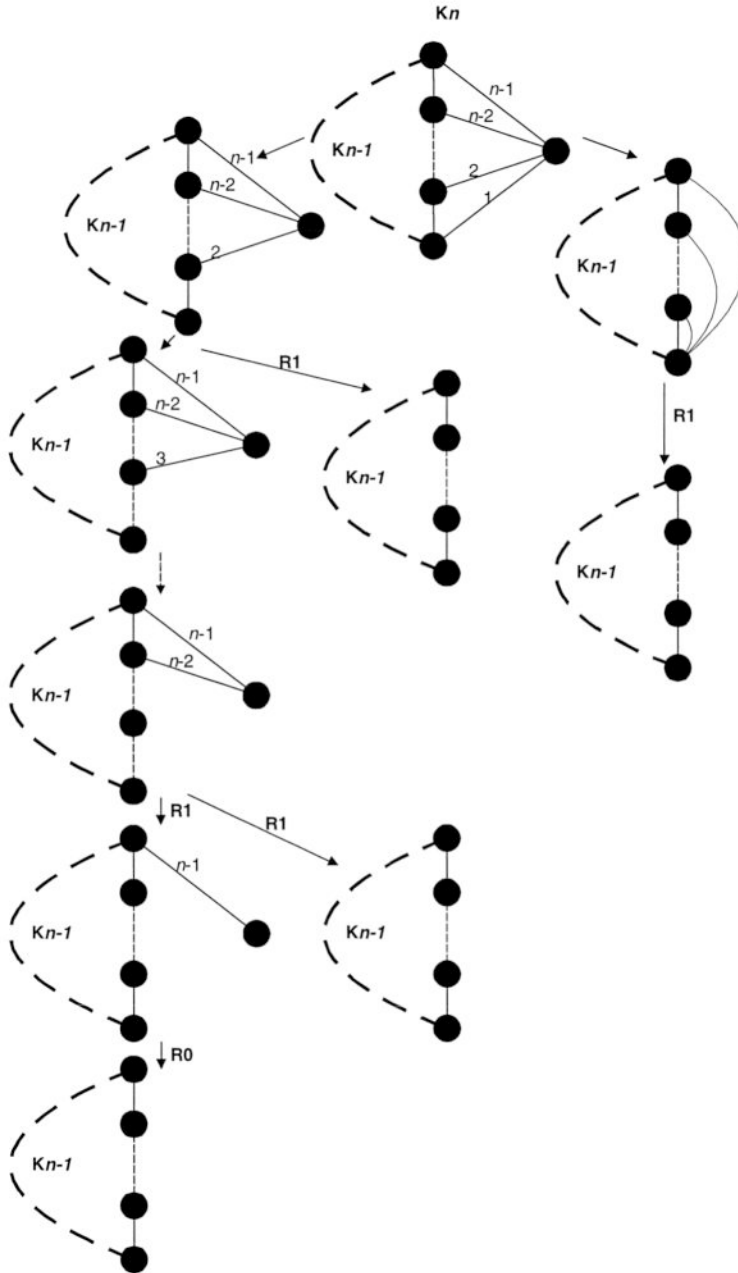
Wybieramy wierzchołek  $u$ , który ma  $n-1$  krawędzi incydentnych. Następnie  $n-3$  razy stosujemy formułę faktoryzacji. Otrzymujemy  $n-2$  węzłów BDO, w których występuje sieć  $K_{n-1}$ . Liczbę liści BDO algorytmu faktoryzacji można więc wyrazić rekurencyjnie:

$$L(K_n) = (n-2)L(K_{n-1})$$

$$L(K_1) = L(K_2) = L(K_3) = 1.$$

W efekcie liczba liści BDO algorytmu faktoryzacji ( $F_0, \{R_0, R_1, R_2, R_3\}, S'$ ) w przypadku obliczania niezawodności wszystkich terminali spełnia zależność  $L(G_v) \leq (n-2)!$  (redukcje  $R_2$  nie są używane). c.n.d.

Łatwo również zauważyć, że gdyby ze zbioru redukcji usunąć redukcję  $R_2$  i  $R_3$ , to interesujący nas fragment BDO będzie wyglądał następująco:



Rysunek 3. Fragment binarnego drzewa obliczeń algorytmu faktoryzacji ( $F_0, \{R_0, R_1\}, S'$ ) przy rozwiązywaniu problemu niezawodności wszystkich terminali sieci reprezentowanych przez graf pełny.

Otrzymujemy  $n-1$  węzłów BDO, w którym występuje sieć  $K_{n-1}$ , stąd  $L(K_n) = (n-1)L(K_{n-1}) = (n-1)! = (n-1) \left( (n-2)! \right)$ . W efekcie prawdziwe jest następujące twierdzenie.

**Twierdzenie 9**

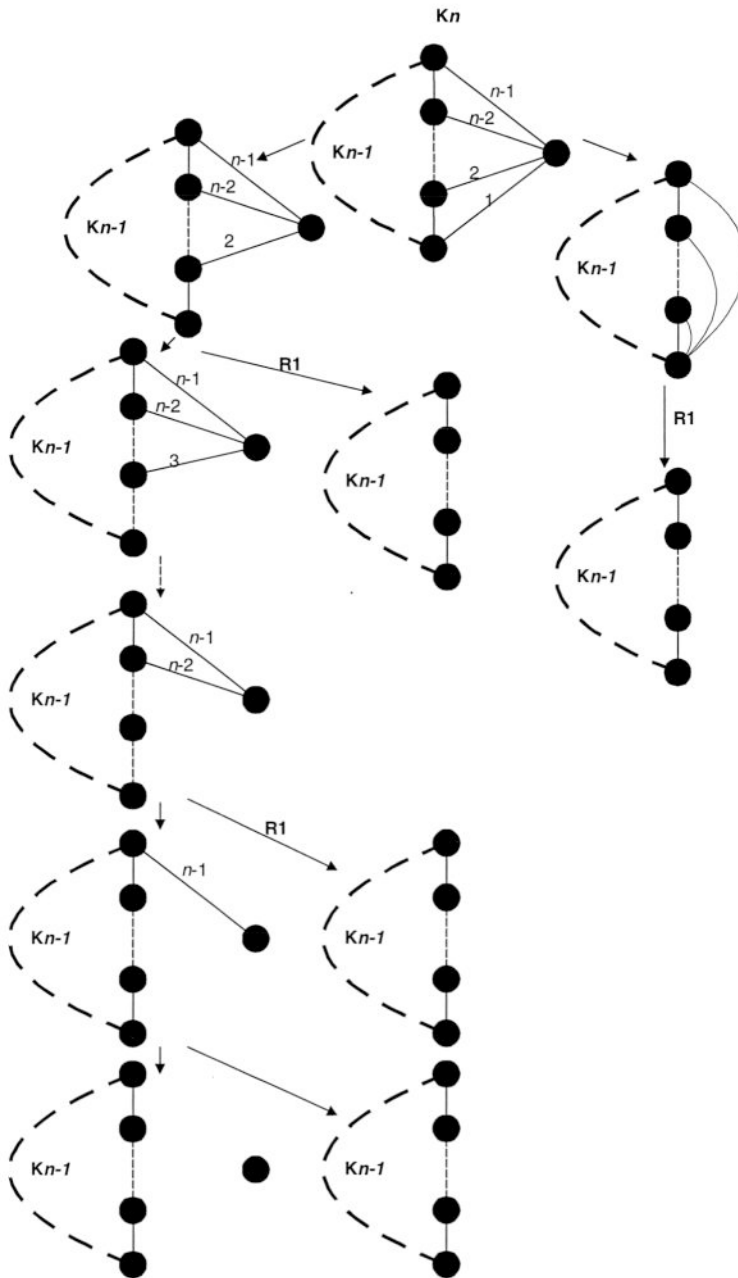
Liczba liści BDO algorytmu faktoryzacji  $(\mathbf{F}_0, \{R_0, R_1\}, \mathbf{S}')$  w przypadku obliczania niezawodności wszystkich terminali spełnia zależność  $L(G_v) \leq (n-1)!$ .

Gdyby ze zbioru redukcji usunąć tylko redukcję  $R_0$ , to liczba liści BDO ulegnie podwojeniu w stosunku do algorytmu bazowego  $(\mathbf{F}_0, \{R_0, R_1, R_2, R_3\}, \mathbf{S}')$ , gdyż  $L(K_2) = L(K_3) = 2$  (a nie 1), stąd  $L(K_n) = 2 \left( (n-2)! \right)$ . Prawdziwe jest więc następujące twierdzenie.

**Twierdzenie 10**

Liczba liści BDO algorytmu faktoryzacji  $(\mathbf{F}_0, \{R_1, R_2, R_3\}, \mathbf{S}')$  w przypadku obliczania niezawodności wszystkich terminali spełnia zależność  $L(G_v) \leq 2 \left( (n-2)! \right)$  (redukcje  $R_2$  nie są używane).

Jeśli usunęlibyśmy ze zbioru redukcji  $\mathbf{R} = \{R_0, R_1, R_2, R_3\}$  zarówno redukcję  $R_0$  jak i redukcje  $R_2$  i  $R_3$ , to fragment BDO algorytmu faktoryzacji  $(\mathbf{F}_0, \{R_1\}, \mathbf{S}')$  wyglądałby następująco:



Rysunek 4. Fragment binarnego drzewa obliczeń algorytmu faktoryzacji  $(F_0, \{R1\}, S')$  przy rozwiązywaniu problemu niezawodności wszystkich terminali sieci reprezentowanych przez graf pełny.

Wybieramy wierzchołek  $u$ , który ma  $n-1$  krawędzi incydentnych. Następnie  $n-1$  razy stosujemy formułę faktoryzacji. Otrzymujemy  $n-1$  węzłów BDO, w których występuje sieć  $K_{n-1}$  oraz jeden węzeł BDO, ze znajdującą się w nim siecią  $K_{n-1}$  i odizolowanym wierzchołkiem  $u$ . Ten węzeł BDO nie jest dalej rozwijany (jest liściem BDO), ponieważ wierzchołek  $u \in K$ , zatem nie wszystkie wierzchołki ze zbioru  $K$  są połączone. Liczbę liści BDO algorytmu faktoryzacji można więc wyrazić rekurencyjnie:

$$L(K_n) = (n-1)L(K_{n-1}) + 1.$$

Stosując metodę czynnika sumującego<sup>63</sup> stosunkowo łatwo jest rozwiązać powyższe równanie rekurencyjne. Metoda ta sprowadza się do pomnożenia obu stron równania rekurencyjnego przez czynnik sumacyjny, który w tym przypadku równa się  $\frac{1}{(n-1)!}$ .

W rezultacie otrzymujemy równanie:

$$\frac{L(K_n)}{(n-1)!} = \frac{(n-1)L(K_{n-1})}{(n-1)!} + \frac{1}{(n-1)!},$$

a po przekształceniu:

$$\frac{L(K_n)}{(n-1)!} = \frac{L(K_{n-1})}{(n-2)!} + \frac{1}{(n-1)!}.$$

Teraz dla  $S_n = \frac{L(K_n)}{(n-1)!}$  otrzymamy

$$S_n = S_{n-1} + \frac{1}{(n-1)!}.$$

Stąd wynika:

$$S_n = \sum_{k=1}^n \frac{1}{(k-1)!},$$

$$S_n = 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots = e \approx 2,718,$$

gdzie  $e$  jest podstawą logarytmów naturalnych.

W rezultacie otrzymujemy:

$$L(K_n) = S_n (n-1)! = e \left( (n-1)! \right) \approx 2,718 \left( (n-1)! \right).$$

Prawdziwe jest więc następujące twierdzenie.

#### **Twierdzenie 11**

Liczba liści BDO algorytmu faktoryzacji  $(\mathbf{F}_0, \{R1\}, \mathbf{S}')$  w przypadku obliczania niezawodności wszystkich terminali spełnia zależność  $L(G_v) \leq e \left( (n-1)! \right)$ .

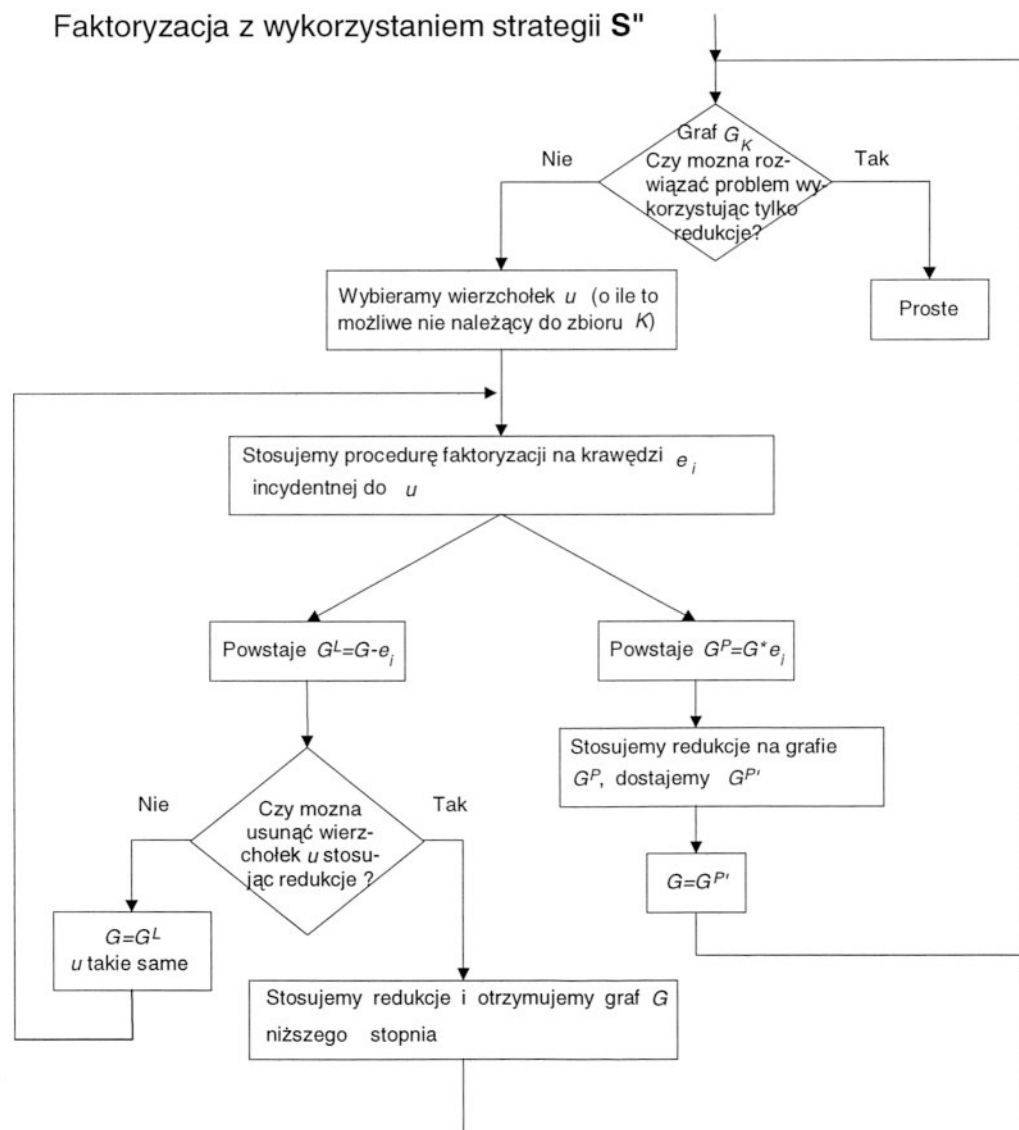
---

<sup>63</sup> Por. [Graham96], s.43.

### 5.1.2.2. Niezawodność $K$ -terminali

W rozdziale tym uogólnimy rozpatrywane w poprzednim rozdziale przypadki tak, by dotyczyły niezawodności  $K$ -terminali (gdy  $2 \leq |K| \leq |V|$ ), a nie tylko niezawodności wszystkich terminali (gdy  $|K| = |V|$ ).

Modyfikując strategię  $S'$  można uogólnić twierdzenie 8 na problem niezawodności  $K$ -terminali. Jeżeli nie wszystkie wierzchołki rozpatrywanego grafu należą do zbioru  $K$ , wystarczy w ramach stosowanej strategii selekcji krawędzi wybierać taki wierzchołek  $u$ , że  $u \notin K$ . Tak zmodyfikowaną strategię selekcji krawędzi  $S''$  przedstawia poniższy rysunek.



Rysunek 5. Faktoryzacja z wykorzystaniem strategii  $S''$  selekcji krawędzi grafu do faktoryzacji.



Modyfikacja strategii selekcji krawędzi grafu do faktoryzacji umożliwia redukcję wierzchołków stopnia 2:

- za pomocą redukcji R2 lub R3 w przypadku, gdy nie wszystkie wierzchołki grafu należą do zbioru  $K$ ;
- za pomocą redukcji R3 w przypadku, gdy wszystkie wierzchołki grafu należą do zbioru  $K$ .

Dzięki temu liczba liści BDO, a co za tym idzie również złożoność algorytmu *Fact* ( $\mathbf{F}_0, \{R0, R1, R2, R3\}, \mathbf{S}''$ ) w przypadku obliczania niezawodności  $K$ -terminali nie ulegnie zmianie w porównaniu do przypadku rozpatrywanego w twierdzeniu 8. Dlatego prawdziwe są poniższe twierdzenia.

#### **Twierdzenie 12**

Liczba liści BDO algorytmu faktoryzacji *Fact*, wykorzystującego zbiór redukcji  $\mathbf{R}=\{R0, R1, R2, R3\}$  oraz strategię  $\mathbf{S}''$  (określoną na rysunku powyżej), przy rozwiązywaniu problemu niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$  spełnia zależność  $L(G_K) \leq (n-2)!$ , gdzie  $n$  jest liczbą wierzchołków.

#### **Twierdzenie 13**

Pesymistyczna złożoność czasowa algorytmu faktoryzacji *Fact* ( $\mathbf{F}_0, \{R0, R1, R2, R3\}, \mathbf{S}''$ ), przy rozwiązywaniu problemu niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$  wynosi  $O(C(n,b)((n-2)!))$ .

Łatwo zauważyć, że gdyby ze zbioru redukcji usunąć redukcję R2 i R3, to dostaniemy  $n-1$  węzłów BDO, w którym występuje sieć  $K_{n-1}$ , stąd  $L(K_n) = (n-1)L(K_{n-1}) = (n-1)! = (n-1) \left( (n-2)! \right)$ . W efekcie prawdziwe są następujące twierdzenia.

#### **Twierdzenie 14**

Liczba liści BDO algorytmu faktoryzacji ( $\mathbf{F}_0, \{R0, R1\}, \mathbf{S}''$ ) w przypadku obliczania niezawodności  $K$ -terminali spełnia zależność  $L(G_K) \leq (n-1)!$ .

**Twierdzenie 15**

Pesymistyczna złożoność czasowa algorytmu faktoryzacji  $(F_0, \{R_0, R_1\}, S'')$ , przy rozwiązywaniu problemu niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$  wynosi  $O(C(n,b)((n-1)!))$ .

Gdyby ze zbioru redukcji  $R=\{R_0, R_1, R_2, R_3\}$  usunąć tylko redukcję  $R_0$ , to (podobnie jak w przypadku obliczania niezawodności wszystkich terminali) liczba liści BDO ulegnie podwojeniu, gdyż  $L(K_2)=L(K_3)=2$  (a nie 1), stąd  $L(K_n)=2\left((n-2)!\right)$ . W efekcie prawdziwe są następujące twierdzenia.

**Twierdzenie 16**

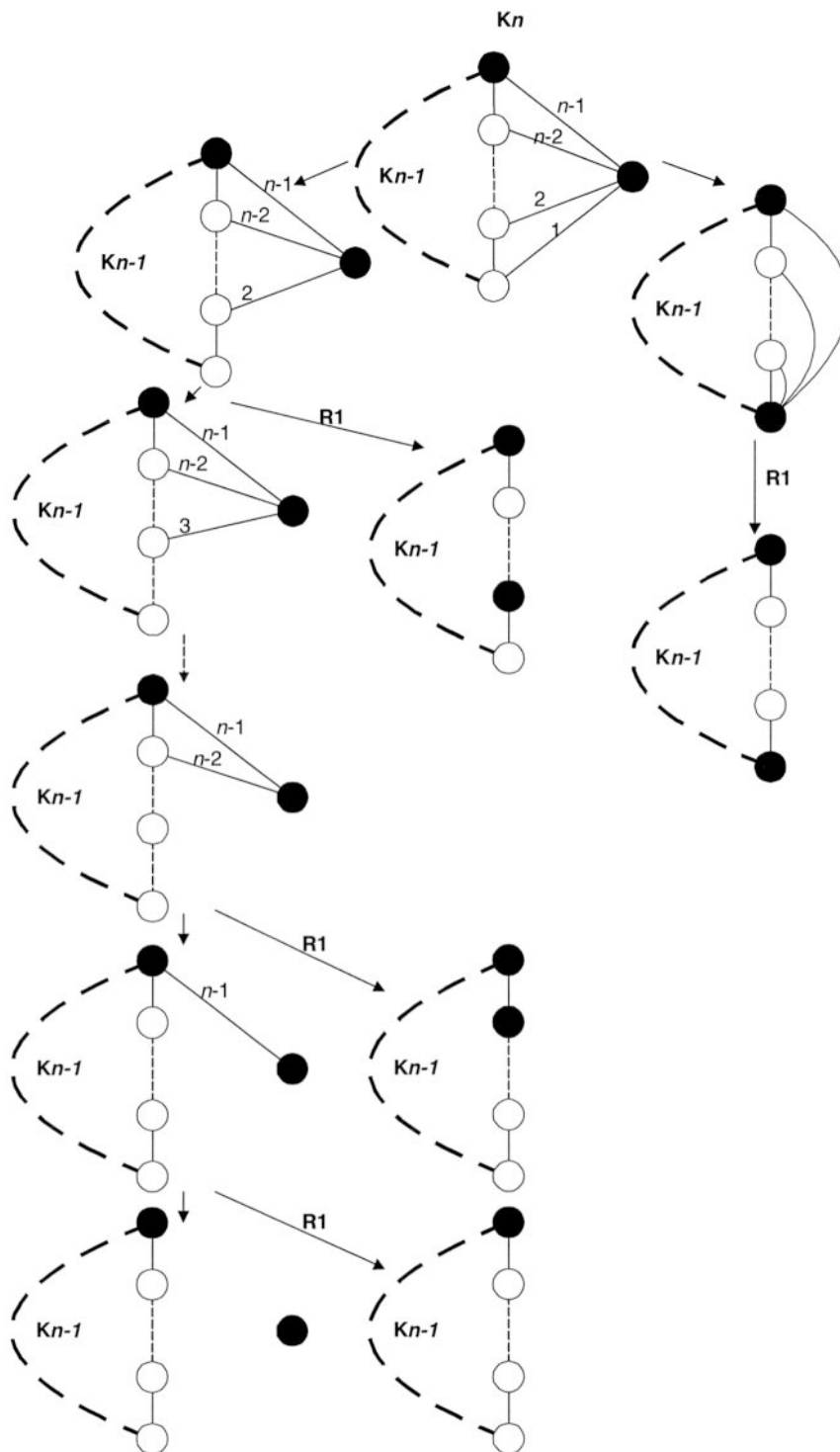
Liczba liści BDO algorytmu faktoryzacji  $(F_0, \{R_1, R_2, R_3\}, S'')$  w przypadku obliczania niezawodności  $K$ -terminali spełnia zależność  $L(G_K) \leq 2\left((n-2)!\right)$ .

**Twierdzenie 17**

Pesymistyczna złożoność czasowa algorytmu faktoryzacji  $(F_0, \{R_1, R_2, R_3\}, S'')$ , przy rozwiązywaniu problemu niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$ , wynosi  $O(C(n,b)((n-2)!))$ .

Jeśliśmy ze zbioru redukcji  $R=\{R_0, R_1, R_2, R_3\}$  usunęli zarówno redukcję  $R_0$ , jak i redukcje  $R_2$  i  $R_3$ , to strategia  $S''$  sprawia, że algorytm staje się bardzo nieefektywny. Modyfikując strategię  $S''$  tak, by wierzchołek  $u \in K$  otrzymujemy strategię  $S'''$  i znacznie efektywniejszy algorytm.

W wyniku tej modyfikacji fragment BDO algorytmu faktoryzacji  $(F_0, \{R_1\}, S''')$  przy rozwiązywaniu problemu niezawodności dwóch terminali wyglądałby następująco:



Rysunek 6. Fragment binarnego drzewa obliczeń algorytmu faktoryzacji ( $F_0, \{R1\}, S'''$ ) przy rozwiązywaniu problemu niezawodności dwóch terminali sieci reprezentowanych przez graf pełny.

Dostajemy  $n$  węzłów BDO z występującą w nich siecią  $K_{n-1}$ . Jednak dwa ostatnie węzły BDO nie są dalej rozwijane (są liśćmi BDO), ponieważ w pierwszym przypadku jest tylko jeden wierzchołek w zbiorze  $K$  (niezawodność takiej sieci wynosi jeden, gdyż wszystkie wierzchołki ze zbioru  $K$  są połączone), w drugim zaś przypadku wierzchołek

$u \in K$  (niezawodność takiej sieci wynosi zero, gdyż wierzchołki ze zbioru  $K$  nie są połączone). Liczbę liści BDO algorytmu faktoryzacji ( $F_0, \{R1\}, S'''$ ) przy rozwiązywaniu problemu niezawodności dwóch terminali można więc wyrazić rekurencyjnie:

$$L(K_n) = (n-2)L(K_{n-1}) + 2$$

Stosując podobnie jak poprzednio metodę czynnika sumującego można łatwo rozwiązać powyższe równanie rekurencyjne. Metoda ta sprowadza się do pomnożenia obu stron równania rekurencyjnego przez czynnik sumacyjny, w tym przypadku równy  $\frac{1}{(n-2)!}$ .

W rezultacie otrzymujemy równanie:

$$\frac{L(K_n)}{(n-2)!} = \frac{(n-2)L(K_{n-1})}{(n-2)!} + \frac{2}{(n-2)!},$$

a po przekształceniu:

$$\frac{L(K_n)}{(n-2)!} = \frac{L(K_{n-1})}{(n-3)!} + \frac{2}{(n-2)!}.$$

Teraz dla  $S_n = \frac{L(K_n)}{(n-2)!}$  otrzymamy:

$$S_n = S_{n-1} + \frac{2}{(n-2)!}.$$

Stąd wynika:

$$S_n = \sum_{k=2}^n \frac{2}{(k-2)!} = 2 \sum_{k=2}^n \frac{1}{(k-2)!},$$

$$S_n = 2 \left( 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots \right) = 2e \approx 5,437,$$

gdzie  $e$  jest podstawą logarytmów naturalnych.

$$L(K_n) = S_n (n-2)! = 2e \left( (n-2)! \right) \approx 5,437 \left( (n-2)! \right).$$

Prawdziwe są więc następujące twierdzenia.

**Twierdzenie 18**

Liczba liści BDO algorytmu faktoryzacji  $(\mathbf{F}_0, \{\mathbf{R1}\}, \mathbf{S}''')$  w przypadku obliczania niezawodności dwóch terminali spełnia zależność  $L(G_K) \leq 2e \binom{n-2}{2}$ .

**Twierdzenie 19**

Pesymistyczna złożoność czasowa algorytmu faktoryzacji  $(\mathbf{F}_0, \{\mathbf{R1}\}, \mathbf{S}''')$ , przy rozwiązywaniu problemu niezawodności dwóch terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$  wynosi  $O(C(n, b)((n-2)!))$ .

Można zauważyć, że w ogólnym przypadku obliczania niezawodności  $K$ -terminali liczba liści BDO algorytmu faktoryzacji  $(\mathbf{F}_0, \{\mathbf{R1}\}, \mathbf{S}''')$  na sieci reprezentowanej poprzez graf pełny wyrażona jest wzorem rekurencyjnym:

$$L(K_{n,k}) = (n-k)L(K_{n-1,k}) + (k-1)L(K_{n-1,k-1}) + 1$$

$$\forall i > 0 \quad L(K_{i,1}) = 1,$$

gdzie  $K_{n,k}$  jest grafem pełnym o  $n$  wierzchołkach, spośród których  $k$  należy do zbioru  $K$  (tzn.  $|K| = k$ ).

Powyższa definicja rekurencyjna wyśmienicie nadaje się do dowodu indukcyjnego:

$$\text{dla } k = 2 \quad L(K_{n,2}) = (n-2)L(K_{n-1,2}) + L(K_{n-1,1}) + 1 = (n-2)L(K_{n-1,2}) + 2;$$

$$\text{dla } k = n \quad L(K_{n,n}) = (n-1)L(K_{n-1,n-1}) + 1,$$

co jest zgodne z uzyskanymi wcześniej wynikami.

Krok indukcyjny wynika dla  $n > k > 2$  natychmiast, gdy założymy, że wzór na  $L(K_{n,k})$  zachodzi dla  $k$  zastąpionego przez  $k-1$ .

**5.1.3. Złożoność pamięciowa algorytmów faktoryzacji**

Jak podkreślono w podsumowaniu rozdziału 4. rozmiar wymaganej pamięci można ogólnie oszacować poprzez iloczyn rozmiaru sieci  $|G|$  (t.j. zestawu danych, wykorzystanych głównie do reprezentacji sieci w pojedynczej instancji rekurencyjnej funkcji *Prob*) i

głębokości rekurencji  $d(G_K)$ . Stąd zgrubnym pesymistycznym oszacowaniem złożoności pamięciowej algorytmów faktoryzacji jest  $O(b|G_K|)$ , gdzie  $b$  jest liczbą krawędzi grafu reprezentującego badaną sieć.

Pesymistyczną złożoność pamięciową algorytmów faktoryzacji można również, podobnie jak w przypadku złożoności czasowej, w prosty sposób analizować wykorzystując grafy pełne. Głębokość rekurencji przy rozwiązywaniu sieci reprezentowanych przez  $G_K$  nie będzie większa niż w przypadku grafu pełnego. Uzyskane w ten sposób oszacowania mogą być jednak (szczególnie w przypadku grafów rzadkich) gorsze od oszacowania  $O(b|G_K|)$ .

Jak można zaobserwować na rysunku 2. głębokość rekurencji algorytmu faktoryzacji, wykorzystującego zbiór redukcji  $\mathbf{R}=\{R0, R1, R2, R3\}$  oraz strategię  $\mathbf{S}'$  lub  $\mathbf{S}''$  w przypadku rozwiązywania problemu niezawodności  $K$ -terminali sieci reprezentowanej poprzez graf pełny  $K_n$  spełnia zależność  $d(K_n)=(n-3)+d(K_{n-1})$ . Stąd:

$$d(K_n) = \frac{1+(n-3)}{2}(n-3) = \frac{(n-2)(n-3)}{2} = \frac{n^2-5n+6}{2} = \frac{n(n-1)-4n+6}{2} = b-2n+3.$$

#### **Twierdzenie 20**

Głębokość rekurencji algorytmu faktoryzacji *Fact*, wykorzystującego zbiór redukcji  $\mathbf{R}=\{R0, R1, R2, R3\}$  oraz strategię  $\mathbf{S}''$ , przy rozwiązywaniu problemu niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$ , spełnia zależność

$$d(G_K) \leq \frac{(n-2)(n-3)}{2}, \text{ gdzie } n \text{ jest liczbą wierzchołków.}$$

#### **Twierdzenie 21**

Pesymistyczna złożoność pamięciowa algorytmu faktoryzacji *Fact* ( $\mathbf{F}_0, \{R0, R1, R2, R3\}, \mathbf{S}''$ ), przy rozwiązywaniu problemu niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$  wynosi  $O\left(\frac{(n-2)(n-3)}{2}|G_K|\right)$ .

Łatwo zauważyć (rysunek 3), że gdyby ze zbioru redukcji  $\mathbf{R}=\{R0, R1, R2, R3\}$  usunąć redukcję  $R2$  i  $R3$ , to dostaniemy  $n-2$  zagłębionych rekurencyjnie węzłów BDO, w którym występuje sieć  $K_{n-1}$ , stąd  $d(K_n)=(n-2)+d(K_{n-1})$ .

Przekształcając otrzymujemy:

$$d(K_n) = \frac{1+(n-2)}{2}(n-2) = \frac{(n-1)(n-2)}{2} = \frac{(n-1)n-2n+2}{2} = b-n+1.$$

W efekcie prawdziwe są następujące twierdzenia.

**Twierdzenie 22**

Głębokość rekurencji algorytmu faktoryzacji  $(F_0, \{R0, R1\}, S'')$ , w przypadku obliczania niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$ , spełnia zależność  $d(G_K) \leq \frac{(n-1)(n-2)}{2}$ .

**Twierdzenie 23**

Pesymistyczna złożoność pamięciowa algorytmu faktoryzacji  $(F_0, \{R0, R1\}, S'')$ , przy rozwiązywaniu problemu niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$  wynosi  $O\left(\frac{(n-1)(n-2)}{2}|G_K|\right)$ .

Satyanarayana i Chang [Satyanarayana83]<sup>64</sup> analizowali złożoność obliczeniową zaproponowanego przez siebie algorytmu faktoryzacji wykorzystującego redukcje R1 i R2, wykrywającego ponadto, czy graf jest drzewem. Zastosowana strategia selekcji krawędzi wymagała, by wybrana krawędź  $e_i$  gwarantowała, że  $D(G_K - e_i) \neq 0$  oraz  $D(G_K * e_i) \neq 0$ . Satyanarayana i Chang oszacowali pesymistyczną złożoność pamięciową tego algorytmu przez  $O((b-n+1)|G_K|)$ . Nie przedstawili oni jednak sposobu, w jaki powyższy rezultat uzyskali.

Jeżeliby ze zbioru redukcji  $R=\{R0, R1, R2, R3\}$  usunąć tylko redukcję R0, to głębokość rekurencji ulegnie zwiększeniu o jeden, gdyż  $d(K_3)=d(K_2)=2$  (a nie 1), stąd

$$d(K_n) = \frac{(n-2)(n-3)}{2} + 1 = b - 2n + 4. \text{ W efekcie prawdziwe są następujące twierdzenia.}$$

---

<sup>64</sup> Por. [Satyanarayana83], s.114.

**Twierdzenie 24**

Głębokość rekurencji algorytmu faktoryzacji  $(F_0, \{R1, R2, R3\}, S'')$ , w przypadku obliczania niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$ , spełnia zależność  $d(G_K) \leq \frac{(n-2)(n-3)}{2} + 1$ .

**Twierdzenie 25**

Pesymistyczna złożoność pamięciowa algorytmu faktoryzacji  $(F_0, \{R1, R2, R3\}, S'')$ , przy rozwiązywaniu problemu niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$  wynosi  $O\left(\left(\frac{(n-2)(n-3)}{2} + 1\right) |G_K|\right)$ .

Jeślibyśmy ze zbioru redukcji  $R=\{R0, R1, R2, R3\}$  usunęli zarówno redukcję  $R0$ , jak i redukcje  $R2$  i  $R3$ , to strategia  $S''$  staje się bardzo nieefektywna. Modyfikując strategię  $S''$  tak, by wierzchołek  $u \in K$ , otrzymujemy strategię  $S'''$ . W wyniku tej modyfikacji fragment BDO algorytmu faktoryzacji  $(F_0, \{R1\}, S''')$  przy rozwiązywaniu problemu niezawodności wszystkich terminali i niezawodności dwóch terminali wyglądałby tak, jak to przedstawiono na rysunkach 4. i 6. Ponieważ parametry zbioru  $K$  nie mają wpływu na głębokość rekurencji, można rozpatrywać ogólny przypadek niezawodności  $K$ -terminali.

Dostajemy  $n-1$  zagłębionych rekurencyjnie węzłów BDO, w których występuje sieć  $K_{n-1}$ . Głębokość rekurencji algorytmu faktoryzacji  $(F_0, \{R1\}, S''')$  przy rozwiązywaniu problemu niezawodności  $K$ -terminali można więc wyrazić rekurencyjnie:

$$d(K_n) = (n-1) + d(K_{n-1}).$$

Stąd:

$$d(K_n) = \frac{1+(n-1)}{2}(n-1) = \frac{n(n-1)}{2} = b.$$

Prawdziwe są więc następujące twierdzenia.

**Twierdzenie 26**

Głębokość rekurencji algorytmu faktoryzacji  $(F_0, \{R1\}, S''')$ , w przypadku obliczania niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$  spełnia zależność  $d(G_K) \leq \frac{n(n-1)}{2}$ .



### Twierdzenie 27

Pesymistyczna złożoność pamięciowa algorytmu faktoryzacji  $(\mathbf{F}_0, \{\mathbf{R1}\}, \mathbf{S}''')$ , przy rozwiązywaniu problemu niezawodności  $K$ -terminali dowolnej sieci reprezentowanej poprzez graf  $G_K$ , wynosi  $O\left(\frac{n(n-1)}{2}|G_K|\right)$ .

## 5.2. PODSUMOWANIE ROZDZIAŁU

Przedstawiony w tym rozdziale algorytm *Fact*  $(\mathbf{F}_0, \{\mathbf{R0}, \mathbf{R1}, \mathbf{R2}, \mathbf{R3}\}, \mathbf{S}'')$  jest modyfikacją algorytmu faktoryzacji przedstawionego przez Page'a i Perry [Page88]. Dzięki zaproponowaniu nowej strategii  $\mathbf{S}''$  selekcji krawędzi do faktoryzacji, odpowiednio dobranego zbioru redukcji, jak również nowej metody analizy pesymistycznej złożoności obliczeniowej algorytmu wykazano, że maksymalna liczba  $(|V|-2)!$  liści binarnego drzewa obliczeń algorytmu faktoryzacji jest osiągalna dla dowolnego zbioru  $K$  ( $2 \leq |K| \leq |V|$ ), a nie tylko dla przypadków granicznych, gdy  $2 \leq |K| \leq 5$  oraz  $|V|-2 \leq |K| \leq |V|$  (jak to ma miejsce w przypadku algorytmu analizowanego przez Wooda [Wood85, Wood86]). Rezultat ten uzyskano przy prostszym niż zaproponowany przez Wooda zbiorze zachowujących niezawodność redukcji grafu  $\mathbf{R}=\{\mathbf{R0}, \mathbf{R1}, \mathbf{R2}, \mathbf{R3}\}$ . Nie ma bowiem konieczności uwzględniania najbardziej skomplikowanego podzbioru redukcji  $\mathbf{R4}$  (redukcji wielokąta do łańcucha). Również zaproponowana strategia selekcji krawędzi do faktoryzacji nie wymaga stosowania złożonych procedur testowania własności (np. koherentności czy dwuspójności) sieci występujących w poszczególnych węzłach BDO.

Zaprezentowana w tym rozdziale metoda analizy pesymistycznej złożoności obliczeniowej znalazła zastosowanie do analizy algorytmu *Fact*, jak również odmian tego algorytmu, wykorzystujących różne zbiory zachowujących niezawodność redukcji grafów. Złożoność czasowa tych algorytmów nie była dotąd analizowana a uzyskane rezultaty ilustrują wpływ zastosowanych redukcji grafu na pesymistyczną złożoność obliczeniową algorytmów faktoryzacji. Dla zaproponowanego algorytmu *Fact* rezultaty uzyskane analitycznie zostały potwierdzone przez testy empiryczne (rozdział 7.3.2).

## 6. ALGORYTM FAKTORYZACJI *FACT&CACHE* Z WYKRYWANIEM RÓWNOWAŻNYCH PODPROBLEMÓW

Badając znany algorytm faktoryzacji można zauważyć, iż w trakcie obliczeń te same podproblemy (sieci powstałe w wyniku stosowania faktoryzacji i redukcji) są często rozwiązywane wiele razy. Zamiast wielokrotnie rozwiązywać ten sam podproblem (obliczać niezawodność tej samej sieci), jak ma to miejsce w przypadku innych algorytmów faktoryzacji, znacznie efektywniej byłoby użyć wcześniej obliczony i zapamiętany rezultat obliczeń, wykorzystując zmodyfikowaną technikę programowania dynamicznego [Madeyski99a]. Ten bardzo istotny fakt nie został jak dotąd dostrzeżony i praktycznie wykorzystany.

Nowy algorytm faktoryzacji *Fact&Cache* można opisać w postaci następującej piątki **(Fc, R, C, S1, S2)**, gdzie:

**Fc** - szkielet algorytmu *Fact&Cache* (przedstawiony poniżej),

**R** - zbiór zachowujących niezawodność redukcji,

**C** - algorytm generowania kodu grafu,

**S1** - strategia wybierania krawędzi do faktoryzacji (dekompozycji problemu),

**S2** - strategia ograniczania ilości przechowywanych rozwiązań, decyduje, czy odrzucić, czy zarejestrować (ewentualnie kosztem wcześniej zarejestrowanego rozwiązania) kod  $c$  podproblemu i jego rozwiązanie  $r$ .

Parametry wejściowe: Graf nieskierowany  $G_K$  z odpowiednimi niezawodnościami krawędzi i wyróżnionym podzbiorem wierzchołków  $K$ . Graf nie zawiera krawędzi równoległych i pętli własnych.

Rezultat: Niezawodność  $K$ -terminali grafu  $G_K$ .

$\text{Prob}(G_K)$

```
{
    if (|K|==1)
        return(1);
    else
    {
        Wygeneruj kod  $c$  grafu  $G_K$  wykorzystując algorytm C;
        if (podproblem o kodzie  $c$  jest zarejestrowany)
            return( $r(c)$ );
        else
        {
```

```

Wybierz krawędź  $e$  używając strategii S1;
Usuń wybraną krawędź  $e$ ;
if(węzły w  $K$  są rozłączone)
     $p1=0$ 
else
{
    Przeprowadź redukcję redukując  $G_K * e$  do postaci  $G'_{K'}$ ;
     $p1 = \Omega_2 * \text{Prob}(G'_{K'})$ ; //gdzie  $\Omega_2$  jest odpowiednim
        //współczynnikiem
    }
    Połącz wierzchołki na obu końcach krawędzi  $e$  tworząc  $G_K * e$ ;
    Przeprowadź redukcję redukując  $G_K * e$  do postaci  $G'_{K'}$ ;
     $p2 = \Omega_2 * \text{Prob}(G'_{K'})$ ; //gdzie  $\Omega_2$  jest odpowiednim wsp.
     $r = ((1-p) * p1 + p * p2)$ ; //gdzie  $p$  niezawodnością krawędzi  $e$ .
    Stosując strategię S2 ograniczania ilości przechowywanych
    rozwiązań odrzuć bądź zarejestruj kod  $c$  podproblemu i jego
    rozwiązanie  $r$ ;
    return( $r$ );
}
}
}

main()
{
    analiza grafu  $G_K$  (przypisanie priorytetów krawędziom grafu, co
    wstępnie określa S1);
     $K\text{TermRel} = \text{Prob}(G_K)$ ;
}

```

**S1** = {Musi być wybrana krawędź  $e = (u, v) \in E$  o maksymalnym priorytecie (priorytety przydzielane są we wstępnej fazie analizy grafu, jak również po zastosowaniu zachowujących niezawodność redukcji grafu)}

**S2** = {Rejestruj każdy rozwiązywany podproblem (w sytuacji gdy bufor jest pełny rejestruj kosztem najwcześniej zarejestrowanego i pamiętanego rozwiązania)}.

Wykonanie algorytmu można podzielić na dwie fazy, wstępną – w której dokonywana jest analiza grafu i zasadniczą – w której obliczana jest niezawodność  $K$ -terminali metodą faktoryzacji z wykorzystaniem techniki częściowego spamiętywania będącej modyfikacją techniki programowania dynamicznego. Efektywność stosowanej w algorytmie techniki częściowego spamiętywania zależy w dużym stopniu od strategii dekompozycji problemu i skuteczności wykrywania równoważności podproblemów (sieci) występujących w binarnym drzewie obliczeń.

## 6.1. ZMODYFIKOWANA TECHNIKA PROGRAMOWANIA DYNAMICZNEGO

Metoda programowania dynamicznego pozwala uniknąć wielokrotnego rozwiązywania tych samych podproblemów. Tworzona jest tablica z zapamiętanymi rozwiązaniami najmniejszych podproblemów (czasami określanymi mianem podproblemów elementarnych), a następnie obliczonych na ich podstawie rozwiązań podproblemów większych rozmiarów. Proces ten jest kontynuowany, aż do uzyskania rozwiązania problemu pierwotnego, przy czym raz znalezione rozwiązanie podproblemu zostaje zapamiętane i może być wykorzystywane bez konieczności ponownego obliczania.

Ze względu na naturę problemu rozpatrywanego w tej pracy trudno jest określić na wstępie podproblemy elementarne i ich rozwiązania, a następnie sposób, formułę otrzymania na ich podstawie rozwiązania problemu pierwotnego. Z tego względu należy użyć zmodyfikowaną technikę programowania dynamicznego wykorzystującą strategię zstępującą.

Użyta w zaproponowanym algorytmie *Fact&Cache* technika częściowego spamiętywania [Madeyski99a] jest modyfikacją techniki spamiętywania określanej w języku angielskim terminem *memoization*<sup>65</sup>. Obie są odmianami techniki programowania dynamicznego, wykorzystującymi strategię zstępującą.

Algorytm wykorzystujący technikę spamiętywania ma formę algorytmu rekurencyjnego z dodanymi funkcjami zapamiętywania parametrów podproblemów wraz z ich rozwiązaniami oraz przeszukiwania zarejestrowanych podproblemów, co zapobiega wielokrotnemu ich rozwiązywaniu.

Algorytm wykorzystujący iteracyjną technikę programowania dynamicznego ze strategią wstępującą, charakteryzuje się podobną złożonością czasową jak algorytm stosujący technikę spamiętywania (różnica o stały czynnik, na korzyść programowania dynamicznego, ze względu na „koszt” wywołań rekurencyjnych i powrotów).

Gdy rozwiązanie problemu wymaga rozwiązania jedynie niektórych spośród wszystkich możliwych podproblemów, to zastosowanie strategii zstępującej zamiast wstępującej zapewnia, że rozpatrywane będą tylko te podproblemy, których rozwiązanie jest niezbędne. Dzięki temu algorytm taki charakteryzować się może mniejszą złożonością pamięciową i czasową, niż algorytm wykorzystujący strategię wstępującą.

---

<sup>65</sup> Por. [Cormen97], s.358.

Technika częściowego spamiętywania [Madeyski99a] umożliwia ograniczenie złożoności pamięciowej algorytmu. Modyfikacja w stosunku do techniki spamiętywania polega na tym, że nie są przechowywane wszystkie rozwiązania, a jedynie te, które – jak przypuszczamy – mogą znacząco wpłynąć na przyspieszenie algorytmu (np. ostatnio zarejestrowane rozwiązania – jeżeli przypuszczamy, że będą one częściej wykorzystywane niż te zarejestrowane wcześniej). Koncepcja ta jest podobna do koncepcji wykorzystania pamięci podręcznej *cache*, stąd nazwa nowego algorytmu *Fact&Cache*.

W przypadku zastosowania techniki częściowego spamiętywania do rozwiązywania innych problemów należy pamiętać o tym, że zarówno strategia ograniczania ilości przechowywanych rozwiązań jak również strategia dekompozycji problemu pierwotnego mogą mieć wpływ na efektywność algorytmu.

## 6.2. WYKRYWANIE RÓWNOWAŻNYCH PODPROBLEMÓW

Aby móc wielokrotnie wykorzystywać raz obliczone rozwiązania podproblemów, konieczne jest posiadanie mechanizmu umożliwiającego wykrywanie równoważnych podproblemów (sieci) występujących w poszczególnych węzłach binarnego drzewa obliczeń.

### Definicja 1 (definicja równoważności sieci)

Dwie sieci są równoważne, jeśli ich grafy są wzajemnie izomorficzne; odpowiadające sobie wierzchołki obu grafów są zawarte w tym samym zbiorze wierzchołków  $K$  lub jego dopełnieniu i niezawodności przypisane odpowiadającym sobie krawędziom grafu są takie same.

### Własność 5 (własność sieci równoważnych)

Jeżeli sieci  $G_K$  i  $H_{K'}$  są równoważne, to niezawodności  $K$ -terminali tych sieci są równe  $R(G_K) = R(H_{K'})$ .

Trudność wykrywania równoważnych podproblemów wynika głównie z konieczności testowania izomorfizmu grafów otrzymywanych w poszczególnych węzłach binarnego drzewa obliczeń. Efektywne testowanie izomorfizmu grafów stanowi jednak temat intensywnych badań i wciąż nie jest znany wielomianowy (w najgorszym przypadku) algorytm testowania izomorfizmu dowolnych grafów. W związku z tym zostanie teraz przedstawiony sposób ominięcia tego problemu.

Dla danych dwóch sieci reprezentowanych przez grafy  $G_K$  i  $H_K$ , chcemy określić, czy są one równoważne. Jednym ze sposobów rozwiązania tego zagadnienia jest konstruowanie kodu dla każdej sieci. Dwie sieci będą równoważne wtedy i tylko wtedy, gdy ich kody będą takie same. Po obliczeniu niezawodności sieci jej kod wraz z odpowiadającą mu wartością niezawodności są zapisywane w pamięci *cache*. Jeżeli w trakcie pracy algorytmu okaże się, iż wygenerowany kod już wcześniej wystąpił, to węzeł drzewa obliczeń, w którym ten fakt zostanie wykryty, nie jest dalej rozwijany, lecz zwracana jest obliczona poprzednio, a przyporządkowana temu kodowi, niezawodność sieci.

Idealny kod  $c$  do badania izomorfizmu grafów posiadałby następujące własności:

#### **Własność 6**

Jeżeli kod  $c$  jest taki sam dla grafów  $G$  i  $H$ , to są one wzajemnie izomorficzne.

#### **Własność 7**

Jeżeli graf  $G$  jest izomorficzny z  $H$ , to ich kody  $c$  są sobie równe.

O ile własność 6 musi być zawsze spełniona, aby nie zniekształcić ostatecznego wyniku, to wystarczy, aby własność 7 była spełniona możliwie często. Jedyną zaś konsekwencją tego jest możliwość niewykrycia izomorfizmu grafów, czyli powtórnego obliczania niezawodności niektórych sieci.

### **6.2.1. Metoda kodowania podproblemów**

W tym rozdziale zostanie przedstawiona bardzo prosta metoda wykrywania sieci równoważnych spełniająca własność 6, ale nie zawsze własność 7. Osłabienie warunku co do spełnialności własności 7 umożliwia przyspieszenie procesu testowania izomorfizmu grafów (w najgorszym przypadku zamiast algorytmu wykładniczego otrzymujemy algorytm wielomianowy) kosztem niewykrycia izomorfizmu niektórych grafów. Oznacza to, że zaproponowana metoda nie gwarantuje wykrycia wszystkich grafów izomorficznych.

Kod sieci składa się z następujących części:

- część odpowiedzialna za możliwie częste wykrywanie izomorfizmu grafów (spełnia własność 6, ale nie zawsze własność 7);
- część identyfikująca wyróżniony zbiór węzłów  $K$ ;
- część zawierająca ilość łączy i ich niezawodności.

Testowanie izomorfizmu grafów można zrealizować wykorzystując macierz przyległości  $A$ . Załóżmy, że graf  $G$  o liczbie węzłów  $|V|$  jest grafem nieskierowanym i prostym, czyli bez krawędzi równoległych i pętli własnych. Macierz przyległości grafu nieskierowanego jest symetryczna, wystarczy więc zapisać w postaci wektora bitów tylko jej górny (lub dolny) trójkąt, stąd rozmiar kodu  $|V|*(|V|-1)/2$  bitów. Wygenerowany kod można traktować jako binarną reprezentację pewnej wartości całkowitej zwanej dalej wartością kodu. Ponieważ jednak sposób przypisania każdemu z wierzchołków grafu etykiet 1, 2, ...,  $|V|$  ma wpływ na wartość wygenerowanego kodu, to własność 7 raczej rzadko byłaby spełniona.

Jednym z rozwiązań tego problemu jest takie przypisanie etykiet poszczególnym wierzchołkom, które dawałoby maksymalną albo minimalną wartość kodu. Dzięki temu własność 7 byłaby zawsze spełniona. Niestety, przebadanie wszystkich możliwych permutacji węzłów jest zbyt czasochłonne. Można zwiększyć efektywność kodowania poprzez grupowanie węzłów grafu ze względu na ich stopień, czyli liczbę krawędzi incydentnych, stopień ich sąsiadów itp. Zabiegi te (pod warunkiem, że nie mamy do czynienia z grafem regularnym) ograniczają liczbę badanych kombinacji węzłów. Jednak zysk jest godny uwagi głównie w przypadku grafów, których niezbyt duży procent wierzchołków ma ten sam stopień.

Ponieważ proste porównywanie macierzy przyległości byłoby bardzo mało skuteczne, a przebadanie permutacji węzłów zbyt czasochłonne, w proponowanym algorytmie zastosowano inne rozwiązanie. Wykorzystano macierz przyległości  $A$  oraz strategię dekompozycji problemu zachowującą pierwotny porządek etykiet. Ponieważ w wyniku stosowania faktoryzacji i redukcji pewne wierzchołki występujące w grafie wejściowym zostają usunięte, etykiety wierzchołków przetwarzanego grafu nie zawsze tworzą ciągły zbiór liczb 1, 2, ...,  $|V|$ . Aby wygenerować część kodu grafu, która odpowiada za wykrywanie izomorfizmu na podstawie macierzy przyległości  $A$ , konieczne jest przypisanie istniejącym wierzchołkom nowych etykiet tworzących ciągły zbiór liczb 1, 2, ...,  $n$ , gdzie  $n$  jest ilością wierzchołków przetwarzanego grafu. Ponieważ zastosowano strategię zachowania pierwotnego porządku etykiet, zmieniają się jedynie etykiety większe od najmniejszej nie używanej etykiety grafu. Strategia dekompozycji problemu jest tak dobrana, by zmiany etykiet były jak najrzadsze, a faktoryzacja następowała na krawędziach incydentnych do wierzchołków o maksymalnej wartości etykiet. Dzięki temu wygenerowany kod możliwie często spełnia własność 7, co daje stosunkowo wysoki współczynnik trafienia (*hit rate*) w pamięci *cache*. W konsekwencji stosunkowo rzadko zdarza się wielokrotne rozwiązywanie



tego samego podproblemu. Współczynnik trafienia określany jest jako stosunek liczby rozwiązań podproblemów odczytanych z pamięci *cache* do liczby wszystkich rozwiązanych podproblemów. Efektywność zaproponowanej w algorytmie *Fact&Cache* techniki częściowego spamiętywania jest tym większa, im większej ilości łączny przypisano tę samą niezawodność. Efektywność algorytmu może być mierzona wartością współczynnika trafienia i średnim rozmiarem podproblemów odczytanych z pamięci *cache*. Tak rozumianą efektywność, zastosowanej w algorytmie *Fact&Cache* techniki częściowego spamiętywania, przebadano w rozdziale 7.6.

### 6.3. STRATEGIA DEKOMPOZYCJI PROBLEMU

Przeprowadzone przez autora eksperymenty świadczą o tym, że strategia dekompozycji problemu, czyli wybierania krawędzi grafu do faktoryzacji, ma wpływ na efektywność proponowanego algorytmu w jeszcze większym stopniu, niż ma to miejsce w przypadku standardowego algorytmu faktoryzacji, wykorzystującego zachowujące niezawodność redukcje.

Idea zaproponowanej strategii dekompozycji jest następująca: powinna ona sprzyjać zachowaniu pierwotnego porządku etykiet, tak, by zmiany etykiet były jak najrzadsze. Zaproponowana strategia dekompozycji wykorzystuje priorytety przyporządkowywane krawędziom grafu. Priorytety są przyporządkowywane zarówno w fazie wstępnej (gdy dokonywana jest analiza grafu), jak i w fazie zasadniczej (gdy powstają nowe krawędzie w wyniku stosowania redukcji równoległej, szeregowej i redukcji stopnia 2). Faktoryzacji podlega zawsze ta krawędź grafu, która ma najwyższy priorytet.

#### 6.3.1. Heurystyka przyporządkowywania priorytetów

Ponumerujmy wierzchołki grafu za pomocą etykiet  $v_1$  do  $v_n$  tak, aby etykiety wierzchołków przyległych różniły się jak najmniej, podczas gdy etykiety wierzchołków najbardziej od siebie oddalonych maksymalnie się różniły. Poprzez faktoryzację na krawędziach incydentnych do wierzchołków o najwyższych etykietach (maksymalnie oddalonych od  $v_1$ ) wpłyniemy w dużym stopniu na zachowanie pierwotnego porządku etykiet wierzchołków. Wygenerowane kody będą stosunkowo często spełniały własność 7, przez co uzyskamy wysoki współczynnik trafienia.



W fazie wstępnej dokonywana jest analiza grafu, w wyniku której wszystkim krawędziom grafu zostają przyporządkowane priorytety. Zaproponowana heurystyczna strategia przyporządkowania priorytetów realizowana jest w następujących krokach:

1. Znajdowane są minimalne odległości między wszystkimi wierzchołkami grafu, tzn. minimalne ilości krawędzi, które trzeba pokonać, aby dostać się z jednego wierzchołka do drugiego (do tego celu wykorzystany został algorytm wyszukiwania najkrótszych ścieżek zaproponowany przez Floyda).
2. Spośród wierzchołków grafu ( $|V| = n$ ) wyznaczamy parę wierzchołków  $(v_1, v_n)$ , które są najbardziej od siebie oddalone. Jeśli jest tylko jedna taka para, to wierzchołki  $v_1$  i  $v_n$  są natychmiast określone. Jeżeli tych par jest więcej to wybierana jest ta para, której składniki  $v_1$  i  $v_n$  pojawiają się najczęściej.
3. Mając wybrane wierzchołki  $v_1$  i  $v_n$ , dla wszystkich wierzchołków grafu wyliczane są następująco zdefiniowane współczynniki położenia:

$$distFactor[k] = \frac{\text{minimalna odległość pomiędzy wierzchołkami } v_1 \text{ i } k}{\text{minimalna odległość pomiędzy wierzchołkami } k \text{ i } v_n} \quad \text{dla } k \neq v_1 \text{ i } k \neq v_n$$

$$distFactor[v_1] = 0$$

$$distFactor[v_n] = maxFactor,$$

gdzie  $maxFactor$  jest odpowiednio dużą stałą, w praktyce musi być ona znacząco większa od wszystkich innych współczynników położenia.

4. Następnie przeprowadzana jest korekcja współczynników położenia o wpływ wierzchołków przyległych.

$P_k$  – zbiór wierzchołków przyległych do wierzchołka  $k$

$$distFactor[k] = distFactor[k] + \frac{\sum_{i \in P_k} distFactor[i]}{|P_k|}.$$

5. Wierzchołki grafu są etykietowane numerami od 1 do  $|V|$ . Im większa wartość współczynnika położenia, tym większy numer wierzchołka. Następnie krawędziom grafu przyporządkowywane są priorytety w zależności od wartości współczynników położenia wierzchołków incydentnych.

Pozostaje jeszcze problem przyporządkowania priorytetów krawędziom powstałym w wyniku redukcji równoległej i redukcji wierzchołków stopnia drugiego. Redukcje te mogą dotyczyć krawędzi o niskich priorytetach, zmniejszając efektywność wykrywania

równoważnych podproblemów. Dlatego nie zawsze ich użycie musi być opłacalne. W przypadku użycia redukcji uzasadnione wydaje się takie postępowanie, że krawędź, która powstaje w miejsce dwóch redukowanych krawędzi otrzymuje priorytet przynależny wcześniej krawędzi o wyższym priorytecie.

Zaproponowana strategia dekompozycji (wybierania krawędzi do faktoryzacji) sprzyja występowaniu tych samych podproblemów w binarnym drzewie obliczeń, co zostało eksperymentalnie potwierdzone w rozdziale 7.

#### 6.4. PODSUMOWANIE ROZDZIAŁU

W rozdziale tym zaproponowano algorytm *Fact&Cache*. Nowy algorytm wykorzystuje fakt, iż w trakcie obliczeń te same podproblemy (sieci powstałe w wyniku stosowania faktoryzacji i redukcji) są często rozwiązywane wiele razy. Zamiast wielokrotnie rozwiązywać ten sam podproblem (obliczać niezawodność tej samej sieci), jak ma to miejsce w przypadku wszystkich innych algorytmów faktoryzacji, nowy algorytm wykorzystuje wcześniej obliczony i zapamiętany rezultat obliczeń, stosując zmodyfikowaną technikę programowania dynamicznego. Efektywność zaproponowanego algorytmu *Fact&Cache* porównano w rozdziale 7 z innymi, aktualnie najefektywniejszymi algorytmami obliczania niezawodności  $K$ -terminali.

## 7. PORÓWNANIE EFEKTYWNOŚCI RÓŻNYCH ALGORYTMÓW OBLICZANIA NIEZAWODNOŚCI K-TERMINALI

O ile efektywność niektórych algorytmów obliczania niezawodności  $K$ -terminali można porównać analitycznie (rozdział 4 i 5), o tyle w przypadku wielu algorytmów złożoność obliczeniowa nie jest określana. Powszechnie przyjęte jest badanie i porównywanie efektywności algorytmów na drodze eksperymentalnej [ResendeM86, ResendeL88, Page88, Carlier91, Theologou91, Torrieri94, Carlier96]. Realizuje się to za pomocą typowych, zwykle wykorzystywanych w literaturze, sieci testowych (załącznik 1). Podobnie jak w przypadku większości sieci testowych wykorzystywanych w literaturze również w tej pracy przyjęto wartości prawdopodobieństwa działania łączy równe 0,9. W niniejszym rozdziale dokonano porównania efektywności różnych algorytmów obliczania niezawodności  $K$ -terminali wykorzystując zarówno rezultaty uzyskane na drodze analitycznej, jak i eksperymentalnej.

### 7.1. ZŁOŻONOŚĆ OBLICZENIOWA ALGORYTMU *FACT*

Zaimplementowany algorytm faktoryzacji *Fact* można opisać następująco ( $\mathbf{F}_0$ ,  $\{R_0, R_1, R_2, R_3\}$ ,  $\mathbf{S}''$ ). Pesymistyczną złożoność czasową tego algorytmu oszacowano wykorzystując grafy pełne w rozdziale 5 (twierdzenie 12 i 13). Również złożoność pamięciową tego algorytmu oszacowano w rozdziale 5 (twierdzenie 20 i 21). Stąd wiadomo, że w przypadku algorytmu faktoryzacji *Fact* ( $\mathbf{F}_0$ ,  $\{R_0, R_1, R_2, R_3\}$ ,  $\mathbf{S}''$ ), przy rozwiązywaniu problemu niezawodności  $K$ -terminali, pesymistyczna złożoność czasowa wynosi  $O(C(n, b)((n-2)!))$ , zaś pesymistyczna złożoność pamięciowa wynosi  $O\left(\frac{(n-2)(n-3)}{2}|G_K|\right)$ .

Powyższe rezultaty uzyskane analitycznie zostały potwierdzone przez testy empiryczne (rozdział 7.3.2).

### 7.2. ZŁOŻONOŚĆ OBLICZENIOWA ALGORYTMU *FACT&CACHE*

W przypadku algorytmu faktoryzacji *Fact&Cache* analiza złożoności czasowej jest utrudniona. Dopiero spełnienie wszystkich poniższych warunków zapewnia, że dowolny węzeł binarnego drzewa obliczeń nie będzie rozwijany:

1. wcześniej zarejestrowano podproblem (i jego rozwiązanie) równoważny aktualnie rozpatrywanemu podproblemowi;
2. powyższy podproblem i jego rozwiązanie są pamiętane (nie został usunięty z pamięci);
3. wykryto równoważność grafów reprezentujących badany i zapamiętany wcześniej podproblem.

Jak widać trudność analizy złożoności czasowej algorytmu *Fact&Cache* wynika między innymi z faktu, iż zastosowana technika generowania kodów podproblemów zapewniając efektywność kodowania, nie gwarantuje wykrycia równoważności podproblemów (rozdział 4.2). Ponadto zastosowana technika ograniczania ilości pamiętanych rozwiązań ograniczając złożoność pamięciową algorytmu sprawia, że możliwa jest sytuacja w której ten sam podproblem musi być po raz kolejny rozwiązywany, gdy zapamiętany wcześniej równoważny podproblem został usunięty z pamięci. Z powyższych względów liczba liści BDO algorytmu *Fact&Cache*  $L(G_K)$  jest trudna do oszacowania.

Chcąc oszacować złożoność czasową należy uwzględnić, że w każdym węźle binarnego drzewa obliczeń algorytmu *Fact&Cache* wykonywane są operacje typowe dla większości znanych algorytmów faktoryzacji (np. redukcje grafu czy selekcja krawędzi do faktoryzacji), których złożoność czasową oznaczyliśmy przez  $C(n, b)$  oraz operacje związane z wykrywaniem równoważnych podproblemów, czyli generowanie kodu podproblemu o złożoności czasowej  $C_c(n, b)$  i poszukiwanie równoważnego podproblemu o złożoności czasowej  $C_s(n, b)$ . Stąd złożoność czasową algorytmu *Fact&Cache* można oszacować przez  $O((C(n, b) + C_c(n, b) + C_s(n, b))L(G_K))$ . Złożoność czasowa operacji związanych z wykrywaniem równoważnych podproblemów<sup>66</sup> nie jest większa niż złożoność czasowa zastosowanych redukcji grafu i selekcji krawędzi do faktoryzacji (typowych dla znanych dotąd algorytmów faktoryzacji). W związku z tym złożoność czasową algorytmu *Fact&Cache* można oszacować przez  $O(C(n, b)L(G_K))$ .

---

<sup>66</sup> Przykładowo pesymistyczną złożoność czasową operacji związanych z generowaniem kodu podproblemu można oszacować poprzez  $O(b^2)$ . Poszukiwanie równoważnego podproblemu za pomocą algorytmu wykorzystującego tablice z haszowaniem wymaga w najgorszym przypadku  $O(c)$  operacji, gdzie  $c$  jest arbitralnie ustaloną maksymalną liczbą pamiętanych podproblemów.

Liczba liści BDO algorytmu faktoryzacji *Fact&Cache* będzie mniejsza lub w najgorszym przypadku równa liczbie liści BDO algorytmu faktoryzacji wykorzystującego tę samą strategię selekcji krawędzi do faktoryzacji, lecz nie wykorzystującego techniki wykrywania równoważnych podproblemów. To samo dotyczy złożoności czasowej.

W praktyce efektywność algorytmu *Fact&Cache* wynika stąd, iż całe poddrzewa BDO nie są rozwijane. Jest ona tym większa im większa część łączy charakteryzuje się tą samą niezawodnością<sup>67</sup>. W tym sensie wypełnia on lukę pomiędzy standardowym algorytmem faktoryzacji, który ma równie szeroki zakres zastosowań, jest jednak często zbyt wolny, a efektywnymi algorytmami obliczania niezawodności sieci, które charakteryzują się bardzo wąskim zakresem zastosowań (np. metody zaproponowane przez Gilberta [Gilbert59], które można zastosować tylko wtedy, gdy niezawodności wszystkich łączy są takie same, sieci reprezentowane są poprzez grafy pełne, a rozważany jest problem niezawodności wszystkich terminali albo problem niezawodności dwóch terminali).

Na złożoność pamięciową algorytmu *Fact&Cache* będą miały wpływ dwa czynniki. Pierwszy z nich związany jest z rekurencyjnym charakterem algorytmu. Zgrubnym pesymistycznym oszacowaniem tego czynnika jest  $O(b|G_K|)$ , gdzie  $b$  to liczba krawędzi grafu reprezentującego badaną sieć. Drugi czynnik związany jest ze stosowaną w algorytmie *Fact&Cache* techniką częściowego spamiętywania. Jeżeli przez  $c$  oznaczmy maksymalną liczbę pamiętanych podproblemów, a przez  $|K|$  rozmiar kodu podproblemu wraz z rozwiązaniem, to pesymistyczna złożoność pamięciowa algorytmu *Fact&Cache* wynosi  $O(b|G_K| + c|K|)$ .

### 7.3. PORÓWNANIE MIAR PESYMISTYCZNEJ ZŁOŻONOŚCI CZASOWEJ ALGORYTMÓW OBLICZANIA NIEZAWODNOŚCI K-TERMINALI

Przyjmijmy następujące oznaczenia:

$L_a(G_K)$  – oznacza określoną analitycznie liczbę liści BDO,

---

<sup>67</sup> Przyjęcie tych samych niezawodności niektórych łączy jest o tyle usprawiedliwione, że nie zawsze posiadamy dokładne dane o niezawodności łączy, w związku z czym często operujemy tymi samymi wartościami w obrębie określonych grup łączy, charakteryzujących się podobnymi wartościami innych parametrów (np. długość łączy, zastosowana technologia itp.).

$L_e(G_K)$  – oznacza określoną eksperymentalnie liczbę liści BDO.

Porównanie przeprowadzimy na sieciach reprezentowanych poprzez graf pełny.

### 7.3.1. Porównanie określonych analitycznie miar pesymistycznej złożoności czasowej różnych algorytmów faktoryzacji

W rozdziale tym przedstawimy porównanie określonej analitycznie maksymalnej liczby liści binarnego drzewa obliczeń  $L_a(G_K)$ , będącej miarą pesymistycznej złożoności czasowej różnych odmian algorytmu faktoryzacji. Porównanie przeprowadzono na sieciach reprezentowanych poprzez graf pełny  $K_n$ , gdzie  $n$  jest ilością wierzchołków.

Poniższa tabela ilustruje dotychczasowy postęp poczyniony na drodze do uzyskania możliwie najefektywniejszego algorytmu. Analizę maksymalnej liczby liści BDO poniższych algorytmów przeprowadzono z wykorzystaniem niezmienników grafu<sup>68</sup>.

| Algorytm                           | Zakres analizy<br>$n =  V $ ,<br>$b =  E $     | Liczba liści BDO dla<br>sieci reprezentowanych<br>przez graf pełny $K_n$ | Przykładowe liczby liści BDO dla<br>sieci reprezentowanych przez graf<br>pełny $K_n$ |                                           |
|------------------------------------|------------------------------------------------|--------------------------------------------------------------------------|--------------------------------------------------------------------------------------|-------------------------------------------|
|                                    |                                                |                                                                          | $K_6$<br>$n=6, b=15$                                                                 | $K_{10}$<br>$n=10, b=45$                  |
| badający całą<br>przestrzeń stanów | $2 \leq  K  \leq n$                            | $L_a(K_n) = 2^b = 2^{\binom{n}{2}}$                                      | 32768                                                                                | 3,51843721e+13                            |
| $(F_0, \{R5, R6\}, S_1)$           | $ K  = n$                                      | $L_a(K_n) = n^{n-2}$                                                     | 1296                                                                                 | 100000000                                 |
| $(F_0, \{R1, R2, R5\}, S_2)$       | $2 \leq  K  \leq n$                            | $L_a(K_n) = (n-1)!$                                                      | 120                                                                                  | 362880                                    |
| $(F_0, \{R1, R2, R3, R4\}, S_4)$   | $2 \leq  K  \leq 5$ ,<br>$n-2 \leq  K  \leq n$ | $L_a(K_n) = (n-2)!$                                                      | 24                                                                                   | 40320* (dla<br>przypadków<br>granicznych) |

**Tabela 2. Porównanie określonej analitycznie (z wykorzystaniem niezmienników) maksymalnej liczby liści binarnego drzewa obliczeń różnych algorytmów faktoryzacji.**

Kolejna tabela zawiera porównanie określonej analitycznie maksymalnej liczby liści binarnego drzewa obliczeń innych (rozważanych przez autora) modyfikacji algorytmu faktoryzacji. Analizę maksymalnej liczby liści BDO przedstawionych poniżej algorytmów

<sup>68</sup> Por. rozdział 4, gdzie są opisane strategie  $S_1$ ,  $S_2$ ,  $S_4$  oraz [Wood86].

przeprowadzono z wykorzystaniem zaproponowanej przez autora metody wykorzystania grafów pełnych<sup>69</sup>.

| Algorytm                                     | Zakres analizy<br>$n =  V $ ,<br>$b =  E $ | Liczba liści BDO dla sieci reprezentowanych przez graf pełny $K_n$ | Przykładowe liczby liści BDO dla sieci reprezentowanych przez graf pełny $K_n$ |                             |
|----------------------------------------------|--------------------------------------------|--------------------------------------------------------------------|--------------------------------------------------------------------------------|-----------------------------|
|                                              |                                            |                                                                    | $K_6$<br>$n=6, b=15$                                                           | $K_{10}$<br>$n=10, b=45$    |
| $(F_0, \{R0, R1\}, S'')$                     | $2 \leq  K  \leq n$                        | $L_a(K_n) = (n-1)!$                                                | 120                                                                            | 362880                      |
| $(F_0, \{R1\}, S''')$                        | $ K  = 2$                                  | $L_a(K_n) = 2e \left( (n-2)! \right)$                              | 130*<br>(dla $ K  = 2$ )                                                       | 219202*<br>(dla $ K  = 2$ ) |
| $(F_0, \{R1, R2, R3\}, S'')$                 | $2 \leq  K  \leq n$                        | $L_a(K_n) = 2 \left( (n-2)! \right)$                               | 48                                                                             | 80640                       |
| <i>Fact</i> $(F_0, \{R0, R1, R2, R3\}, S'')$ | $2 \leq  K  \leq n$                        | $L_a(K_n) = (n-2)!$                                                | 24                                                                             | 40320                       |

**Tabela 3. Porównanie określonej analitycznie (z wykorzystaniem grafów pełnych) maksymalnej liczby liści binarnego drzewa obliczeń różnych algorytmów faktoryzacji.**

Algorytmy faktoryzacji  $(F_0, \{R0, R1\}, S'')$ ,  $(F_0, \{R1\}, S''')$ ,  $(F_0, \{R1, R2, R3\}, S'')$  i  $(F_0, \{R0, R1, R2, R3\}, S'')$  zostały zaimplementowane w języku C++. Otrzymane eksperymentalnie rezultaty potwierdziły, przedstawione w powyższej tabeli, określone analitycznie maksymalne liczby liści BDO algorytmów faktoryzacji.

Szczególnie istotne jest to, że w przypadku (zaproponowanego przez autora) algorytmu *Fact*  $(F_0, \{R0, R1, R2, R3\}, S'')$  można zagwarantować<sup>70</sup>, iż liczba liści BDO  $L(G_K) = (n-2)!$  będąca miarą pesymistycznej złożoności czasowej będzie osiągalna dla dowolnego  $K$  ( $2 \leq |K| \leq |V|$ ) w odróżnieniu od innych algorytmów, w tym również od analizowanego przez Wooda algorytmu  $(F_0, \{R1, R2, R3, R4\}, S_4)$  charakteryzującego się jak dotąd najmniejszą pesymistyczną złożonością czasową.

Powyższy rezultat uzyskano przy prostszym zbiorze zachowujących niezawodność redukcji grafu (nie ma bowiem konieczności uwzględniania najbardziej skomplikowanego podzbioru redukcji wielokąta do łańcucha) i nieskomplikowanej strategii selekcji krawędzi do faktoryzacji.

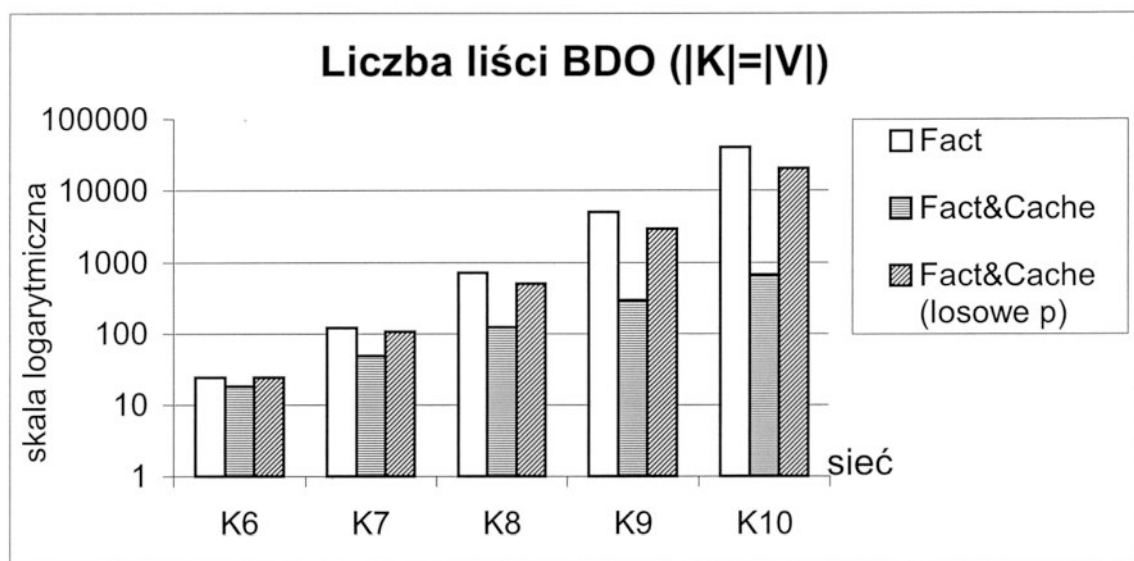
<sup>69</sup> Por. rozdział 5, gdzie są opisane strategie  $S''$  i  $S'''$ .

<sup>70</sup> Por. rozdział 5.



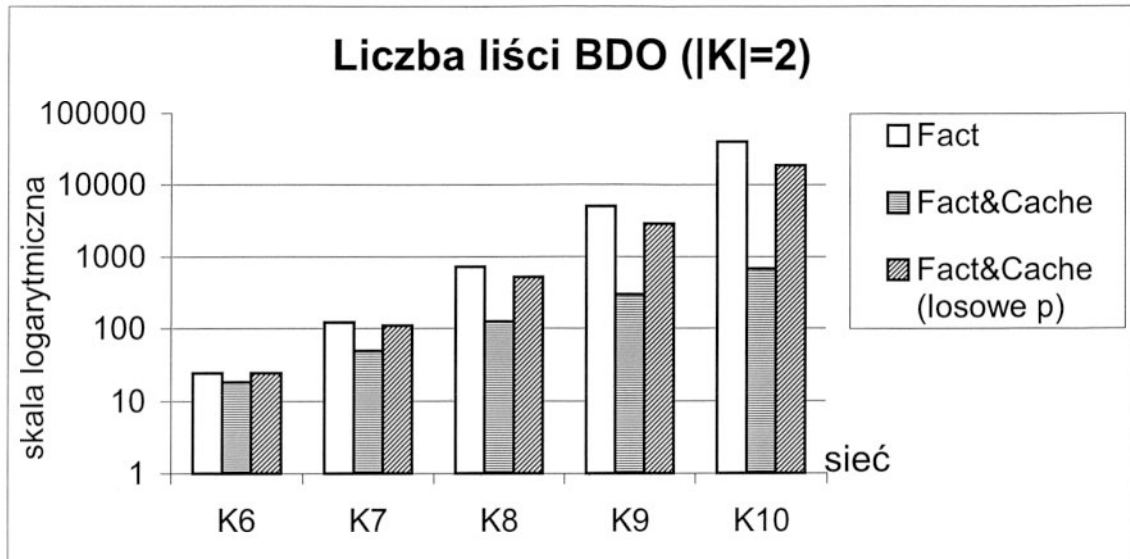
### 7.3.2. Porównanie otrzymanych eksperymentalnie miar pesymistycznej złożoności czasowej algorytmu *Fact* i *Fact&Cache*

W rozdziale tym przeprowadzone zostanie porównanie otrzymanych eksperymentalnie miar złożoności czasowej algorytmu *Fact* i *Fact&Cache* dla sieci reprezentowanych poprzez graf pełny. W przypadku algorytmu *Fact&Cache* pesymistyczną złożoność obliczeniową można oszacować określając maksymalną liczbę liści BDO uzyskaną na odpowiednio licznej próbie grafów pełnych o losowych prawdopodobieństwach krawędzi. Rozpatrzone zostaną dwa najczęściej rozważane przypadki niezawodności wszystkich terminali (rysunek 7.) i niezawodności dwóch terminali (rysunek 8.).



Rysunek 7. Porównanie określonej eksperymentalnie liczby liści binarnego drzewa obliczeń algorytmów *Fact* i *Fact&Cache* w przypadku obliczania niezawodności wszystkich terminali ( $|K|=|V|$ ) sieci reprezentowanych poprzez graf pełny.





**Rysunek 8. Porównanie określonej eksperymentalnie liczby liści binarnego drzewa obliczeń algorytmów *Fact* i *Fact&Cache* w przypadku obliczania niezawodności dwóch terminali ( $|K|=2$ ) sieci reprezentowanych poprzez graf pełny.**

Liczba liści binarnego drzewa obliczeń algorytmu *Fact&Cache* jest we wszystkich rozpatrywanych przypadkach mniejsza lub równa liczbie liści BDO algorytmu *Fact*. Pozwala to wysunąć hipotezę o mniejszej pesymistycznej złożoności czasowej algorytmu *Fact&Cache*. Różnica w liczbie liści BDO algorytmów *Fact* i *Fact&Cache* jest tym większa, im większa jest rozpatrywana sieć, co dodatkowo zwiększa atrakcyjność algorytmu *Fact&Cache*.

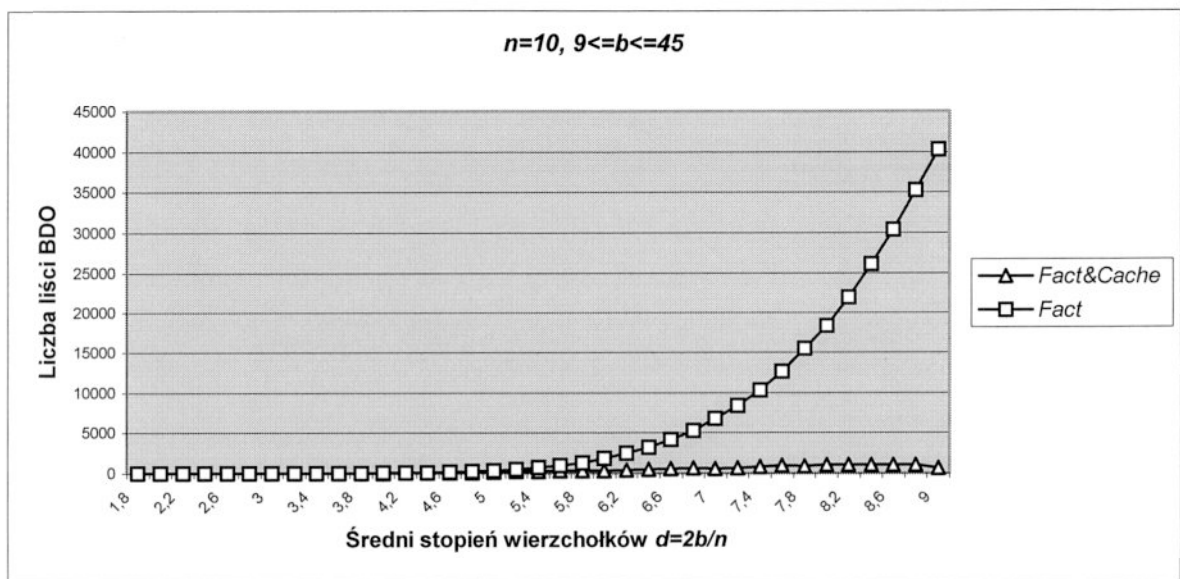
#### 7.4. PORÓWNANIE EFEKTYWNOŚCI ALGORYTMÓW FAKTORYZACJI NA PRZYKŁADZIE GRAFÓW LOSOWYCH Z UWZGLĘDNIENIEM GĘSTOŚCI GRAFÓW

Zgodnie z sugestią pani Balińskiej [Balińska98] do badania efektywności algorytmów faktoryzacji *Fact* i *Fact&Cache* wykorzystano grafy losowe. Efektywność algorytmów faktoryzacji bada się zwykle określając liczbę liści BDO. Liczba liści BDO zależy nie tylko od rozmiaru, ale również od stopnia trudności rozpatrywanych problemów (sieci reprezentowanych za pomocą grafów). Na ów stopień trudności wpływa w istotny sposób gęstość rozpatrywanych grafów, którą wyraża średni stopień wierzchołków  $d = \frac{2b}{n}$ .

Zachowujące niezawodność redukcje grafu umożliwiają bowiem efektywną (w czasie wielomianowym) redukcję jedynie wierzchołków niskiego stopnia (stopnia 1. i w dużej

mierze stopnia 2.). W przypadku grafów, których średni stopień wierzchołków jest niski, obliczenie miar niezawodności będzie wymagać stosunkowo niewielkich nakładów czasowych ze względu na niewielką liczbę faktoryzacji.

Poniżej porównano liczbę liści BDO algorytmów faktoryzacji wykorzystując grafy losowe  $G(n,b)$  o określonej liczbie wierzchołków  $n$ , lecz o zmiennej liczbie krawędzi  $b$  (a co za tym idzie również o zmiennej gęstości) grafów. Badania przeprowadzono generując po 1000<sup>71</sup> grafów losowych  $G(n,b)$  o liczbie wierzchołków  $n = 10$  i liczbie krawędzi  $9 \leq b \leq 45$  (gęstości  $1,8 \leq d \leq 9$ ).



**Rysunek 9.** Porównanie średniej liczby liści BDO algorytmów *Fact* i *Fact&Cache* przy obliczaniu niezawodności wszystkich terminali ( $|K|=|V|$ ) dla grafów losowych  $G(n,b)$  o  $n=10$ ,  $9 \leq b \leq 45$ .

Przedstawione na rysunku 9. porównanie średniej liczby liści BDO algorytmów *Fact* i *Fact&Cache* przy obliczaniu niezawodności wszystkich terminali ( $|K|=|V|$ ) dla grafów losowych  $G(n,b)$  o  $n = 10$ ,  $9 \leq b \leq 45$  ilustruje wyższość zaproponowanego algorytmu *Fact&Cache*. Niewątpliwie jego mocną stroną jest większa w porównaniu z algorytmem *Fact* skuteczność, szczególnie w przypadku grafów o dużej gęstości i rozmiarze.

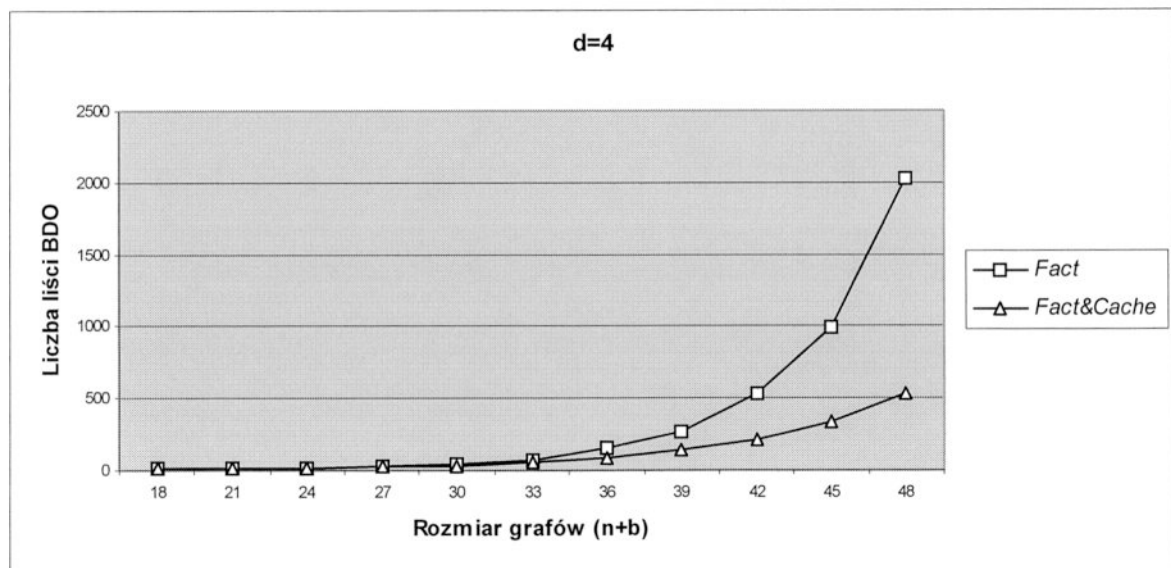
Na rysunku 9. wyraźnie zaznaczono, że wraz ze wzrostem rozmiaru badanych grafów wzrastała ich gęstość. Badając efektywność algorytmów faktoryzacji w zależności od rozmiaru rozpatrywanych grafów należy zwrócić szczególną uwagę na – umykające niekiedy uwadze – zmiany gęstości badanych grafów. Gęstość grafów ma bowiem istotny wpływ na efektywność algorytmów. Przykładowo taka niekorzystna sytuacja ma miejsce wówczas, gdy

<sup>71</sup> Za wyjątkiem przypadków brzegowych, np. gdy  $d=1,8$ , czy  $d=9$  (graf pełny).

rozważamy efektywność algorytmów faktoryzacji dla grafów  $G(n,p)$  (gdzie  $p$  jest prawdopodobieństwem wystąpienia krawędzi), w przypadku których wraz z rosnącym rozmiarem badanych grafów (sieci) zwiększa się ich gęstość:  $b = \frac{n(n-1)}{2} p$ ,

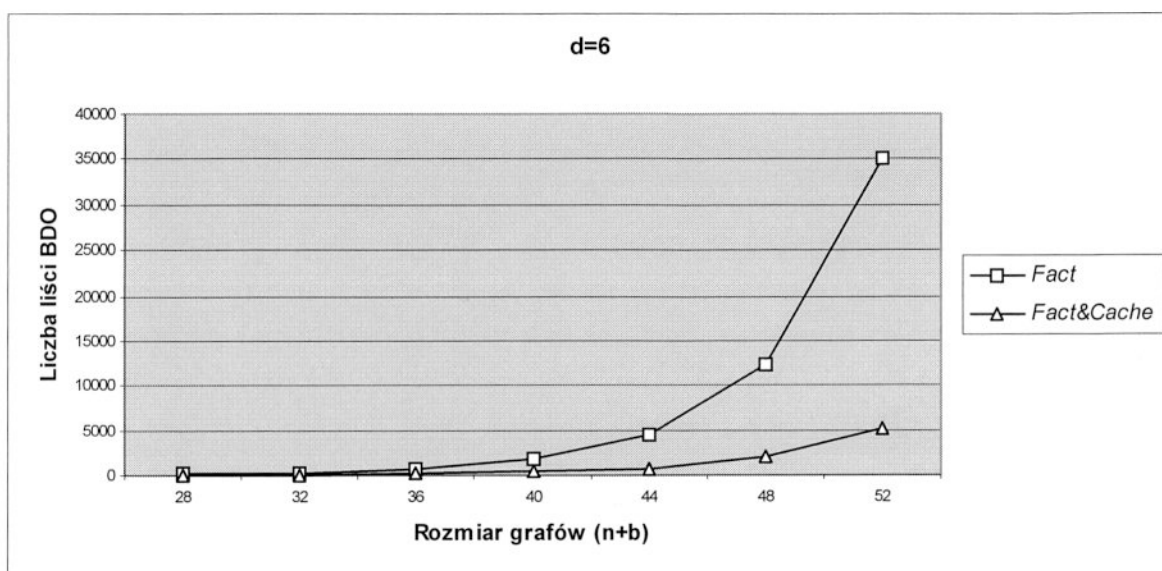
$$\text{stąd } d = \frac{2b}{n} = \frac{n(n-1)p}{n} = (n-1)p.$$

Z tego względu obliczono średnią liczbę liści BDO algorytmów faktoryzacji w zależności od rozmiaru rozpatrywanych grafów losowych  $G(n,b)$  przy określonych (stałych) wartościach gęstości grafów ( $d = 2, 4, 6$  i  $8$ )<sup>72</sup>. Niezawodność wszystkich terminali sieci o gęstości  $d = 2$  można obliczyć wykorzystując zachowujące niezawodność redukcje (faktoryzacja nie jest konieczna), a średnia liczba liści BDO jest równa jeden. W pozostałych przypadkach rozważamy sieci o wyższych gęstościach więc faktoryzacja jest niezbędna. Poniższe rysunki przedstawiają porównanie średniej liczby liści BDO algorytmów *Fact* i *Fact&Cache* przy obliczaniu niezawodności wszystkich terminali ( $|K|=|V|$ ) w zależności od rozmiaru rozpatrywanych grafów  $G(n,b)$  i gęstości grafów.

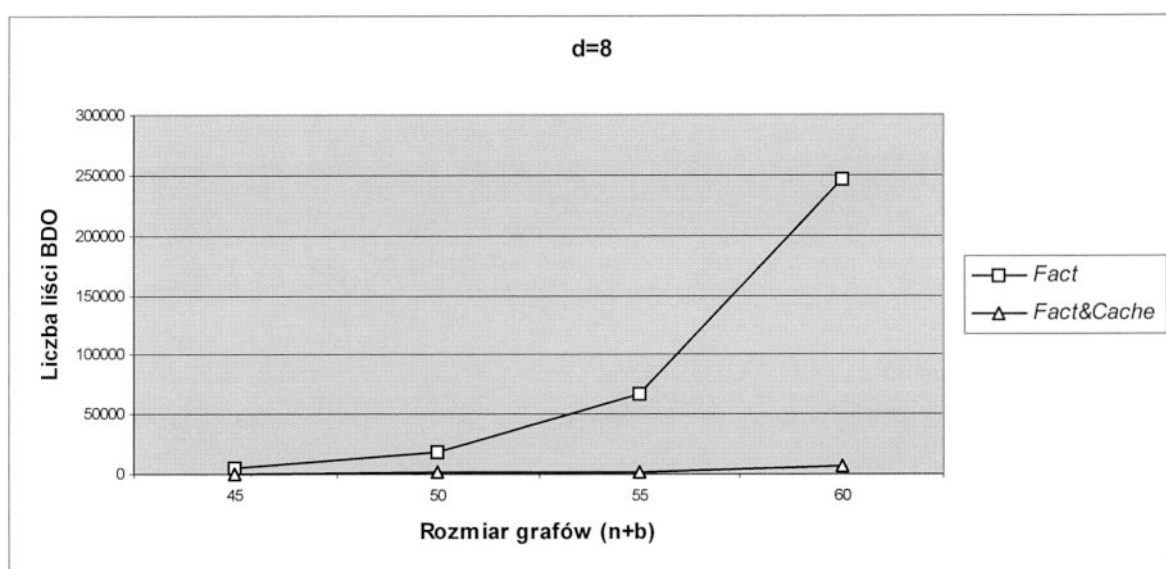


**Rysunek 10.** Porównanie średniej liczby liści BDO algorytmów *Fact* i *Fact&Cache* przy obliczaniu niezawodności wszystkich terminali ( $|K|=|V|$ ) w zależności od rozmiaru rozpatrywanych grafów  $G(n,b)$  przy stałej gęstości  $d=4$ .

<sup>72</sup> Gdy badamy grafy o stałej gęstości, to nie ma większego znaczenia czy przez rozmiar grafu rozumiemy liczbę wierzchołków, liczbę krawędzi, czy ich sumę.



Rysunek 11. Porównanie średniej liczby liści BDO algorytmów *Fact* i *Fact&Cache* przy obliczaniu niezawodności wszystkich terminali ( $|K|=|V|$ ) w zależności od rozmiaru rozpatrywanych grafów  $G(n,b)$  przy stałej gęstości  $d=6$ .



Rysunek 12. Porównanie średniej liczby liści BDO algorytmów *Fact* i *Fact&Cache* przy obliczaniu niezawodności wszystkich terminali ( $|K|=|V|$ ) w zależności od rozmiaru rozpatrywanych grafów  $G(n,b)$  przy stałej gęstości  $d=8$ .

Na rysunkach 10., 11. i 12. widzimy wpływ rozmiaru badanych grafów na efektywność algorytmów faktoryzacji<sup>73</sup>. W przypadku algorytmu *Fact&Cache* wpływ ten jest zauważalnie mniejszy niż w przypadku algorytmu *Fact*. Również zmiany gęstości badanych grafów w różny sposób oddziałują na oba badane algorytmy. Ze wzrostem gęstości

<sup>73</sup> Wpływ ten nie jest zakłócony przez zmiany gęstości grafów.

rozpatrywanych grafów uwidacznia się znaczna dysproporcja pomiędzy liczbą liści BDO algorytmów *Fact* i *Fact&Cache*.

## 7.5. EFEKTYWNOŚĆ ALGORYTMU *FACT&CACHE* NA TLE AKTUALNIE NAJEFEKTYWNIJSZYCH ALGORYTMÓW OBLICZANIA NIEZAWODNOŚCI K-TERMINALI

Ze względu na znaczną ilość zaproponowanych metod obliczania miar niezawodności sieci w przeprowadzonym w tym rozdziale porównaniu ujęto jedynie metody o najwyższej efektywności.

W 1991 roku ukazały się dwa artykuły [Carlier91, Theologou91], których autorzy Carlier i Theologou z Compiègne University of Technology dokonali porównania – zaproponowanych do końca lat osiemdziesiątych – algorytmów obliczania niezawodności sieci. Przedstawiony przez nich algorytm faktoryzacji RES, wykorzystujący redukcje proste (R1, R2, R3), redukcje wielokąta do łańcucha i dwuspójną dekompozycję, okazał się najefektywniejszym i zdaniem autorów „umożliwiał obliczanie niezawodności sieci o większych, niż to wcześniej było możliwe, rozmiarach”<sup>74</sup>. Można więc przyjąć, iż algorytm ten w znacznym stopniu reprezentuje szczytowe osiągnięcia w dziedzinie analizy niezawodności sieci przełomu lat osiemdziesiątych i dziewięćdziesiątych.

W latach dziewięćdziesiątych okazało się, że w dziedzinie analizy niezawodności sieci algorytmy dekompozycji [Carlier96, Beichelt91a, Beichelt91b] należą do najefektywniejszych.

Ostatecznie w porównaniu zostały uwzględnione następujące algorytmy i programy służące do analizy niezawodności sieci:

- RES91 – odmiana algorytmu faktoryzacji, której autorami są Carlier i Theologou [Carlier91, Theologou91] (Uwaga: Autorzy wykazują pewną niekonsekwencję podając, w przypadku niektórych sieci, w różnych artykułach [Carlier91, Theologou91, Carlier96] różne czasy wykonania. W związku z tym podzielono dane w zależności od źródła pochodzenia np. C91 z [Carlier91], T91 z [Theologou91], C96 z [Carlier96]);
- CL96 – algorytm dekompozycji, którego autorami są Carlier i Lucet [Carlier96];

---

<sup>74</sup> Por. [Carlier91], s.3.

- BT91 – algorytm będący połączeniem algorytmu dekompozycji z algorytmem faktoryzacji umożliwiające badanie niezawodności wszystkich terminali, którego autorami są Beichelt i Tittmann [Beichelt91a];
- NRA97 (Network Reliability Analysis (C)1997, HTW Mittweida) – pakiet do analizy niezawodnościowej sieci, pracujący w środowisku Windows 3.1/Windows95 stworzony na Uniwersytecie Mittweida<sup>75</sup>;
- *Fact* jest zaproponowanym przez autora zmodyfikowanym algorytmem Page’a i Perry [Page88], który został omówiony w rozdziale 5;
- *Fact&Cache* jest zaproponowanym przez autora algorytmem, który został omówiony w rozdziale 6.

### 7.5.1. Porównanie czasów obliczeń

Głównym celem twórców nowych metod analizy niezawodności sieci jest stworzenie możliwie najefektywniejszego algorytmu. Ze względu na fakt, że efektywność większości prezentowanych algorytmów jest określana jedynie eksperymentalnie, na pewnych typowo rozpatrywanych w literaturze sieciach testowych, w rozdziale tym dokonano porównania uzyskanych eksperymentalnie czasów obliczeń. Zestawienie wszystkich badanych sieci znajduje się w załączniku 1.

| sieć | <i>n</i> | <i>b</i> | czas obliczeń [s] |                       |              |              |       |
|------|----------|----------|-------------------|-----------------------|--------------|--------------|-------|
|      |          |          | PC Pentium        |                       |              | VAX 8530     |       |
|      |          |          | <i>Fact</i>       | <i>Fact&amp;Cache</i> | <i>NRA97</i> | <i>RES91</i> |       |
|      |          |          |                   |                       |              | C91          | T91   |
| 3x6  | 18       | 27       | 0,2               | <0,1                  | 0,3          |              | 0,97  |
| 3x7  | 21       | 32       | 0,8               | <0,1                  | 1,3          |              | 3,91  |
| 3x8  | 24       | 37       | 3,7               | <0,1                  | 5,9          |              | 15,42 |

c.d. tabeli na następnej stronie

<sup>75</sup> Prof. Tittmann, będąc współautorem algorytmu BT91, wchodzi również w skład grupy badawczej zajmującej się niezawodnością sieci, która stworzyła pakiet NRA97. Aktualnie bierze udział w realizacji projektów badawczych, dotyczących między innymi algorytmów obliczania miar niezawodności sieci komputerowych i komunikacyjnych realizowanym we współpracy z CINDATEC GmbH, a finansowanym przez rząd Saksonii, jak również projektowania i optymalizacji sieci, finansowanym przez GERMAN TELEKOM

| sieć   | n  | b  | czas obliczeń [s] |                       |              |              |        |
|--------|----|----|-------------------|-----------------------|--------------|--------------|--------|
|        |    |    | PC Pentium        |                       |              | VAX 8530     |        |
|        |    |    | <i>Fact</i>       | <i>Fact&amp;Cache</i> | <i>NRA97</i> | <i>RES91</i> |        |
|        |    |    |                   |                       |              | C91          | T91    |
| 3x9    | 27 | 42 | 17                | 0,1                   | 28           |              | 61,27  |
| 3x10   | 30 | 47 | 80                | 0,2                   | 133          |              | 240,39 |
| 3x11   | 33 | 52 | 370               | 0,2                   | 625          |              |        |
| 3x12   | 36 | 57 | 1712              | 0,2                   | 2941         |              |        |
| 4x4    | 16 | 24 | <0,1              | <0,1                  | 0,2          |              |        |
| 4x5    | 20 | 31 | 0,8               | 0,1                   | 1,3          |              |        |
| 4x6    | 24 | 38 | 8,5               | 0,3                   | 13           |              |        |
| 4x7    | 28 | 45 | 91                | 0,4                   | 146          |              |        |
| 4x8    | 32 | 52 | 989               | 0,7                   | 1661         |              |        |
| SODET1 | 24 | 38 | 5,2               | 0,4                   |              | 18,22        |        |
| SODET2 | 26 | 43 | 22                | 0,4                   |              | 64,12        |        |
| CUBE16 | 16 | 24 | 0,1               | <0,1                  |              | 0,6-0,2      |        |
| CUBE20 | 20 | 30 | 0,6               | 0,2                   |              |              | 4,23   |
| BAILEY | 17 | 25 | 0,2               | <0,1                  |              | 0,72         |        |
| FRA732 | 11 | 21 | <0,1              | <0,1                  |              |              |        |

**Tabela 4. Porównanie czasów wykonania w przypadku obliczania niezawodności dwóch terminali ( $|K|=2$ ).**

| sieć | n  | b  | czas obliczeń [s] |                       |              |             |                |              |     |
|------|----|----|-------------------|-----------------------|--------------|-------------|----------------|--------------|-----|
|      |    |    | PC Pentium        |                       |              | PC 486      | Robotron A7150 | VAX 8530     |     |
|      |    |    | <i>Fact</i>       | <i>Fact&amp;Cache</i> | <i>NRA97</i> | <i>CL96</i> | <i>BT91</i>    | <i>RES91</i> |     |
|      |    |    |                   |                       |              |             |                | C91          | C96 |
| 3x6  | 18 | 27 | <0,1              | <0,1                  | 0,1          |             |                |              |     |
| 3x7  | 21 | 32 | <0,1              | <0,1                  | 0,2          |             |                |              |     |
| 3x8  | 24 | 37 | 0,3               | <0,1                  | 0,4          |             |                |              |     |
| 3x9  | 27 | 42 | 0,7               | <0,1                  | 1,2          |             |                |              |     |

c.d. tabeli na następnej stronie



| sieć    | <i>n</i> | <i>b</i> | czas obliczeń [s] |                       |              |             |                   |              |            |
|---------|----------|----------|-------------------|-----------------------|--------------|-------------|-------------------|--------------|------------|
|         |          |          | PC Pentium        |                       |              | PC 486      | Robotron<br>A7150 | VAX 8530     |            |
|         |          |          | <i>Fact</i>       | <i>Fact&amp;Cache</i> | <i>NRA97</i> | <i>CL96</i> | <i>BT91</i>       | <i>RES91</i> |            |
|         |          |          |                   |                       |              |             |                   | <i>C91</i>   | <i>C96</i> |
| 3x10    | 30       | 47       | 2,3               | <0,1                  | 3,7          | 1,7         |                   |              | 26,6       |
| 3x11    | 33       | 52       | 7                 | <0,1                  | 11,4         |             |                   |              |            |
| 3x12    | 36       | 57       | 21                | 0,1                   | 36           |             |                   |              |            |
| 4x4     | 16       | 24       | <0,1              | <0,1                  | <0,1         |             |                   |              |            |
| 4x5     | 20       | 31       | 0,2               | <0,1                  | 0,3          |             |                   |              |            |
| 4x6     | 24       | 38       | 1,4               | 0,1                   | 2,1          |             |                   |              |            |
| 4x7     | 28       | 45       | 11                | 0,2                   | 17,5         | 4,5         |                   |              |            |
| 4x8     | 32       | 52       | 92                | 0,3                   | 146          |             |                   |              |            |
| Arpanet | 21       | 32       | 0,5               | <0,1                  |              | 1,82        |                   | 4            | 6,51       |
| SODET1  | 24       | 38       | 2                 | 0,2                   |              | 7,36        |                   |              | 40         |
| SODET2  | 26       | 43       | 9,7               | 0,3                   |              | 4,5         |                   |              | 108,7      |
| EDF2    | 30       | 52       | 153               | 0,9                   |              | 12,8        |                   | 1315         | 1315       |
| PLG     | 21       | 57       | 5147              | 2,3                   | 7570         |             | 120               |              |            |

**Tabela 5. Porównanie czasów wykonania w przypadku obliczania niezawodności wszystkich terminali ( $|K|=|V|$ ).**

Porównanie czasów obliczeń zarówno w przypadku niezawodności dwóch terminali (tabela 4), jak i niezawodności wszystkich terminali (tabela 5), wskazuje na bardzo wysoką efektywność algorytmu *Fact&Cache*. Różnica w stosunku do uznawanego dotąd za najefektywniejszy, a drugiego w powyższym rankingu, algorytmu dekompozycji *CL96*, jest znacząca w każdym rozpatrywanym przypadku.

Chociaż trudno przesądzać o efektywności algorytmu *BT91* na podstawie tylko jednego przykładu, to jednak wydaje się, że zaproponowany przez Beichelta i Tittmanna algorytm jest również godny uwagi. Znacznie mniej efektywne do analizy niezawodności sieci wydają się być algorytmy *Fact*, *RES91* i pakiet *NRA97*.



### 7.5.2. Porównanie liczby liści binarnego drzewa obliczeń algorytmów faktoryzacji

Ze względu na różnice szybkości komputerów bardzo interesujące jest porównanie liczby liści binarnego drzewa obliczeń różnych algorytmów faktoryzacji.

Porównanie liczby liści binarnego drzewa obliczeń algorytmów faktoryzacji *Fact*, *Fact&Cache* oraz RES91 w przypadku obliczania niezawodności dwóch terminali, jak również niezawodności wszystkich terminali dla typowo rozważanych w literaturze sieci, przedstawiono w poniższych tabelach.

| sieć   | <i>n</i> | <i>b</i> | liczba liści (węzłów) binarnego drzewa obliczeń |                       |                      |      |
|--------|----------|----------|-------------------------------------------------|-----------------------|----------------------|------|
|        |          |          | <i>Fact</i>                                     | <i>Fact&amp;Cache</i> | <i>RES91</i>         |      |
|        |          |          |                                                 |                       | CT91                 | TC91 |
| 3x6    | 18       | 27       | 614                                             | 88                    | 226                  |      |
| 3x7    | 21       | 32       | 2799                                            | 118                   | 873                  |      |
| 3x8    | 24       | 37       | 12766                                           | 150                   | 3741                 |      |
| 3x9    | 27       | 42       | 58231                                           | 184                   | 13592                |      |
| 3x10   | 30       | 47       | 265622                                          | 216                   | 53629                |      |
| 3x11   | 33       | 52       | 1211647                                         | 248                   |                      |      |
| 3x12   | 36       | 57       | 5526990                                         | 280                   |                      |      |
| 4x4    | 16       | 24       | 275                                             | 114                   |                      |      |
| 4x5    | 20       | 31       | 2760                                            | 288                   |                      |      |
| 4x6    | 24       | 38       | 28740                                           | 478                   |                      |      |
| 4x7    | 28       | 45       | 303380                                          | 668                   |                      |      |
| 4x8    | 32       | 52       | 3219542                                         | 858                   |                      |      |
| SODET1 | 24       | 38       | 17670                                           | 740                   | (9494) <sup>76</sup> |      |
| SODET2 | 26       | 43       | 73351                                           | 747                   | 20234(40467)         |      |

c.d. tabeli na następnej stronie

<sup>76</sup> Wielkość ta wydaje się być nieprawidłowa, gdyż liczba węzłów binarnego drzewa obliczeń powinna być nieparzysta. Najprawdopodobniej jest to wielkość zanizona, skoro po zastosowaniu w algorytmie RES techniki aproksymacyjnej, umożliwiającej zmniejszenie liczby odwiedzanych węzłów BDO, konieczne jest odwiedzenie większej (np. 9834) liczby węzłów BDO. Por [Carlier91], s.12.

| sieć   | $n$ | $b$ | liczba liści (węzłów) binarnego drzewa obliczeń |                       |              |           |
|--------|-----|-----|-------------------------------------------------|-----------------------|--------------|-----------|
|        |     |     | <i>Fact</i>                                     | <i>Fact&amp;Cache</i> | <i>RES91</i> |           |
|        |     |     |                                                 |                       | CT91         | TC91      |
| BAILEY | 17  | 25  | 532                                             | 138                   | 254(507)     | 225(451)  |
| CUBE20 | 20  | 30  | 2057                                            | 308                   | 856(1711)    | 990(1979) |

**Tabela 6. Porównanie liczby liści binarnego drzewa obliczeń w przypadku obliczania niezawodności dwóch terminali ( $|K|=2$ ).**

| sieć    | $n$ | $b$ | liczba liści (węzłów) binarnego drzewa obliczeń |                       |              |                |
|---------|-----|-----|-------------------------------------------------|-----------------------|--------------|----------------|
|         |     |     | <i>Fact</i>                                     | <i>Fact&amp;Cache</i> | <i>RES91</i> |                |
|         |     |     |                                                 |                       | CT91         |                |
| 3x6     | 18  | 27  | 81                                              | 35                    |              |                |
| 3x7     | 21  | 32  | 243                                             | 76                    |              |                |
| 3x8     | 24  | 37  | 729                                             | 99                    |              |                |
| 3x9     | 27  | 42  | 2187                                            | 74                    |              |                |
| 3x10    | 30  | 47  | 6561                                            | 87                    |              |                |
| 3x11    | 33  | 52  | 19683                                           | 100                   |              |                |
| 3x12    | 36  | 57  | 59049                                           | 113                   |              |                |
| 4x4     | 16  | 24  | 72                                              | 44                    |              |                |
| 4x5     | 20  | 31  | 569                                             | 129                   |              |                |
| 4x6     | 24  | 38  | 4491                                            | 236                   |              |                |
| 4x7     | 28  | 45  | 35442                                           | 346                   |              |                |
| 4x8     | 32  | 52  | 279697                                          | 454                   |              |                |
| Arpanet | 21  | 32  | 1686                                            | 132                   |              | 1600(3199)     |
| CUBE16  | 16  | 24  | 247                                             | 112                   |              | 247(493)       |
| EDF2    | 30  | 52  | 489920                                          | 1125                  |              | 464106(928211) |
| PLG     | 21  | 57  | 18197988                                        | 2850                  |              |                |

**Tabela 7. Porównanie liczby liści binarnego drzewa obliczeń w przypadku obliczania niezawodności wszystkich terminali ( $|K|=|V|$ ).**

Powyższe tabele przedstawiają porównanie liczby liści binarnego drzewa obliczeń trzech algorytmów faktoryzacji. Bardzo znaczna różnica w liczbie liści BDO, pomiędzy algorytmem *Fact&Cache* a pozostałymi algorytmami, świadczy o tym, że nowy algorytm

wyraźnie dystansuje pozostałe algorytmy faktoryzacji. Różnica jest szczególnie widoczna w przypadku rozbudowanych sieci, co dodatkowo zwiększa atrakcyjność zaproponowanego algorytmu.

## 7.6. WSPÓŁCZYNNIK TRAFIENIA ALGORYTMU *FACT&CACHE*

Efektywność zaimplementowanego w algorytmie *Fact&Cache* mechanizmu wykrywania równoważnych podproblemów może być mierzona wartością współczynnika trafienia i średnim rozmiarem podproblemów odczytanych z pamięci *cache* (co ilustrują poniższe tabele).

| Sieć   | $n$ | $b$ | współczynnik trafienia [%] | średni rozmiar podproblemu odczytanego z pamięci <i>cache</i> |      |
|--------|-----|-----|----------------------------|---------------------------------------------------------------|------|
|        |     |     |                            | $n$                                                           | $b$  |
| 3x6    | 18  | 27  | 40,8                       | 7,6                                                           | 9,6  |
| 3x7    | 21  | 32  | 43,5                       | 9,1                                                           | 12   |
| 3x8    | 24  | 37  | 45                         | 10,6                                                          | 14,5 |
| 3x9    | 27  | 42  | 46                         | 12,1                                                          | 17,1 |
| 3x10   | 30  | 47  | 46,7                       | 13,6                                                          | 19,6 |
| 3x11   | 33  | 52  | 47,2                       | 15,1                                                          | 22,1 |
| 3x12   | 36  | 57  | 47,6                       | 16,6                                                          | 24,6 |
| 4x4    | 16  | 24  | 22,1                       | 6,6                                                           | 7,9  |
| 4x5    | 20  | 31  | 36,5                       | 7,7                                                           | 9,9  |
| 4x6    | 24  | 38  | 42,5                       | 9,6                                                           | 13,1 |
| 4x7    | 28  | 45  | 44,9                       | 11,6                                                          | 16,7 |
| 4x8    | 32  | 52  | 46,1                       | 13,6                                                          | 20,2 |
| CUBE16 | 16  | 24  | 32,8                       | 6,1                                                           | 7,4  |
| CUBE20 | 20  | 30  | 39,4                       | 7,7                                                           | 9,7  |
| SODET1 | 24  | 38  | 38,4                       | 7,6                                                           | 10,1 |
| SODET2 | 26  | 43  | 42,5                       | 8,8                                                           | 12   |

Tabela 8. Współczynnik trafienia i średni rozmiar podproblemów algorytmu *Fact&Cache* w przypadku obliczania niezawodności dwóch terminali ( $|K|=2$ ).

| Sieć    | $n$ | $b$ | współczynnik trafienia [%] | średni rozmiar podproblemu odczytanego z pamięci <i>cache</i> |      |
|---------|-----|-----|----------------------------|---------------------------------------------------------------|------|
|         |     |     |                            | $n$                                                           | $b$  |
| 3x6     | 18  | 27  | 34,6                       | 6,7                                                           | 8,9  |
| 3x7     | 21  | 32  | 37,5                       | 7,4                                                           | 10,2 |
| 3x8     | 24  | 37  | 41                         | 8,8                                                           | 12,5 |
| 3x9     | 27  | 42  | 43,8                       | 10,8                                                          | 15,9 |
| 3x10    | 30  | 47  | 44,9                       | 12,3                                                          | 18,4 |
| 3x11    | 33  | 52  | 45,9                       | 13,8                                                          | 20,9 |
| 3x12    | 36  | 57  | 46,4                       | 15,3                                                          | 23,3 |
| 4x4     | 16  | 24  | 18,9                       | 6,9                                                           | 9,4  |
| 4x5     | 20  | 31  | 31,2                       | 7,3                                                           | 10,2 |
| 4x6     | 24  | 38  | 40,1                       | 8,8                                                           | 12,9 |
| 4x7     | 28  | 45  | 43,5                       | 10,6                                                          | 16   |
| 4x8     | 32  | 52  | 45,2                       | 12,4                                                          | 19,2 |
| CUBE16  | 16  | 24  | 30,6                       | 6,1                                                           | 7,7  |
| CUBE20  | 20  | 30  | 36,1                       | 7,9                                                           | 10,4 |
| SODET1  | 24  | 38  | 34                         | 7,9                                                           | 11   |
| SODET2  | 26  | 43  | 39,5                       | 9                                                             | 13   |
| Arpanet | 21  | 32  | 41,3                       | 9,4                                                           | 12,8 |
| EDF2    | 30  | 52  | 47,6                       | 9,8                                                           | 15   |
| PLG     | 21  | 57  | 45,4                       | 11,2                                                          | 23,2 |

**Tabela 9. Współczynnik trafienia i średni rozmiar podproblemów algorytmu *Fact&Cache* w przypadku obliczania niezawodności wszystkich terminali ( $|K|=|V|$ ).**

W przypadku dużych sieci (np. 3x11, 3x12, 4x8, EDF2, PLG) współczynnik trafienia zbliża się do wartości 50% zarówno w przypadku obliczania niezawodności dwóch terminali, jak i niezawodności wszystkich terminali. Średni rozmiar podproblemów odczytanych z pamięci *cache* (rozumiany jako suma  $n + b$ ), zbliża się do wartości 45% rozmiaru sieci wejściowej (w przypadku takich sieci, jak 3x12 czy PLG).

## 7.7. PODSUMOWANIE ROZDZIAŁU

Dokonano porównania efektywności aktualnie najefektywniejszych algorytmów obliczania niezawodności  $K$ -terminali, wykorzystując zarówno rezultaty uzyskane na drodze analitycznej, jak i eksperymentalnej. Uzyskane rezultaty potwierdziły stosunkowo wysoką efektywność zaproponowanego w tej pracy algorytmu *Fact&Cache*.

## 8. PODSUMOWANIE PRACY

### 8.1. WNIOSKI KOŃCOWE I PODSUMOWANIE REZULTATÓW

Przeprowadzone badania i otrzymane rezultaty mają praktyczne znaczenie dla analizy niezawodności sieci. Efektywność znanych z literatury algorytmów faktoryzacji jest często niewystarczająca. W związku z tym w rozdziale 6. zaproponowano znacznie efektywniejszy algorytm *Fact&Cache*. Algorytm ten dzięki wykorzystaniu zalet programowania dynamicznego jest aktualnie jednym z najefektywniejszych narzędzi umożliwiających analizę niezawodności sieci przez obliczanie niezawodności  $K$ -terminali. W algorytmie wykorzystano fakt, że w trakcie obliczeń te same podproblemy są często rozwiązywane wiele razy. Zamiast wielokrotnie rozwiązywać ten sam podproblem (obliczać niezawodność tej samej sieci), jak ma to miejsce w przypadku wszystkich znanych dotąd algorytmów faktoryzacji, w nowym algorytmie wykorzystuje się wcześniej obliczony i zapamiętany rezultat obliczeń, stosując zmodyfikowaną technikę programowania dynamicznego. Zapobiega to rozwijaniu całych poddrzew binarnego drzewa obliczeń. Fakt, że te same podproblemy mogą wielokrotnie występować w BDO algorytmu faktoryzacji nie został jak dotąd dostrzeżony ani praktycznie wykorzystany.

W rozdziale 4. przedstawiono dotychczasowe osiągnięcia w dziedzinie analizy pesymistycznej złożoności algorytmów faktoryzacji. Najmniejszą pesymistyczną złożonością czasową charakteryzuje się algorytm  $(F_0, \{R1, R2, R3, R4\}, S_4)$  opisany przez Wooda [Wood85, Wood86]. W przypadku tego algorytmu maksymalna liczba liści binarnego drzewa obliczeń algorytmu faktoryzacji  $(|V|-2)!$  jest osiągalna dla przypadków granicznych gdy  $2 \leq |K| \leq 5$  oraz  $|V|-2 \leq |K| \leq |V|$ . Znane są również inne algorytmy faktoryzacji [Satyanarayana83, Wood86], dla których maksymalna liczba liści binarnego drzewa obliczeń jest równa  $(|V|-1)!$  dla dowolnego zbioru  $K$  ( $2 \leq |K| \leq |V|$ ). W rozdziale 5. uzyskano lepszy rezultat proponując algorytm *Fact*  $(F_0, \{R0, R1, R2, R3\}, S'')$ . Dzięki nowej strategii  $S''$  selekcji krawędzi do faktoryzacji, odpowiednio dobranemu zbiorowi redukcji, jak również nowej metodzie analizy pesymistycznej złożoności obliczeniowej algorytmu wykazano, że maksymalna liczba  $(|V|-2)!$  liści binarnego drzewa obliczeń powyższego algorytmu jest osiągalna dla dowolnego zbioru  $K$  ( $2 \leq |K| \leq |V|$ ), a nie tylko dla przypadków granicznych, gdy  $2 \leq |K| \leq 5$  oraz  $|V|-2 \leq |K| \leq |V|$  (jak to ma miejsce w przypadku algorytmu

analizowanego przez Wooda). Rezultat ten uzyskano przy prostszym niż zaproponowany przez Wooda zbiorze zachowujących niezawodność redukcji grafu  $\mathbf{R}=\{R_0, R_1, R_2, R_3\}$ . Nie ma bowiem konieczności uwzględniania najbardziej skomplikowanego podzbioru redukcji  $R_4$  (redukcji wielokąta do łańcucha). Również zaproponowana strategia selekcji krawędzi do faktoryzacji nie wymaga stosowania złożonych procedur testowania własności (np. koherentności czy dwuspójności) sieci występujących w poszczególnych węzłach BDO. W porównaniu do algorytmów analizowanych przez Wooda, Satyanarayana i Changa [Satyanarayana83, Wood86] uzyskano znaczne, bo  $(|V|-1)$ -krotne, zmniejszenie maksymalnej liczby liści binarnego drzewa obliczeń.

W rozdziale 5. zaprezentowano metodę analizy pesymistycznej złożoności obliczeniowej algorytmu *Fact*, jak również odmian tego algorytmu faktoryzacji wykorzystujących różne zbiory zachowujących niezawodność redukcji grafów.

Algorytmy *Fact* i *Fact&Cache* zostały zaimplementowane w języku C++. W celu weryfikacji efektywności zaproponowanych algorytmów w rozdziale 7. dokonano porównania najefektywniejszych algorytmów i programów obliczania niezawodności  $K$ -terminali wykorzystując zarówno rezultaty uzyskane analitycznie, jak i eksperymentalnie (z wykorzystaniem typowych, rozpatrywanych w literaturze, sieci testowych).

Za zasadniczy wkład pracy autor uważa:

- Zaproponowanie dwóch nowych, konkurencyjnych w stosunku do znanych dotąd algorytmów faktoryzacji, algorytmów *Fact* i *Fact&Cache* obliczania niezawodności  $K$ -terminali:
  - w przypadku algorytmu *Fact* możemy mieć pewność, że maksymalna liczba  $(n-2)!$  liści binarnego drzewa obliczeń jest osiągalna dla dowolnego zbioru  $K$  ( $2 \leq |K| \leq n$ ), (jest to aktualnie najlepszy uzyskany analitycznie rezultat);
  - algorytm *Fact&Cache* [Madeyski98a, Madeyski98b] dzięki wykorzystaniu techniki częściowego spamiętywania (*partial memoization*) [Madeyski99a] będącej odmianą techniki programowania dynamicznego jest aktualnie jednym z najefektywniejszych narzędzi umożliwiających analizę niezawodności sieci (fakt, że te same podproblemy mogą wielokrotnie występować w BDO algorytmu faktoryzacji nie został jak dotąd dostrzeżony ani praktycznie wykorzystany).

- Zaproponowanie nowej metody analizy pesymistycznej złożoności obliczeniowej algorytmu *Fact* jak również odmian tego algorytmu wykorzystujących różne zbiory zachowujących niezawodność redukcji grafu [Madeyski99b].

Powyższe algorytmy zaimplementowano w języku C++ i porównano z rozwiązaniami konkurencyjnymi wykorzystując rezultaty uzyskane analitycznie jak i eksperymentalnie (z użyciem typowych – wykorzystywanych w literaturze – sieci testowych) potwierdzając efektywność zaproponowanych rozwiązań.

W kolejnym rozdziale zwrócono uwagę na zagadnienie syntezy niezawodnych sieci.

## 8.2. PROJEKTOWANIE NIEZAWODNYCH SIECI

Chociaż problem syntezy niezawodnych sieci wykracza poza zakres tej pracy, warto jednak zasygnalizować możliwość wykorzystania rozważanych w tej pracy algorytmów faktoryzacji przy projektowaniu niezawodnych sieci.

Jedną z możliwych strategii projektowania niezawodnych sieci jest wykorzystanie algorytmów umożliwiających analizę niezawodności sieci<sup>77</sup>. Sieć po zaprojektowaniu jest poddawana analizie, aby określić jej niezawodność. Jeśli nie jest ona satysfakcjonująca, to projekt sieci jest korygowany (z wykorzystaniem innych kryteriów projektowych, bądź manualnie). Wytworzony, zmodyfikowany projekt sieci jest poddawany powtórnej analizie niezawodnościowej. Procedura ta, opisana przez Balla, Colbourn i Provana<sup>78</sup>, jest powtarzana aż do uzyskania zadowalającego rezultatu. Podobną procedurę polegającą na konstruowaniu możliwych topologii sieci i porównywaniu ich niezawodności opisuje Myrvold<sup>79</sup>. Powyższe procedury zakładają wykorzystanie algorytmów do analizy niezawodności sieci.

Jeżeli skorygowany projekt sieci niewiele różni się od poprzednio rozpatrywanego np. zawiera jedno łącze więcej lub jedno łącze mniej (zwykle ilość węzłów sieci rozpatrywanych podczas projektu sieci nie ulega zmianie), to można w prosty sposób zwiększyć efektywność analizy niezawodności. Jeżeli rozpatrywany właśnie graf  $H$  zawiera o jedną krawędź więcej niż poprzednio rozpatrywany graf  $G$  czyli  $H = G + e$ , to zgodnie z twierdzeniem faktoryzacji

---

<sup>77</sup> Por. [Myrvold96], rozdział „Analysis-Progress and Proposed Research”.

<sup>78</sup> Por. [Ball95], s.4.

<sup>79</sup> Por. [Myrvold96], rozdział „Analysis-Progress and Proposed Research”.



$$R(H) = p_e R(H * e) + q_e R(H - e),$$

czyli:

$$R(H) = p_e R(H * e) + q_e R(G).$$

Ponieważ jeden z dwóch podproblemów reprezentowany przez niezawodność grafu  $G$  był wcześniej rozwiązany, wystarczy rozwiązać jedynie drugi podproblem.

Powyższy schemat postępowania umożliwia wykorzystanie algorytmów faktoryzacji, w tym także algorytmu *Fact&Cache*, przy projektowaniu niezawodnych sieci. Wymaga jednak drobnej modyfikacji programu komputerowego będącego implementacją algorytmu w celu ułatwienia korzystania z powyższego schematu postępowania.

W następnym rozdziale przedstawione zostaną propozycje dalszych badań w tym także możliwość modyfikacji algorytmu *Fact&Cache* w celu wykorzystania go do oceny istotności łączy przy projektowaniu niezawodnych sieci.

### 8.3. PROPOZYCJE DALSZYCH BADAŃ I MODYFIKACJI ZAPROPONOWANYCH ALGORYTMÓW

W rozdziale tym zaprezentowane zostaną propozycje dalszych badań, w szczególności określenia i zintegrowania z zaproponowanym algorytmem *Fact&Cache* miar zdolności do prawidłowego działania (*performability measures*), modyfikacji zastosowanej techniki generowania kodów podproblemów, jak również wyszukiwania równoważnych podproblemów. Omówione zostaną również możliwości modyfikacji zaproponowanych algorytmów w celu obliczania miar niezawodności sieci o zawodnych węzłach jak również uzyskania algorytmu aproksymacyjnego. Przedstawiona zostanie także możliwość modyfikacji algorytmu *Fact&Cache* pod kątem wykorzystania go przy projektowaniu niezawodnych sieci.

#### 8.3.1. Zastosowanie miar zdolności do prawidłowego działania (*performability measures*)

Niezawodność sieci jest zwykle wyrażana za pomocą miar bazujących na spójności sieci. Jednak w przypadku, kiedy miary bazujące na spójności sieci wydają się

nieodpowiednie, rozpatruje się miary zdolności do prawidłowego działania (*performability measures*). Zdefiniowana wcześniej oczekiwana wydajność (*expected performance*) jest jedną z najpowszechniej wykorzystywanych miar zdolności do prawidłowego działania<sup>80</sup>. Jeżeli przyjąć, że wykorzystywana tam funkcja  $\psi$  będąca miarą wydajności jest prostą funkcją binarną wskazującą istnienie  $K$ -drzewa, to algorytmy *Fact* i *Fact&Cache* można uznać za algorytmy obliczania oczekiwanej wydajności. Istnieje więc możliwość prostego rozbudowania każdego z powyższych algorytmów, tak, aby wykorzystywał on bardziej zaawansowane miary wydajności uwzględniające na przykład wymagania przepływu, czy pojemności łączy. Dalsze badania powinny iść w kierunku określenia odpowiedniej miary wydajności  $\psi(S)$  i zintegrowania jej z zaproponowanym algorytmem faktoryzacji *Fact&Cache*.

### 8.3.2. Modyfikacja techniki wykrywania równoważnych podproblemów

Niezwykle istotnym kierunkiem dalszych badań jest modyfikacja techniki wykrywania równoważnych podproblemów w celu dalszego zwiększenia efektywności zaproponowanego algorytmu *Fact&Cache*, a co za tym idzie możliwości dokładnej analizy niezawodnościowej większych niż dotychczas to było możliwe sieci. W obecnej postaci algorytm ten wykorzystuje efektywny sposób kodowania podproblemów, gwarantujący równoważność podproblemów, jeżeli kody są identyczne. Zależność ta nie jest jednak prawdziwa w drugą stronę, dlatego skuteczność wykrywania równoważnych podproblemów nie jest stuprocentowa. Z tego względu możliwe są dwie strategie postępowania: modyfikacja aktualnie używanej techniki w celu zwiększenia jej skuteczności i zaproponowanie nowej techniki gwarantującej wykrywanie równoważnych podproblemów. O ile pierwsza strategia postępowania jest kontynuacją prowadzonych dotąd prac, o tyle druga strategia postępowania wyznacza odrębny obszar badań umożliwiając wykorzystanie osiągnięć w dziedzinie testowania izomorfizmu i kanonicznego etykietowania grafów.

---

<sup>80</sup> Por. rozdział 1.3.

### 8.3.2.1. Wykorzystanie osiągnięć w dziedzinie testowania izomorfizmu i kanonicznego etykietowania grafów

Konstrukcja algorytmu *Fact&Cache* umożliwia wykorzystanie osiągnięć w dziedzinie testowania izomorfizmu grafów (GI) do dalszego zwiększenia efektywności algorytmu. Jest to tym bardziej istotne, że część badaczy przytacza argumenty za tym, że problem GI nie jest problemem NP-zupełnym [Mathon79]. Efektywne testowanie izomorfizmu grafów wciąż stanowi temat intensywnych badań i nie jest znany wielomianowy (w najgorszym przypadku) algorytm testowania izomorfizmu dowolnych grafów. Zaproponowane algorytmy testowania izomorfizmu grafów można generalnie podzielić na dwie grupy. Są to albo efektywne algorytmy testowania izomorfizmu pewnych podklas grafów (np. grafów planarnych), albo nieefektywne (w najgorszym przypadku) algorytmy testowania izomorfizmu dowolnych grafów. Spośród wielu zaproponowanych algorytmów testowania izomorfizmu, warto wymienić przynajmniej kilka. Hopcroft i Tarjan [Hopcroft72, Hopcroft73] zaproponowali algorytm testowania izomorfizmu grafów planarnych o złożoności  $O(n \log n)$ . Balińska i Miądowicz [Balińska95] zaproponowali interesujący, prosty implementacyjnie algorytm testowania izomorfizmu dwóch grafów o pesymistycznej złożoności  $O((n/2)^{n-2})$ . Algorytm wykorzystuje macierz przyległości, ciąg stopni i posortowany ciąg stopni. Jeżeli grafy mają dużą liczbę krawędzi, to testuje się izomorfizm ich dopełnień. W wyniku przeprowadzonych testów autorzy pokazali między innymi, że w przypadku grafów losowych oczekiwana złożoność czasowa algorytmu jest funkcją kwadratową względem liczby wierzchołków. Ostatnio Bennett i Edwards [Bennett96] przedstawili algorytm do testowania izomorfizmu, wykorzystujący pseudoinwersję o złożoności  $O(nb^2)$  „w prawie wszystkich przypadkach”. Średnia złożoność czasowa wielu algorytmów testowania izomorfizmu grafów jest bardzo niska. Babai i Kucera [Babai79, Kucera87] zaprezentowali z kolei algorytmy wykorzystujące kanoniczne etykietowanie grafów, których średnia złożoność czasowa jest liniowa.

Algorytm kanonicznego etykietowania przekształca graf wejściowy  $G$  na jego formę kanoniczną (*canonical form*)  $C(G)$ . Kanonicznie zaetykietowany graf – zwany niekiedy kanonicznym reprezentantem klas podobieństwa grafów<sup>81</sup> – można opisać za pomocą etykiety kanonicznej (*canonical labeling*), nazywanej również wektorem będącym

---

<sup>81</sup> Por. [Kulikowski86], s.400.

kanonicznym reprezentantem swojej klasy<sup>82</sup>. Kanoniczne etykietowanie gwarantuje, że  $C(G) = C(H)$  wtedy i tylko wtedy, gdy  $G$  i  $H$  są izomorficzne.

Chociaż Babai i Kucera [Babai79, Kucera87] zaproponowali algorytmy wykorzystujące kanoniczne etykietowanie grafów, których średnia złożoność czasowa jest liniowa, to nie jest znany algorytm kanonicznego etykietowania dowolnych grafów, którego złożoność czasowa w najgorszym przypadku byłaby wielomianowa [Babai80]. Najefektywniejszy znany algorytm kanonicznego etykietowania dowolnych grafów o  $n$  wierzchołkach ma w najgorszym przypadku złożoność  $O(\exp(n^{1/2+o(1)}))$  [Luks82]. Wykorzystując teorię grup udowodniono, że dla pewnych typów grafów kanoniczne formy mogą być obliczone w czasie wielomianowym. Dotyczy to między innymi grafów o ograniczonym stopniu, co pokazał Babai i Luks [Babai83] oraz niezależnie Fürer, Schnyder i Specker [Fürer83]<sup>83</sup>.

McKay natomiast [McKay81] badał praktyczne możliwości testowania izomorfizmu grafów. Zaproponował i praktycznie zaimplementował w języku C (pakiet NAUTY [McKay98]) procedury określania grupy automorficznej i kanonicznego etykietowania grafu. Pakiet NAUTY jest uważany aktualnie za najefektywniejszy praktycznie zaimplementowany algorytm kanonicznego etykietowania i testowania izomorfizmu grafów. Algorytm ten należy do klasy algorytmów z powrotami. W celu zwiększenia efektywności wykorzystuje on haszowanie częściowych informacji o etykietach węzłów, jak również umożliwia użytkownikowi opcjonalne korzystanie z różnych heurystyk (niezmienników grafów). Analiza złożoności czasowej tego algorytmu jest więc utrudniona. Jednak Miyazaki<sup>84</sup> [Miyazaki97] próbując badać złożoność czasową tego algorytmu wykazał, że dla pewnych klas grafów złożoność ta ma charakter wykładniczy.

Metoda kanonicznego etykietowania grafu jest interesująca nie tylko ze względu na jej stosunkowo wysoką efektywność<sup>85</sup>. Wykorzystanie tej metody w algorytmie *Fact&Cache* byłoby słuszne ze względu na konieczność wielokrotnego testowania izomorfizmu z

---

<sup>82</sup> Tamże, s.401.

<sup>83</sup> Por. [Miyazaki97], s.239.

<sup>84</sup> Takunari Miyazaki, Department of Computer and Information Science, University of Oregon, e-mail: miyazaki@cs.uoregon.edu.

<sup>85</sup> Por. [Fortin96], s.12. Pakiet NAUTY jest uważany za jeden z najszybszych obecnie pakietów programowych umożliwiających testowanie izomorfizmu grafów.

wykorzystaniem grafów z zarejestrowanych podproblemów<sup>86</sup>. Grafy te byłyby opisane za pomocą etykiet kanonicznych (wektorów będących kanonicznymi reprezentantami swoich klas podobieństwa). Ten kierunek dalszych badań wydaje się szczególnie obiecujący.

### 8.3.3. Modyfikacja funkcji haszującej wykorzystywanej do wyszukiwania równoważnych podproblemów

Jest możliwe niewielkie zwiększenie efektywności algorytmu *Fact&Cache* poprzez udoskonalenie funkcji haszującej metody wyszukiwania równoważnych podproblemów. Tablice haszujące<sup>87</sup> są jedną z najefektywniejszych metod wyszukiwania (średnia złożoność czasowa wyszukiwania jest  $O(1)$ , pesymistyczna złożoność czasowa wyszukiwania jest  $O(n)$ ). Chcąc jednak ograniczyć pesymistyczną złożoność czasową wyszukiwania, można zastosować drzewa czerwono-czarne, których zarówno średnia, jak i pesymistyczna złożoność wyszukiwania wynosi  $O(\lg n)$ .

### 8.3.4. Faktoryzacja i redukcje na sieciach o zawodnych węzłach

Wierzchołki ze zbioru  $K$  mogą być uważane za niezawodne<sup>88</sup>, gdyż w przeciwnym wypadku niezawodność oryginalnego grafu można zapisać następująco:

$$R(G_K) = \prod_{v \in K} p_v R(G'_K),$$

gdzie  $G'_K$  jest  $G_K$  z niezawodnymi wierzchołkami  $K$ .

Z powyższego wynika, iż nie ma potrzeby dokonywania poważniejszych modyfikacji algorytmów faktoryzacji w przypadku obliczania niezawodności  $K$ -terminali sieci o zawodnych węzłach, jeżeli tylko znajdują się one w zbiorze  $K$ . W przypadku obliczania niezawodności wszystkich terminali wszystkie wierzchołki zawarte są w zbiorze  $K$ .

Jednak w ogólnym przypadku konieczna jest modyfikacja algorytmu faktoryzacji. Wykorzystując technikę zaproponowaną przez Theologou i Carliera [Theologou91]

---

<sup>86</sup> Por. [Kulikowski86], s.400-401.

<sup>87</sup> Są one również nazywane tablicami mieszającymi.

<sup>88</sup> Por. [Theologou91] s.212.

zaimplementowano wstępną wersję algorytmu faktoryzacji umożliwiającego obliczanie niezawodności  $K$ -terminali sieci, których nie tylko łączy, lecz również węzły są zawodne. Algorytm wykorzystuje odmienną niż w algorytmie *Fact* strategię selekcji krawędzi do faktoryzacji (wybrana krawędź musi mieć co najmniej jeden niezawodny wierzchołek). Modyfikacji uległy również zachowujące niezawodność redukcje grafu. Dalsze badania mogłyby iść w kierunku opracowania efektywnej (być może bardziej restrykcyjnej) strategii selekcji krawędzi do faktoryzacji, a w szczególności analitycznego określenia pesymistycznej złożoności algorytmu faktoryzacji do obliczania niezawodności  $K$ -terminali sieci o zawodnych węzłach. Byłoby to o tyle istotne, że Theologou i Carlier [Theologou91] analizowali efektywność swego algorytmu jedynie eksperymentalnie.

### 8.3.5. Faktoryzacja z aproksymacją

Pomysł stworzenia algorytmu aproksymacyjnego na bazie algorytmu faktoryzacji został najprawdopodobniej po raz pierwszy przedstawiony przez Balla i Van Slyke'a [Ball77]<sup>89</sup>. Był on później wykorzystywany przez Feo i Johnsona [Feo90] w algorytmie aproksymującym niezawodność dwóch terminali grafów pełnych oraz przez Carliera i Theologou [Carlier91] do określania ograniczeń na niezawodność  $K$ -terminali dowolnych sieci.

Algorytm aproksymujący może znaleźć zastosowanie w trudnych obliczeniowo przypadkach analizy niezawodności złożonych sieci. Zamiast określenia dokładnej wartości niezawodności  $K$ -terminali, technika ta umożliwia szybsze określenie górnego i dolnego ograniczenia na wartość niezawodności  $K$ -terminali.

Technika aproksymacji w algorytmie faktoryzacji polega na tym, że w pewnym momencie obliczania niezawodności, poprzez rekurencyjne stosowanie formuły faktoryzacji i stosowanie zachowujących niezawodność redukcji, możemy efektywnie aproksymować pewne węzły BDO zamiast dalej rozwijać je stosując faktoryzację. W wyniku faktoryzacji i redukcji niezawodność  $K$ -terminali przedstawiona jest w postaci sumy iloczynów niezawodności sieci reprezentowanych w liściach BDO i współczynnika którego wartość wynika z przeprowadzonych redukcji i niezawodności łączy wybranych do faktoryzacji. Jeśli ów współczynnik dla danego węzła BDO będzie mały to znikomy będzie również wpływ

---

<sup>89</sup> Por. [Feo90] s.291.



aproxymacji niezawodności  $K$ -terminali tego węzła na wartość niezawodności  $K$ -terminali całej sieci. BDO jest na początku rozwijane tak samo jak w algorytmie faktoryzacji nie wykorzystującym techniki aproxymacyjnej do momentu w którym wartość owego współczynnika jest odpowiednio mała (wartość ta może zostać podana przez użytkownika). Od tego momentu obliczane jest górne lub dolne ograniczenie na niezawodność  $K$ -terminali.

Niestety uzyskane przez Carliera i Theologou [Carlier91] ograniczenia mogą być satysfakcjonujące jedynie przy dużych niezawodnościach łączy<sup>90</sup>. Interesująca wydaje się jednak możliwość poszukiwania (w ramach dalszych badań) lepszych schematów aproxymujących do konstruowania górnych i dolnych ograniczeń na niezawodność  $K$ -terminali.

### 8.3.6. Wykorzystanie zmodyfikowanego algorytmu *Fact&Cache* przy projektowaniu niezawodnych sieci

Projektantów sieci od dawna nurtowało pytanie, czy zawsze istnieje sieć optymalna ze względu na rozważaną miarę niezawodności (*uniformly-most reliable network*), bez względu na wartości niezawodności przypisane łączom. Odpowiedź na to pytanie jest niestety negatywna [Kelmans81, Myrvold96], jakkolwiek sieci tego typu są tematem badań. Podejmowane są próby określenia, w jakich przypadkach tego typu sieci istnieją [Myrvold96]. Ogólnie jednak możliwe jest istnienie dwóch sieci, z których jedna jest bardziej niezawodna dla małych, jak i dużych prawdopodobieństw działania łączy, podczas gdy druga jest bardziej niezawodna dla średnich wartości prawdopodobieństw działania łączy [Myrvold96]. Możliwa jest bowiem sytuacja, w której wielomiany niezawodności<sup>91</sup> mogą się przecinać [Colbourn93]. Ponadto dwa grafy, które nie są izomorficzne mogą mieć te same wielomiany niezawodności, czyli być równoważne niezawodnościowo (*reliability equivalent*) [Myrvold96].

Z powyższych względów Page i Perry [Page94] zaproponowali metodę pozwalającą porównywać istotność łączy bez względu na wartości prawdopodobieństw działania łączy. Dla projektanta sieci możliwość określenia, które łącza są istotniejsze, a które mniej istotne z

---

<sup>90</sup> Por. [Carlier91] s11.

<sup>91</sup> W przypadku, gdy prawdopodobieństwa uszkodzenia wszystkich łączy w sieci są takie same ( $\forall i \ p_i = p$ ), niezawodność systemu może być zapisana w postaci wielomianu w funkcji  $p$ .

punktu widzenia liczonej miary niezawodności sieci odgrywa bowiem równie istotną rolę, jak sama wartość obliczonej miary niezawodności sieci. Powyższa metoda wykorzystuje wielomiany niezawodności dla grafów  $G_K * e_i$  oraz  $G_K - e_i$ . Mogą być one łatwo generowane z wykorzystaniem mechanizmu faktoryzacji<sup>92</sup> wykorzystywanego między innymi w omawianych w niniejszej pracy algorytmach *Fact* i *Fact&Cache*. Jak wynika z pracy Page'a i Perry zaproponowana metoda jest znacznie dokładniejsza niż metody wykorzystujące inne miary kombinatoryczne bazujące na zbiorach ścieżek i cięć grafu. Jeżeli  $G_K$  opisuje rozpatrywany graf z wyróżnionym zbiorem wierzchołków  $K$ , a  $e_1$  i  $e_2$  są rozpatrywanymi krawędziami grafu, to krawędź  $e_1$  jest co najmniej tak istotna, jak krawędź  $e_2$  (co jest oznaczane jako  $e_1 > e_2$ ), jeżeli:

- $R(G_K - e_2) \geq R(G_K - e_1)$  dla wszystkich  $p \in (0,1)$ ,
- $R(G_K * e_2) \leq R(G_K * e_1)$  dla wszystkich  $p \in (0,1)$ .

Jeżeli  $e_1 > e_2$  i  $e_2 > e_1$ , to są one ekwiwalentne ( $e_1 \approx e_2$ ).

Oczywiście zaproponowany w tej pracy algorytm *Fact&Cache* umożliwia bez jakichkolwiek przeróbek określenie istotności łączy dla konkretnych wartości prawdopodobieństw działania łączy. Jednak Page i Perry słusznie proponują modyfikację algorytmu faktoryzacji tak, aby możliwe było określenie istotności łączy dla dowolnych wartości prawdopodobieństwa działania łączy. Ma to olbrzymie znaczenie z punktu widzenia projektowania sieci, gdyż zapewnia, że względna istotność łączy może być porównywana bez wstępnej, dokładnej wiedzy o wartości niezawodności łączy. Page i Perry proponują wykorzystanie standardowego algorytmu faktoryzacji, w którym krawędziom zamiast określających niezawodność liczb rzeczywistych, przypisane są wyrażenia będące wielomianami.

W istocie zamiast modyfikować standardowy algorytm faktoryzacji można modyfikować znacznie efektywniejszy algorytm faktoryzacji *Fact&Cache* wykorzystujący technikę wykrywania równoważnych podproblemów. Odpowiednio zmodyfikowany algorytm *Fact&Cache* może więc znaleźć zastosowanie także w dziedzinie projektowania, a nie tylko analizy sieci.

---

<sup>92</sup> Por. [Page94], s.57.



## BIBLIOGRAFIA

- [AboElFotoh89] H. M. AboElFotoh, C. J. Colbourn, Series-Parallel Bounds for the Two-Terminal Reliability Problem, *ORSA J. Comp.* 1989(1). 209-222.
- [Abraham79] J. A. Abraham, An improved algorithm for network reliability, *IEEE Trans. Reliability* 1979 (R-28), 58-61.
- [Aggarwal75] K. K. Aggarwal, K. B. Misra, J. S. Gupta, A fast algorithm for reliability evaluation, *IEEE Trans. Reliability* 1975 (R-24), 83-85.
- [Agrawal84] A. Agrawal, R. E. Barlow, A survey of network reliability domination theory, *Operations Research* 1984, Vol. 32(3), 478-492.
- [Amin88] A. T. Amin, K. T. Siegrist, P. J. Slater, Pair-connected reliability of communication networks with vertex failures, *Congressus Numeratium* 67, 233-242.
- [Assous84] J. Y. Assous, Bounds for terminal reliability, *Preprint*, Temple University, 1984.
- [Babai79] L. Babai, L. Kucera: Canonical Labelling of Graphs in Linear Average Time, *20th IEEE Annual Symposium on Foundations of Computer Science* 1979, 39-46.
- [Babai80] L. Babai. On the complexity of canonical labeling of strongly regular graphs, *SIAM Journal on Computing*, February 1980, 212-216.
- [Babai83] L. Babai and E. M. Luks. Canonical labeling of graphs, *Proceedings of the 15<sup>th</sup> Fifteenth Annual ACM Symposium on Theory of Computing (STOC 1983)*, 171-183.
- [Bailey86] M. P. Bailey, V. G. Kulkarni, A recursive algorithm for computing exact reliability measures, *IEEE Trans. Reliability*, 1986 (35), 36-40.
- [Balińska95] K. T. Balińska, J. Miądowicz: A graph isomorphism algorithm, *Graph Theory Notes of New York XXIX* 1995(8), 33-37.
- [Balińska98] K. T. Balińska, Informacja ustna, *Seminarium naukowe Instytutu Informatyki Politechniki Poznańskiej* (Katedra Automatyki, Robotyki i Informatyki, Politechnika Poznańska, e-mail:balinska@rose.man.poznan.pl).
- [Ball77] M. Ball, R. Van Slyke. Backtracking algorithms for network reliability analysis, *Ann. Discrete Mathematics I: Studies in Integer Programming*, HammerJohnson, Korte, Nemhauser (Eds), 1977, 49-64.

- [Ball80] M. O. Ball, Complexity of network reliability computations, *Networks* 1980 (10), 153-165.
- [Ball82] M. O. Ball, J. S. Provan, Bounds on the reliability polynomial for shellable independence systems, *SIAM J. Alg. Disc. Math.* 1982(3), 166-181.
- [Ball83] M. O. Ball, J. S. Provan, The complexity of counting cuts and of computing the probability that a graph is connected, *SIAM J. Comp.* 1983 (12), 777-788.
- [Ball84] M. O. Ball, J. S. Provan, Computing network reliability in time polynomial in the number of cuts, *Operations Research* 1984 (32), 516-526.
- [Ball86] M. O. Ball, An overview of the complexity of network reliability computations, *IEEE Trans. Reliability*, 1986 (R-36) Aug., 230-239.
- [Ball95] M.O. Ball, C. J. Colbourn, J. S. Provan, Network reliability, *Handbook of Operations Research: Network Models*, Elsevier North-Holland 1995, 673-762. (tekst dostępny także w formie raportu M.O. Ball, C. J. Colbourn, J. S. Provan, *Network reliability* TR92-74, Institute for Systems Research, University of Maryland, 1992).
- [Beichelt91a] F. Beichelt, P. Tittmann, A Generalized Reduction Method for the Connectedness Probability of Stochastic Networks, *IEEE Trans. Reliability* 1991 (40), 198-204.
- [Beichelt91b] F. Beichelt, P. Tittmann, Reliability analysis of communication networks by decomposition, *Microelectron. Reliab.* 1991 (31), 868-872.
- [Bennett96] J. M. Bennett, J. J. Edwards, A graph isomorphism algorithm using pseudoinverses, *BIT* 1996(36:1), 41-53.
- [Boesch86] F. T. Boesch, Synthesis of reliable networks – a survey, *IEEE Trans. Reliability* 1986 (35), 240-246.
- [Brecht85] T. B. Brecht, C. J. Colbourn, Lower bounds for two-terminal network reliability, *CCNG Report E-127*, University of Waterloo 1985. Także [w:] *Discrete Appl. Math.* 1988 (21), 185-198.
- [Brecht86] T. B. Brecht, C. J. Colbourn, Improving Reliability Bounds in Computer Networks, *Networks* 1986 (16), 369-380.
- [Brecht89] T. B. Brecht, C. J. Colbourn, Multiplicative Improvements in Network Reliability Bounds, *Networks* 1989 (19), 521-529.

- [Brown71] D. B. Brown, A computerized algorithm for determining the reliability of redundant configurations, *IEEE Trans. Reliability* 1971(20), 121-124.
- [Brown93] J. I. Brown, C. J. Colbourn, J. S. Devitt, Network Transformations and Bounding Network Reliability, *Networks* 1993 (23), 1-17.
- [Buzacott80] J. A. Buzacott, A recursive algorithm for finding reliability measures related to the connection of nodes in a graph, *Networks* 1980 (15), 311-327.
- [Calvert97] K. L. Calvert, M. B. Doar, Modeling Internet Topology, *IEEE Communications Magazine* June 1997.
- [Cancela94] H. Cancela, M. E. Khadiri, A recursive variance reduction algorithm for estimating communication network reliability, *Technical Report 860*, IRISA, Campus de Beaulieu, 35042 Rennes, France, 1994.
- [Cancela95] H. Cancela, M. E. Khadiri, Recursive path conditioning Monte Carlo simulation of communication network reliability, *Technical Report 915*, IRISA, Campus de Beaulieu, 35042 Rennes, France, 1995.
- [Carlier91] J. G. Carlier, O. R. Theologou, An efficient program form computing undirected networks reliability, R. Avenhaus, H. Karkar i M. Rudnianski ed., *Defense Decision Making* (Springer, Berlin, 1991), 3-13.
- [Carlier96] J. Carlier, C. Lucet, A decomposition algorithm for network reliability evaluation, *Discrete Appl. Math.* 1996(65), 141-156.
- [Carrasco85] E. H. Carrasco, C. J. Colbourn, Reliability bounds for networks with statistical dependence, *Preprint*, University of Waterloo, 1985.
- [Chari90] M. K. Chari, Combinatorial Aspects of Degeneracy in Linear Systems of Network Reliability, *Ph.D. Thesis*, Univ. of North Carolina, 1990.
- [Chinneck97] J. W. Chinneck, How to Organize your Thesis, November 19, 1997, (Prof. John W. Chinneck, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada, e-mail: chinneck@sce.carleton.ca).
- [Choi95] M. S. Choi, C. H. Jun, Some variants of polygon-to-chain reductions in evaluation reliability of undirected network, *Microelectron. Reliab.* 1995(35), 1-11.
- [Colbourn87] C. J. Colbourn, *The Combinatorics of Network Reliability*, Oxford University Press, New York 1987.

- [Colbourn88] C. J. Colbourn, D. D. Harms, Bounding All-Terminal Reliability in Computer Networks, *Networks* 1988 (18), 1-12.
- [Colbourn93] C. J. Colbourn, D. D. Harms, W. J. Myrvold, Brief communication: Reliability polynomials can cross twice, *The Franklin Institute* 1993 (330), 629-633.
- [Colbourn98] C. J. Colbourn, Reliability issues in telecommunications network planning, *Telecommunications Network Planning*, (oddane do druku, dostępne przez Internet <http://www.emba.uvm.edu/~colbourn/>).
- [Cormen97] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Wprowadzenie do algorytmów*, WNT 1997.
- [Debany86] W. H. Debany, P. K. Varshney, C. R. P. Hartman, Network reliability evaluation using probability expressions, *IEEE Trans. Reliability* 1986 (35), 161-166.
- [Deo80] N. Deo, *Teoria grafów i jej zastosowania w technice i informatyce*, PWN, Warszawa 1980.
- [Deo92] N. Deo, M. Medidi, Parallel algorithm for terminal-pair reliability, *IEEE Trans. Reliability* 1992 (41), 201-209.
- [Dotson79] W. P. Dotson, J. O. Gobein, A New Analysis Technique for Probabilistic Graphs, *IEEE Trans. on Circuits and Systems* 1979 (CAS-26), 855-865.
- [Easton80] M. C. Easton, C. K. Wong, Sequential destruction method for Monte Carlo evaluation of system reliability, *IEEE Trans. Reliability* 1980 (R-29) Apr., 27-32.
- [Elperin91] T. Elperin, I. Gertsbakh, M. Lomonosov, Estimation of Network Reliability Using Graph Evolution Models, *IEEE Trans. Reliability* 1991(40), 572-581.
- [Feo90] T. A. Feo, R. Johnson, Partial Factoring: An Efficient Algorithm for Approximating 2-Terminal Reliability on Complete Graphs, *IEEE Trans. Reliability* 1990 (R-39), 290-295.
- [Fishman86a] G. S. Fishman, A Monte Carlo sampling plan for estimating network reliability, *Operations Research* 1986 (34), 581-592.
- [Fishman86b] G. S. Fishman, A comparison of four Monte Carlo methods for estimating the probability of s-t connectedness, *IEEE Trans. Reliability* 1986 (R-35), 145-154.
- [Fishman87] G. S. Fishman, A Monte Carlo sampling plan for estimating reliability parameters and related functions, *Networks* 1987 (17), 169-186.

- [Fishman89] G. S. Fishman, Estimation the s-t reliability function using importance and stratified sampling, *Operations Research* 1989 (37), 462-473.
- [Fortin96] S. Fortin, The graph isomorphism problem, *TR 96-20*, Department of Computer Science, The University of Alberta, Edmonton, Alberta, Canada.
- [Frank72] H. Frank, R. E. Kahn, L. Kleinrock, Computer communications network design: Experience with theory and practice, *Networks* 1972 (2), 135-166.
- [Fratta73] L. Fratta, U. G. Montanari, A boolean algebra method for computing the terminal reliability in a communication network, *IEEE Trans. Circuit Theory* 1973 (CT-20) May, 203-211.
- [Fratta78] L. Fratta, U. G. Montanari, A recursive method based on case analysis for computing network terminal reliability, *IEEE Trans. Communications* 1978 (COM-26) Aug., 1166-1177.
- [Fürer83] W. Fürer, W. Schnyder, E. Specker, Normal forms for trivalent graphs and graphs of bounded valence, *Proceedings of the Fifteenth Annual ACM Symposium on the Theory of Computing (STOC 1983)*, 161-170.
- [Gilbert59] E. N. Gilbert, Random graphs, *Ann. Math. Stat.* 1959(30), 1141-1144.
- [Graham96] R. L. Graham, D. E. Knuth, O. Patashnik, *Matematyka konkretna*, PWN 1996.
- [Harms83] D. D. Harms, An investigation into bounds on network reliability, *M.Sc. thesis*, University of Saskatchewan, Canada, 1983.
- [Harms93] D. D. Harms, C. J. Colbourn, Renormalization of Two-Terminal Reliability, *Networks* 1993(23), 289-297.
- [Hopcroft72] J. E. Hopcroft, R. E. Tarjan, Isomorphism of planar graphs (working paper), *Complexity of Computer Computations* (R. E. Miller, J. W. Thatcher), Plenum Press, New York, 1972.
- [Hopcroft73] J. E. Hopcroft, R. E. Tarjan, A VlogV algorithm for isomorphism of triconnected planar graphs, *Journal of Computer and System Sciences* 1973(7), 323-331.
- [Jacobs59] I. M. Jacobs, Connectivity in probabilistic graphs, *Technical Report 356*, ERL, MIT, 1959.
- [Jóźwiak79] I. J. Jóźwiak, Metoda oceny niezawodności sieci komputerowych o strukturach rozgałęzionych, *rozprawa doktorska*, Politechnika Wrocławska, 1979.

- [Jóźwiak92] I. J. Jóźwiak, The reliability and functional model of computer network with branched structure, *Microelectron. Reliab.*, 1992 (32), 345-349.
- [Karger95a] D. R. Karger, A random sampling in graph optimization problems, *Ph.D thesis*, Stanford University, 1995.
- [Karger95b] D. R. Karger, A randomized fully polynomial time approximation scheme for the all terminal network reliability problem, *Proc. Symp. Theory Comput., STOC '95*, Las Vegas 1995, 11-17.
- [Karger95c] D. R. Karger, C. Stein, A new approach to minimum cut problem, *Journal of the ACM* Vol. 43, No. 4 (July 1996), 601-640.
- [Karger97] D. R. Karger, Implementing a fully polynomial time approximation scheme for the all terminal network reliability problem, *Proc. of the Eight Annual ACM-SIAM Symposium on Discrete Algorithms SODA 1997*, New Orleans 1997, 334-343.
- [Karp83] R. M. Karp, M. G. Luby, A new Monte Carlo method for estimating the failure probability of an n-component system, Computer Science Division, University of California, Berkeley, 1983.
- [Karp89] R. M. Karp, M. G. Luby, N. Madras, Monte Carlo approximation algorithms for enumeration problems, *Journal of Algorithms* 1989, 429-448.
- [Kasprzak97] A. Kasprzak, *Rozległe sieci komputerowe z komutacją pakietów*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 1997.
- [Katona66] G. Katona, A theorem of finite sets, *Theory of Graphs* (P. Erds, G. Katona, Eds.) Academic Press, New York 1966, 209-214.
- [Kelmans81] A. K. Kelmans, On graphs with randomly deleted edges, *Acta Math. Acad. Sci., Hungary* 1981 (37), 77-88.
- [Kershenbaum73] A. Kershenbaum, R. M. Van Slyke, Recursive Analysis of Network Reliability, *Networks* 1973(3), 81-94.
- [Khadiri92a] M. E. Khadiri, G. Rubino, A Monte Carlo Method Based on Antithetic Variates for Network Reliability Computations, *Technical Report 826*, IRISA, Campus de Beaulieu, 35042 Rennes, France 1992.
- [Kruskal63] J. B. Kruskal, The number of simplices in a complex, *Mathematical Optimization Techniques* (R. Bellman, Ed.), UC Press, Berkeley 1963, 251-278.

- [Kucera87] L. Kucera, Canonical labeling of regular graphs in linear average time, *28th IEEE Annual Symposium on Foundations of Computer Science*, 1987, 271-279.
- [Kulikowski86] J. L. Kulikowski, *Zarys teorii grafów*, PWN 1986.
- [Kumamoto77] H. Kumamoto, K. Tanaka, K. Inoue, Efficient Evaluation of System Reliability by Monte Carlo Method, *IEEE Trans. Reliability* 1977 (R-26), 311-315.
- [Kumamoto80] H. Kumamoto, K. Tanaka, K. Inoue, E. J. Henley, Dagger-Sampling Monte Carlo for system unavailability evaluation, *IEEE Trans. Reliability* 1980 (R-29), 122-125.
- [Lai94] M. K. F. Lai, Polygon-to-chain reductions work for networks with imperfect vertices, *Microelectron. Reliab.* 1994(34), 267-274.
- [Lee55] C. Y. Lee, Analysis of switching networks, *Bell Syst. Tech. J.* 34(1955), 1287-1315.
- [Li84] V. O. K. Li, J. A. Silvester, Performance analysis of networks with unreliable components, *IEEE Trans. Commun.* 1984(COM-32), 1105-1110.
- [Lomonosov71] M. V. Lomonosov, V. P. Polesskii, An upper bound for the reliability of information networks, *Prob. Inf. Trans.* 1971, 337-339.
- [Lomonosov72] M. V. Lomonosov, V. P. Polesskii, Lower bound of network reliability, *Prob. Inf. Trans.* 1972, 118-123.
- [Luks82] E. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, *Journal of Computer and System Sciences* 1982(25), s.42-65.
- [Madeyski98a] L. Madeyski, Z. Mazur, Nowy algorytm do szybkiego obliczania niezawodności sieci, *Zagadnienia Eksploatacji Maszyn* 1998 nr 3, 391-404.
- [Madeyski98b] L. Madeyski, Z. Mazur, Wykrywanie równoważnych podproblemów w algorytmie faktoryzacji do obliczania niezawodności  $K$ -terminali, *Zeszyty Naukowe Politechniki Częstochowskiej* 1998, 73-81.
- [Madeyski99a] L. Madeyski, Z. Mazur, Zmodyfikowana technika programowania dynamicznego, *ProDialog* 1999 nr 9 (przyjęte do druku).
- [Madeyski99b] L. Madeyski, Z. Mazur, Pesymistyczna złożoność obliczeniowa algorytmu faktoryzacji Fact, *ProDialog* 1999 nr 9 (przyjęte do druku).
- [Mathon79] R. Mathon, A note on the graph isomorphism counting problem, *Information Processing Letters* 1979, 131-132.



- [McKay81] B. McKay, Practical graph isomorphism, *Congressum Numerantium* 1981(30), 45-87.
- [McKay98] B. McKay, *Nauty Users Guide*, Computer Science Department, Australian National University, dostępne przez Internet <http://cs.anu.edu.au/people/bdm/>.
- [Mine59] H. Mine, Reliability of physical systems, *IEEE Trans. Circuit Theory* 1959(6), 138-151.
- [Miyazaki97] T. Miyazaki, The Complexity of McKay's Canonical Labeling Algorithm, *DIMACS* 1997(28), 239-256.
- [Moore56] Moore, Shannon, Reliable circuits using less reliable relays, *J. Franklin Inst.* 1956(262), 191-208, 281-297.
- [Moskowitz58] F. Moskowitz, The analysis of redundancy networks, *AIEE Trans. Commun. Electron.* 1958 (39), 627-632.
- [Motwani92] R. Motwani, *Approximation Algorithms*, Książka w przygotowaniu (wersja wstępna: Technical Report No. STAN-CS-92-1435, Department of Computer Science, Stanford University.)
- [Motwani95] R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.
- [Myrvold96] W. Myrvold, *Analysis and synthesis of reliable networks*, Department of Computer Science, University of Victoria, 1996, (tekst dostępny w formie elektronicznej <http://www.csc.uvic.ca/~wendym/>).
- [Nel90] L. D. Nel, C. J. Colbourn, Combining Monte Carlo estimates and bounds for network reliability, *Networks* 1990(20), 277-290.
- [Ng91] T. P. Ng, K-terminal reliability of hierarchical networks, *IEEE Trans. Reliability* 1991 (40), 218-225.
- [Page88] L. B. Page, J. E. Perry, A practical implementation of the factoring theorem for network reliability, *IEEE Trans. Reliability* 1988 (37) Aug, 259-267.
- [Page89] L. B. Page, J. E. Perry, Reliability of directed networks using the factoring theorem, *IEEE Trans. Reliability* 1989 (38) Dec, 556-562.
- [Page94] L. B. Page, J. E. Perry, Reliability polynomials and link importance in networks, *IEEE Trans. Reliability* 1994 (43) Mar, 51-58.



- [Polesskii71] V. P. Polesskii, A lower boundary for the reliability of information networks, *Prob. Inf. Trans.* 1971(7), 165-171.
- [Politof86] T. Politof, A. Satyanarayana, Efficient algorithms for reliability analysis of planar networks – a survey, *IEEE Trans. Reliability* 1986 (R-35), 252-260.
- [Provan86] J. S. Provan, The complexity of reliability computations in planar and acyclic graphs, *SIAM J. Computing* 1986 (15), 694-702.
- [ResendeM86] M. G. C. Resende, A program for reliability evaluation of undirected networks via polygon-to-chain reductions, *IEEE Trans. Reliability* 1986 (R-35) Apr, 24-29.
- [ResendeL88] L. I. P. Resende, Implementation of factoring algorithm for reliability evaluation of undirected networks, *IEEE Trans. Reliability* 1988 (37) Dec, 462-468.
- [Rosenthal77] A. Rosenthal, Computing the reliability of complex networks, *SIAM J. Appl. Math.*, 1977 (32), 384-393.
- [Satyanarayana78] A. Satyanarayana, A. Prabhakar, New topological formula and rapid algorithm for reliability analysis of complex networks, *IEEE Trans. Reliability*, 1978 (R-27), 82-100.
- [Satyanarayana80] A. Satyanarayana, *Multi-terminal network reliability*, Operations Research Center, University of California, Report ORC 80-6, 1980.
- [Satyanarayana81a] A. Satyanarayana, J. N. Hagstrom, New algorithm for the analysis of multiterminal network reliability, *IEEE Trans. Reliability*, 1981 (R-30), 325-334.
- [Satyanarayana81b] A. Satyanarayana, J. N. Hagstrom, Combinatorial properties of directed graphs useful in computing network reliability, *Networks*, 1981 (1), 357-366.
- [Satyanarayana82a] A. Satyanarayana, R. K. Wood, *Polygon-to-chain reductions and network reliability*, Operations Research Center, University of California, Report ORC 82-4, 1982.
- [Satyanarayana82b] A. Satyanarayana, A unified formula for analysis of some network reliability problems, *IEEE Trans. Reliability*, 1982 (R-31), 23-32.
- [Satyanarayana83] A. Satyanarayana, M. K. Chang, Network Reliability and the Factoring Theorem, *Networks*, 1983 (13), 107-120.
- [Satyanarayana85] A. Satyanarayana, R. K. Wood, A linear-time algorithm for computing K-terminal reliability in series-parallel networks, *SIAM J. Computing*, 1985 (14), 818-832.

- [Shier85] D. R. Shier, Iterative algorithms for calculating network reliability, *Graph theory with applications to algorithms and computer science*, Wiley, New York, 1985, 741-752.
- [Shier87] D. R. Shier, D. E. Whited, Algebraic methods applied to network reliability problems, *SIAM J. Alg. Disc. Meth.*, 1987 (8), 251-262.
- [Shier88a] D. R. Shier, Algebraic aspects of computing network reliability, *Applications of discrete mathematics*, SIAM, Philadelphia, 137-147.
- [Shier88b] D. R. Shier, D. E. Whited, Approximating network reliability, *Congressus Numeratium*, 1988 (8), 221-228.
- [Shier89] D. R. Shier, N. Liu, *Bounding the reliability of networks*, Technical Report 89-04, Department of Mathematics, College of William and Mary.
- [Shier91] D. R. Shier, *Network Reliability and Algebraic Structures*, Oxford University Press, New York 1991.
- [Strayer95a] H. Strayer, C.J. Colbourn, *Consecutive Cuts and Paths, and k-Terminal Reliability*, Tech. Rep. 93-23, DIMACS, April 93. Także [w:] *Networks* 1995(25), 165-175.
- [Strayer95b] H. Strayer, C. J. Colbourn, *Bounding Network Reliability via Surpace Duality*, TR, Combinatorics and Optimization, University of Waterloo, 1995.
- [Strayer98] H. Strayer, K.C Wellsch, C.J. Colbourn, F. Glover, Planarization, Dualization and All-Terminal Reliability, *Proceedings of the Eighth Quadrennial International Conference on Graph Theory, Combinatorics, Algorithms and Applications*, accepted.
- [Tanenbaum88] A. S. Tanenbaum, *Sieci Komputerowe*, WNT, Warszawa 1988.
- [Theologou91] O. R. Theologou, J. G. Carlier, Factoring and Reductions for Networks with Imperfect Vertices, *IEEE Trans. Reliability*, 1991 (40), 210-217.
- [Thomas94] M. Thomas, E. W. Zegura, *Generation and analysis of random graphs to model internetworks*, GIT-CC-94/46 College of Computing Georgia Institute of Technology, Atlanta, Georgia.
- [Torrieri94] D. Torrieri, Calculation of Node-Pair Reliability in Large Networks with Unreliable Nodes, *IEEE Trans. Reliability*, 1994 (43), 375-377,382.
- [Tutte66] W. T. Tutte, *Connectivity in graphs*, University of Toronto Press, Toronto 1966.
- [Valiant79a] L. G. Valiant, The complexity of computing the permanent, *SIAM J. Computing*, 1979 (8), 189-201.

- [Valiant79b] L. G. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Computing*, 1979 (8), 410-421.
- [Van Slyke72] R. Van Slyke, H. Frank, Network reliability analysis, *Networks*, 1972(1), 279-290.
- [Wing64] O. Wing, P. Demetriou, Analysis of probabilistic networks, *IEEE Trans. Commun. Technol.* 1964(12), 34-40.
- [Wood82] R. K. Wood, *Polygon-to-chain reductions and extensions for reliability evaluation of undirected networks*, PhD thesis, Dept. of Industrial Engineering and Operations Research, University of California, Berkeley, 1982.
- [Wood85] R. K. Wood, A Factoring Algorithm Using Polygon-to-Chain Reductions for Computing K-Terminal Network Reliability, *Networks*, 1985 (15), 173-190.
- [Wood86] R. K. Wood, Factoring Algorithms for Computing K-Terminal Network Reliability, *IEEE Trans. Reliability*, 1986 (R-35), 269-278.
- [Wood89] R. K. Wood, Triconnected decomposition for computing K-terminal network reliability, *Networks*, 1989 (19), 203-220.
- [Zegura96] E. W. Zegura, K. L. Calvert, S. Bhattacharjee, How to model an internetwork, *Proceedings of IEEE Infocom '96*, San Francisco, CA.
- [Zemel82] E. Zemel, Polynomial algorithms for estimating network reliability, *Networks* 1982 (12), 439-452.

## INDEKS OZNACZEŃ, SKRÓTÓW I STAŁYCH NUMERYCZNYCH

$\forall$  – oznacza „dla wszystkich”

$\exists$  – oznacza „istnieje(istnieją)”

$\lceil \rceil$  – zaokrąglenie do najbliższej liczby całkowitej

$\lfloor \rfloor$  – zaokrąglenie w dół do najbliższej liczby całkowitej, określane niekiedy zgodnie z terminologią angielską podłogą (*floor*)

$\lceil \rceil$  – zaokrąglenie w górę do najbliższej liczby całkowitej, określane niekiedy zgodnie z terminologią angielską sufitem (*ceiling*)

$(x) \bmod (y)$  –  $x$  modulo  $y$

$\hat{X}$  – górne ograniczenie na  $X$

 – wierzchołek grafu ze zbioru  $K$

 – wierzchołek grafu spoza zbioru  $K$

 – wierzchołek grafu, który może lecz nie musi należeć do zbioru  $K$

$N_o$  – liczba formacji o liczności nieparzystej

$N_e$  – liczba formacji o liczności parzystej

$r(G)$  – liczba drzew rozpinających grafu

$D(G_K)$  – dominacja grafu

$\mu(G) = \min_{K: |K|=2} D(G_K)$  – minimalna dominacja

BDO – binarne drzewo obliczeń

$N(G_K)$  – liczba węzłów BDO

$L(G_K)$  – liczba liści BDO

$I(G_K)$  – liczba węzłów wewnętrznych BDO

$e$  – podstawa logarytmów naturalnych ( $e \approx 2,718$ )

## WYKAZ RYSUNKÓW

- Rysunek 1. Faktoryzacja z wykorzystaniem strategii  $S'$  selekcji krawędzi grafu do faktoryzacji..... 56
- Rysunek 2. Fragment binarnego drzewa obliczeń algorytmu faktoryzacji ( $F_0$ ,  $\{R_0, R_1, R_2, R_3\}$ ,  $S'$ ) przy rozwiązywaniu problemu niezawodności wszystkich terminali sieci reprezentowanych przez graf pełny..... 57
- Rysunek 3. Fragment binarnego drzewa obliczeń algorytmu faktoryzacji ( $F_0$ ,  $\{R_0, R_1\}$ ,  $S'$ ) przy rozwiązywaniu problemu niezawodności wszystkich terminali sieci reprezentowanych przez graf pełny..... 58
- Rysunek 4. Fragment binarnego drzewa obliczeń algorytmu faktoryzacji ( $F_0$ ,  $\{R_1\}$ ,  $S'$ ) przy rozwiązywaniu problemu niezawodności wszystkich terminali sieci reprezentowanych przez graf pełny..... 60
- Rysunek 5. Faktoryzacja z wykorzystaniem strategii  $S''$  selekcji krawędzi grafu do faktoryzacji..... 62
- Rysunek 6. Fragment binarnego drzewa obliczeń algorytmu faktoryzacji ( $F_0$ ,  $\{R_1\}$ ,  $S''$ ) przy rozwiązywaniu problemu niezawodności dwóch terminali sieci reprezentowanych przez graf pełny..... 65
- Rysunek 7. Porównanie określonej eksperymentalnie liczby liści binarnego drzewa obliczeń algorytmów *Fact* i *Fact&Cache* w przypadku obliczania niezawodności wszystkich terminali ( $|K|=|V|$ ) sieci reprezentowanych poprzez graf pełny. .... 86
- Rysunek 8. Porównanie określonej eksperymentalnie liczby liści binarnego drzewa obliczeń algorytmów *Fact* i *Fact&Cache* w przypadku obliczania niezawodności dwóch terminali ( $|K|=2$ ) sieci reprezentowanych poprzez graf pełny. .... 87
- Rysunek 9. Porównanie średniej liczby liści BDO algorytmów *Fact* i *Fact&Cache* przy obliczaniu niezawodności wszystkich terminali ( $|K|=|V|$ ) dla grafów losowych  $G(n,b)$  o  $n=10$ ,  $9 \leq b \leq 45$ ..... 88
- Rysunek 10. Porównanie średniej liczby liści BDO algorytmów *Fact* i *Fact&Cache* przy obliczaniu niezawodności wszystkich terminali ( $|K|=|V|$ ) w zależności od rozmiaru rozpatrywanych grafów  $G(n,b)$  przy stałej gęstości  $d=4$ ..... 89
- Rysunek 11. Porównanie średniej liczby liści BDO algorytmów *Fact* i *Fact&Cache* przy obliczaniu niezawodności wszystkich terminali ( $|K|=|V|$ ) w zależności od rozmiaru rozpatrywanych grafów  $G(n,b)$  przy stałej gęstości  $d=6$ ..... 90

Rysunek 12. Porównanie średniej liczby liści BDO algorytmów *Fact* i *Fact&Cache* przy obliczaniu niezawodności wszystkich terminali ( $|K|=|V|$ ) w zależności od rozmiaru rozpatrywanych grafów  $G(n,b)$  przy stałej gęstości  $d=8$ ..... 90

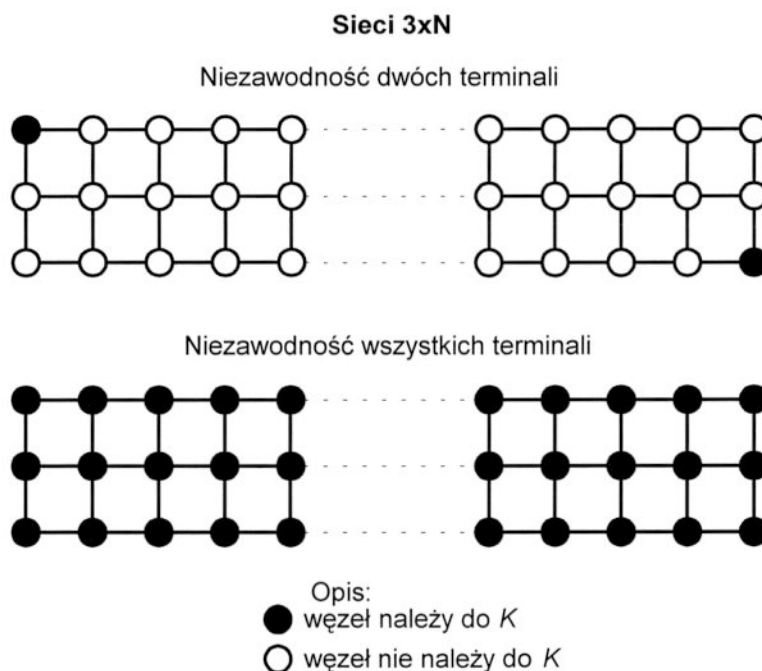
## WYKAZ TABEL

|                                                                                                                                                                              |    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Tabela 1. Redukcje wielokąta do łańcucha ( <i>polygon-to-chain reductions</i> ) zaproponowane przez Satyanarayane i Wooda .....                                              | 39 |
| Tabela 2. Porównanie określonej analitycznie (z wykorzystaniem niezmienników) maksymalnej liczby liści binarnego drzewa obliczeń różnych algorytmów faktoryzacji. ....       | 84 |
| Tabela 3. Porównanie określonej analitycznie (z wykorzystaniem grafów pełnych) maksymalnej liczby liści binarnego drzewa obliczeń różnych algorytmów faktoryzacji. ....      | 85 |
| Tabela 4. Porównanie czasów wykonania w przypadku obliczania niezawodności dwóch terminali ( $ K =2$ ). ....                                                                 | 93 |
| Tabela 5. Porównanie czasów wykonania w przypadku obliczania niezawodności wszystkich terminali ( $ K = V $ ). ....                                                          | 94 |
| Tabela 6. Porównanie liczby liści binarnego drzewa obliczeń w przypadku obliczania niezawodności dwóch terminali ( $ K =2$ ). ....                                           | 96 |
| Tabela 7. Porównanie liczby liści binarnego drzewa obliczeń w przypadku obliczania niezawodności wszystkich terminali ( $ K = V $ ). ....                                    | 96 |
| Tabela 8. Współczynnik trafienia i średni rozmiar podproblemów algorytmu <i>Fact&amp;Cache</i> w przypadku obliczania niezawodności dwóch terminali ( $ K =2$ ). ....        | 97 |
| Tabela 9. Współczynnik trafienia i średni rozmiar podproblemów algorytmu <i>Fact&amp;Cache</i> w przypadku obliczania niezawodności wszystkich terminali ( $ K = V $ ). .... | 98 |

## ZAŁĄCZNIK 1. – ZESTAWIENIE TYPOWYCH SIECI WYKORZYSTYWANYCH DO TESTOWANIA EFEKTYWNOŚCI ALGORYTMÓW

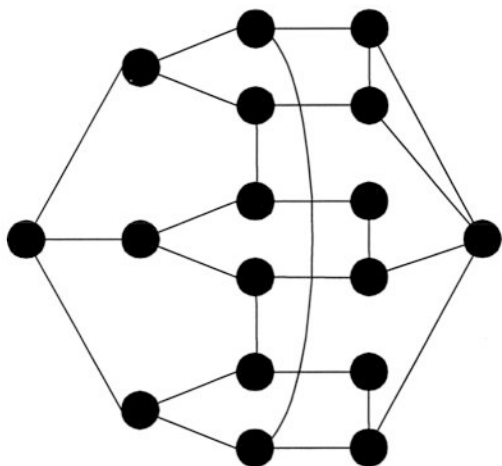
Pewne typowe sieci, zwykle wykorzystywane w literaturze do porównywania efektywności różnych zaproponowanych metod obliczania niezawodności  $K$ -terminali, zostały również wykorzystane w niniejszej pracy. Załącznik ten zawiera zestawienie takich sieci.

1) Sieci typu  $3 \times N$ ,  $4 \times N$  (*street networks*) są często używane do testowania algorytmów [Page88, Theologou91, Carlier96].

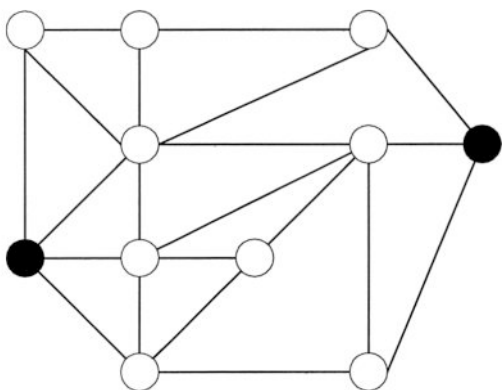


Podobnie są konstruowane sieci typu  $4 \times N$ .

2) BAILEY – sieć użyta do testowania algorytmów, których autorami są między innymi Buzacott [Buzacott80], Ball i Provan [Ball84], Bailey i Kulkarni [Bailey86], Page i Perry [Page88], Carlier i Theologou [Carlier91].



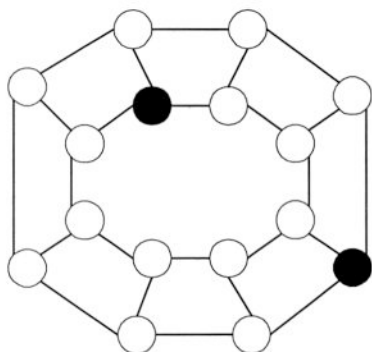
3) FRA732 – sieć użyta do testowania algorytmów, których autorami są między innymi Fratta i Montanari [Fratta73], Abraham [Abraham79], Dotson i Gobein [Dotson79], Debany, Varshney i Hartman [Debany86], Carlier i Theologou [Carlier91, Theologou91] oraz Carlier i Lucet [Carlier96].



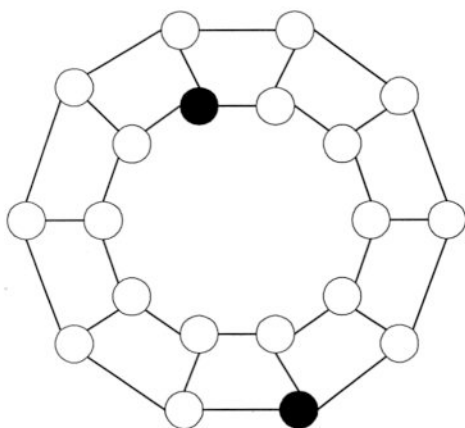


4) CUBE16 i CUBE20 – sieci użyte do testowania algorytmów, których autorami są między innymi Page i Perry [Page88], Carlier i Theologou [Carlier91, Theologou91].

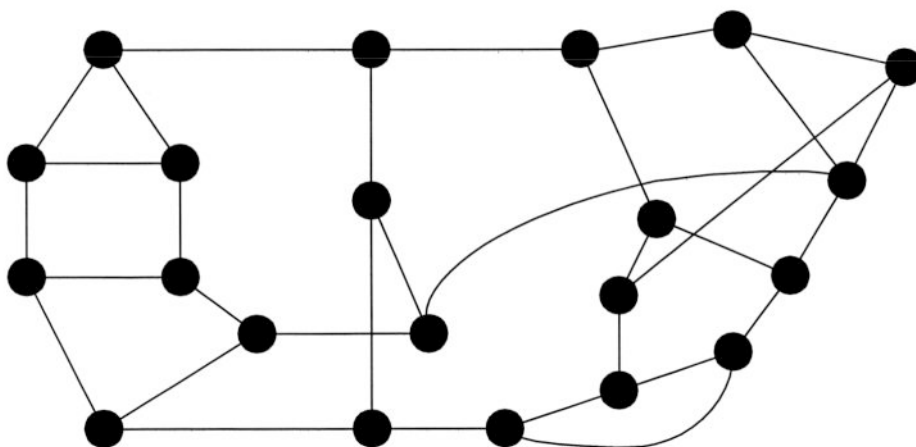
CUBE16:



CUBE20:

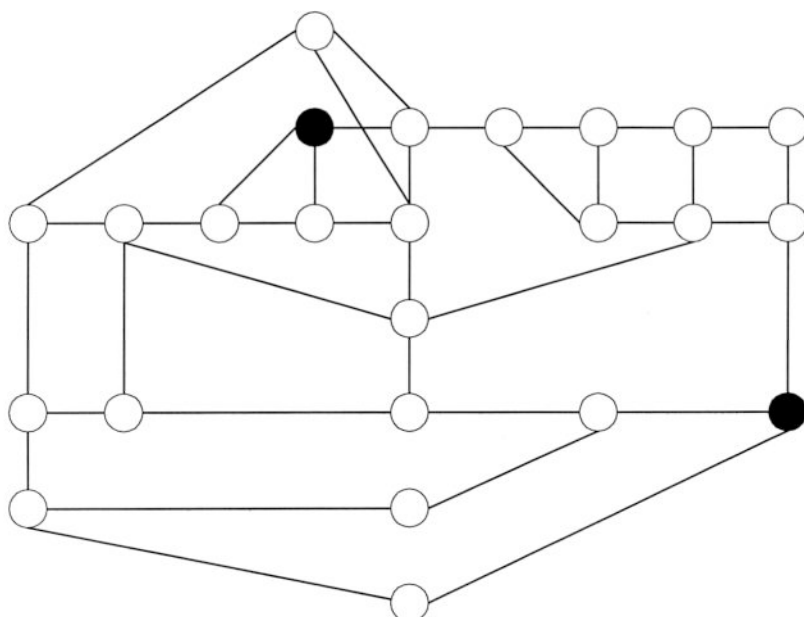


5) Arpanet – sieć użyta do testowania algorytmów, których autorami są między innymi Carlier i Theologou [Carlier91] oraz Carlier i Lucet [Carlier96].

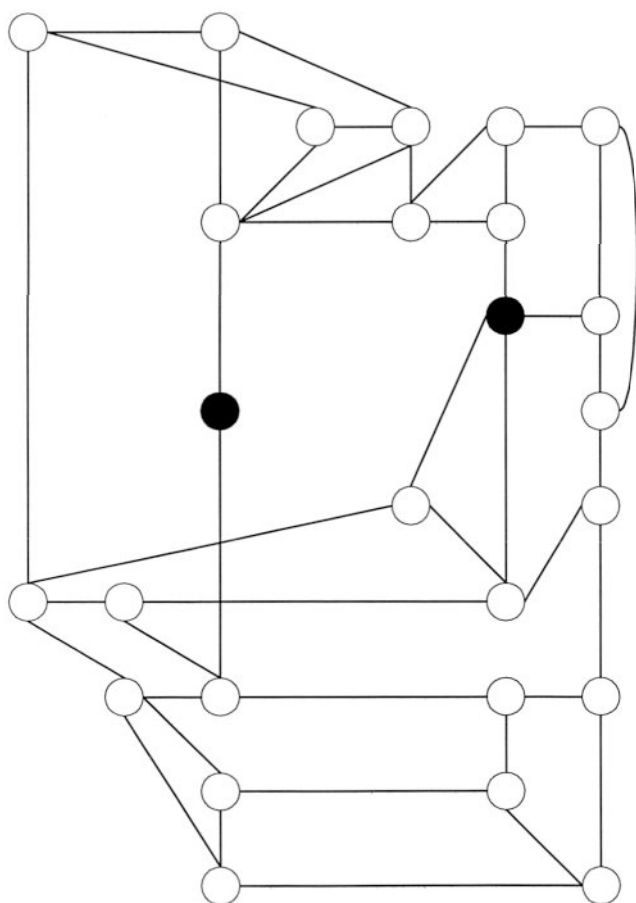


6) SODET1, SODET2, EDF2 – sieci użyte do testowania algorytmów, których autorami są między innymi Carlier i Theologou [Carlier91] oraz Carlier i Lucet [Carlier96].

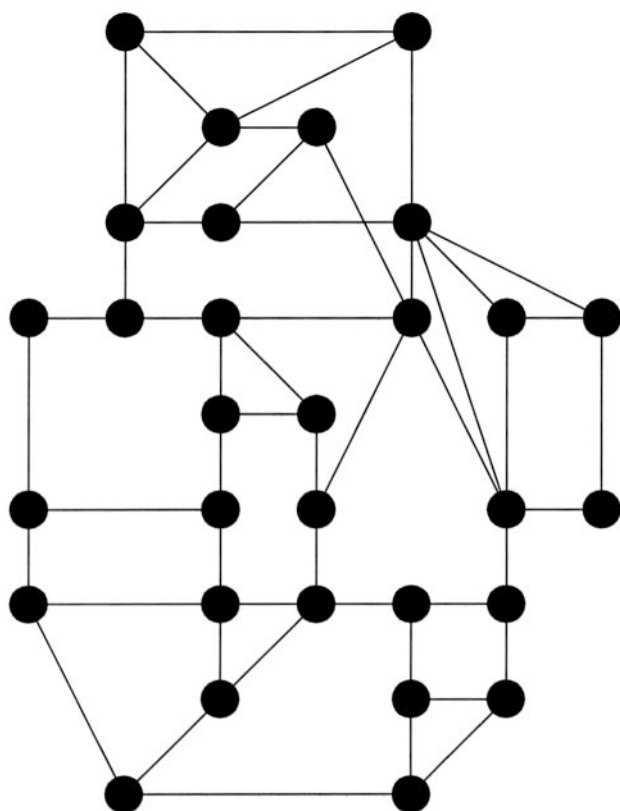
SODET1:



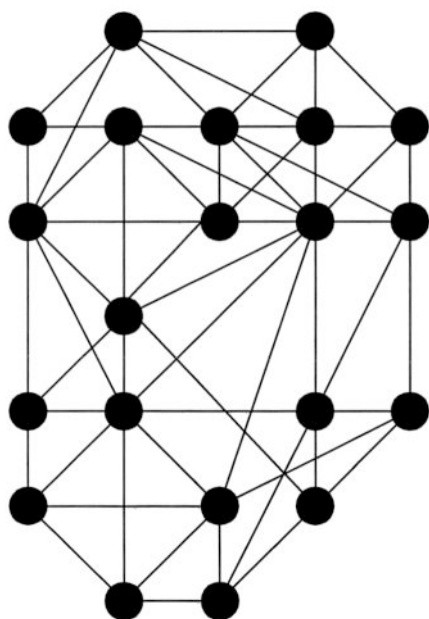
SODET2:



EDF2:



6) sieć PLG – najbardziej skomplikowana spośród sieci testowych (z punktu widzenia analizy niezawodności ze względu na wysokie stopnie wierzchołków grafu), została użyta do testowania algorytmu dekompozycji, którego autorami są Beichelt i Tittmann [Beichelt91a].



Dla sieci PLG, której niezawodności krawędzi wynoszą 0,9, plik z rezultatami zawiera następujące informacje:

Program: FACT&CACHE

Input file name: PLG

Mean distance between vertices 2.047619 v1=2 vn=20

Reliability = 0.97803

Number of calls(recursions) = 5995

Time ticks->secs = 52->2.9

Number of parallel edge reductions = 4923

Number of one-degree vertex reductions = 780

Number of two-degree(serial/deg2) vertex reductions = 2978(0/2978)

Number of times factoring theorem is used = 2997

Number of cache hit/miss = 2218/2668

Cache hit rate = 45.4%

Recursion depth = 31

Average hit network has 24.3 edges and 11.6 vertices

# SKOROWIDZ

## A

algorytm

aproxymacyjny, 25, 29, 31

*Fact*, 3, 4, 6, 16, 17, 54, 55, 63, 68, 71, 72, 74, 75, 78, 80, 81, 82, 83, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 117, 123, 124

*Fact&Cache*, 3, 4, 6, 17, 55, 72, 74, 75, 78, 80, 81, 82, 83, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 103, 104, 105, 106, 107, 109, 110, 123, 124

Floyda, 79

## C

częściowe spamiętywanie, 73, 74, 75, 78, 83, 101

## D

dekompozycja problemu, 3, 21, 33, 35, 41, 55, 71, 72, 73, 75, 77, 78, 101

## F

*Fact*, 3, 4, 6, 16, 17, 54, 55, 63, 68, 71, 72, 74, 75, 78, 80, 81, 82, 83, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 117, 123, 124

*Fact&Cache*, 3, 4, 6, 17, 55, 72, 74, 75, 78, 80, 81, 82, 83, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 103, 104, 105, 106, 107, 109, 110, 123, 124

faktoryzacja, 33, 77, 89

funkcja haszująca, 4, 107

## K

kanoniczne etykietowanie grafów, 4, 104, 105, 106

kodowanie podproblemów, 3, 76, 104

## M

metody konstruowania ograniczeń, 27

metody Monte Carlo, 23, 30, 31

miary

niezawodności, 11, 14, 15, 22, 30, 110

zdolności do prawidłowego działania, 104

model sieci, 9, 10, 16

## P

programowanie dynamiczne, 3, 72, 73, 74, 80, 100, 101, 117

## S

szkielet algorytmu, 35, 72

## W

współczynnik trafienia, 77, 78, 87, 97, 98, 130

## Z

zachowujące niezawodność redukcje grafu, 2, 18, 19, 22, 33, 34, 35, 36, 44, 49, 78, 82, 87, 89, 108

złożoność

czasowa, 4, 18, 26, 31, 45, 46, 48, 49, 50, 52, 55, 63, 64, 67, 68, 81, 82, 83, 86, 105, 106, 107

obliczeniowa, 15, 16, 45, 52, 81, 100, 117

pamięciowa, 26, 52, 53, 68, 69, 70, 71, 75, 81, 83

pesymistyczna, 3, 4, 16, 17, 45, 52, 54, 55, 71, 83, 84, 85, 100, 101, 102, 105, 108

