Lech Madeyski [1], Zygmunt Mazur[2]

Nowy algorytm do szybkiego obliczania niezawodności sieci

1. Wprowadzenie

Analiza niezawodności różnego rodzaju sieci, których łącza ulegają losowym uszkodzeniom jest tematem intensywnych badań. Najczęściej rozważanym matematycznym modelem różnego rodzaju sieci jest graf nieskierowany *G* z wyróżnionym podzbiorem węzłów *K*. Krawędzie grafu ulegają wzajemnie niezależnie losowym uszkodzeniom ze znanym prawdopodobieństwem. Graf ten jest niekiedy nazywany grafem lub siecią stochastyczną [1-3] bądź probabilistyczną [4]. Wierzchołki grafu reprezentują węzły (miejsca) w sieci. W przypadku sieci komputerowej mogą to być poszczególne komputery, podsieci lokalne bądź metropolitalne. Krawędzie grafu reprezentują łącza komunikacyjne, które ulegają uszkodzeniom. Zakłada się, że węzły sieci są całkowicie niezawodne. Każde łącze komunikacyjne umożliwia dwukierunkową komunikację pomiędzy węzłami pod warunkiem, że jest ono sprawne.

Gdy będzie nam zależało na podkreśleniu, że graf jest reprezentacją rzeczywistych sieci, użyjemy terminów węzły i łącza, zamiast wierzchołki i krawędzie. Zdecydowano się również na szereg własnych tłumaczeń z racji braku polskich odpowiedników niektórych terminów.

Niezawodnością sieci jest prawdopodobieństwo, że sieć może realizować swoje funkcje. Formułowane są różne miary oceny niezawodności sieci probabilistycznych. Najczęściej rozważany problem niezawodności *K*-terminali (*K-terminal network reliability problem*) określa prawdopodobieństwo, iż wszystkie węzły znajdujące się w pewnym określonym zbiorze *K* ($2<=|K|<=|V|$) są połączone poprzez ścieżki nie uszkodzonych łączy.

Często również są rozważane przypadki brzegowe:

- problem niezawodności dwóch terminali (*2-terminal network reliability problem*), gdy $|K|=2$;
- problem niezawodności wszystkich terminali (*All-terminal network reliability problem*), gdy $|K|=|V|$.

Notacja

$G=(V,E)$	graf nieskierowany składający się ze zbioru wierzchołków $V=\{v_1, v_2, ...\}$ i zbioru krawędzi $E=\{e_1, e_2, ...\}$;
K	wyróżniony podzbiór wierzchołków grafu $K \subseteq V$;
p_i	niezawodność krawędzi e_i grafu G;

[1] Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania, Politechnika Wrocławska, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław. E-mail: madeyski@ci.pwr.wroc.pl
[2] Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania, Politechnika Wrocławska, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław. E-mail: mazur@ci.pwr.wroc.pl

G_K	G z wyróżnionym zbiorem wierzchołków K;
$R(G_K)$	niezawodność K-terminali grafu G (prawdopodobieństwo, że wszystkie wierzchołki ze zbioru K są połączone);
$R(G_K,*e_i)$	niezawodność K-terminali grafu G pod warunkiem, że krawędź e_i jest sprawna (dlatego krawędź ta jest usunięta a węzły incydentne z tą krawędzią połączone);
$R(G_K-e_i)$	niezawodność K-terminali grafu G pod warunkiem, że krawędź e_i jest uszkodzona (dlatego krawędź ta jest usunięta);
Fact	algorytm faktoryzacji z zachowującymi niezawodność redukcjami (równoległą, szeregową, pierwszego i drugiego stopnia);
Fact&Cache	algorytm faktoryzacji z zachowującymi niezawodność redukcjami (równoległą, szeregową, pierwszego i drugiego stopnia) wykorzystujący zaproponowaną technikę *przechowywania części rozwiązań*.

Ponieważ wykazano [5-7], że problem niezawodności sieci K-terminali należy do klasy problemów NP-trudnych, więc mało prawdopodobne jest istnienie wielomianowego algorytmu do rozwiązania tego problemu.

Najczęściej stosowanym algorytmem do wyznaczania niezawodności K-terminali jest algorytm faktoryzacji [8, 9]. Niestety mała efektywność tego algorytmu ogranicza jego zastosowanie do stosunkowo niewielkich sieci. Zastosowanie zachowujących niezawodność redukcji, poprawia efektywność algorytmu. Jest ona jednak wciąż niewystarczająca za względu na konieczność analizy coraz bardziej skomplikowanych sieci. W pracy przedstawiono nowy algorytm wykorzystujący zaproponowaną przez autorów technikę *przechowywania części rozwiązań*. Efektywność nowo powstałego algorytmu *Fact&Cache* pozwoli stosować go do obliczania niezawodności znacznie większych sieci niż miało to miejsce w przypadku standardowego algorytmu faktoryzacji. *Fact&Cache* wydaje się być konkurencyjny nie tylko w stosunku do innych algorytmów faktoryzacji, nawet tych wyposażonych w mechanizmy redukcji wielokątów do łańcuchów [10-12] czy trzyspójnej dekompozycji [13], ale i algorytmów wykorzystujących metodę dekompozycji [2, 3, 14]. W rozdziale 5 porównano na przykładach osiągi programów wykorzystujących algorytm faktoryzacji, algorytm dekompozycji (jak dotąd najefektywniejszy) i zaproponowany przez nas algorytm *Fact&Cache*.

2. Algorytm faktoryzacji

Metody obliczania niezawodności K-terminali wykorzystujące formułę zawierania-wykluczania czy sumę rozłącznych iloczynów bazują na koncepcji zbiorów stanów, w których wierzchołki z wyróżnionego zbioru K grafu G_K są połączone. Ponieważ zbiory te są określone poprzez kolekcje krawędzi, mamy do czynienia ze zdarzeniami złożonymi.

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. Stąd niezawodność K-terminali można wyrazić w postaci prostej formuły niezawodności warunkowej, jako:

$$R(G_K) = p_i R(G_K | e_i \ \ funkcjonuje) + (1 - p_i) R(G_K | e_i \ nie \ \ funkcjonuje). \quad (1)$$

Twierdzenie faktoryzacji [15] jest topologiczną interpretacją tej formuły dla grafów nieskierowanych:

$$R(G_K) = p_i R(G_{K'} * e_i) + (1 - p_i) R(G_K - e_i), \quad (2)$$

gdzie:

e_i – dowolna krawędź grafu G_K;

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

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

$G_K - e_i = (V, E - e_i).$

Wbudowanie mechanizmu zachowujących niezawodność redukcji grafu [9] zwiększa efektywność algorytmu. Redukcje grafu wymagają tylko wielomianowego czasu wykonania zmniejszając wykładniczą przestrzeń stanów problemu i rozmiar grafu. Redukcje zmieniają graf G_K topologicznie i probabilistycznie. W ich wyniku otrzymuje się zredukowany graf $G'_{K'}$ taki, że

$$R(G_K) = \Omega R(G'_{K'}), \quad (3)$$

gdzie Ω jest stałą uzyskaną w wyniku redukcji oryginalnego grafu G_K.

Niezawodność $R(G_K)$ może być obliczona, dla dowolnego grafu, poprzez rekurencyjne wywoływanie formuły (2) i ewentualne dokonywanie zachowujących niezawodność redukcji. Grafy rozpatrywane w wyniku rekurencyjnego wywoływania formuły faktoryzacji są coraz prostsze (mają o jeden wierzchołek i jedną krawędź mniej w przypadku grafu $G_{K'} * e_i$ i o jedną krawędź mniej w przypadku grafu $G_K - e_i$). Procedura ta odpowiada konstruowaniu binarnego drzewa obliczeń. Otrzymywane w ten sposób grafy mogą również podlegać zachowującym niezawodność redukcjom. W końcu grafy są redukowane do prostych struktur, dla których niezawodność można bardzo prosto obliczyć lub wierzchołki należące do zbioru K są rozłączone (niezawodność wynosi zero). W ten sposób można obliczyć niezawodność dowolnej sieci.

3. Technika *przechowywania części rozwiązań* na tle innych zaawansowanych technik projektowania algorytmów

Techniki czy metody projektowania efektywnych algorytmów są klasyczną, szeroko omawianą dziedziną informatyki [16-20]. Zaproponowaną przez nas technikę będziemy opisowo określać jako

technikę *przechowywania części rozwiązań*, albo jednym słowem *cache*[3] ze względu na fakt, iż jest ono bliskie znaczeniowo prezentowanej technice. Proponowana technika jest modyfikacją techniki określanej w języku angielskim terminem *memoization* [16], którą będziemy określać jako technikę *przechowywania rozwiązań*. Obie są odmianami znanej techniki *programowania dynamicznego*.

Metoda *programowania dynamicznego* pozwala uniknąć wielokrotnego rozwiązywania tych samych podproblemów. Tworzona jest tablica, w której zapamiętywane są rozwiązania najmniejszych podproblemów (czasami określanych mianem podproblemów elementarnych), a następnie obliczonych na ich podstawie podproblemów większych rozmiarów. Proces ten jest kontynuowany aż do obliczenia rozwiązania problemu pierwotnego *P*, przy czym raz znalezione rozwiązanie podproblemu zostaje zapamiętane i może być wykorzystywane bez konieczności ponownego obliczania.

Nie zawsze jednak natura problemu pierwotnego pozwala zastosować klasyczną metodę *programowania dynamicznego* wykorzystującą strategię wstępującą. Czasami trudno jest określić na wstępie podproblemy elementarne i ich rozwiązania, a następnie sposób, formułę otrzymania na ich podstawie rozwiązań podproblemów większych rozmiarów, niezbędnych, aby otrzymać rozwiązanie problemu pierwotnego *P*. W takiej sytuacji wygodnie jest zastosować technikę *przechowywania rozwiązań*. Wykorzystuje ona zaletę *programowania dynamicznego*, która polega na jednokrotnym obliczaniu tych samych podproblemów. Jest jednak implementowana z wykorzystaniem strategii zstępującej. Algorytm wykorzystujący technikę *przechowywania rozwiązań* ma formę algorytmu rekurencyjnego z dodanymi funkcjami zapamiętywania parametrów podproblemów wraz z ich rozwiązaniami[4] oraz przeszukiwania zarejestrowanych podproblemów w celu wielokrotnego wykorzystania raz obliczonego rozwiązania.

Zwykle algorytm wykorzystujący iteracyjną technikę *programowania dynamicznego* ze strategią wstępującą charakteryzuje się podobną złożonością czasową jak algorytm stosujący technikę *przechowywania rozwiązań* (różnica o stały czynnik, na korzyść *programowania dynamicznego*, ze względu na „koszt” wywołań rekurencyjnych i powrotów). Jeżeli jednak rozwiązanie problemu pierwotnego wymaga rozwiązania jedynie niektórych podproblemów to algorytm stosujący technikę *przechowywania rozwiązań* może charakteryzować się mniejszą złożonością czasową jak i pamięciową. Rozwiązywane będą bowiem tylko te podproblemy, których rozwiązanie jest niezbędne.

Gdy złożoność pamięciowa algorytmu ze względu na liczbę (rozmiar) możliwych podproblemów jest zbyt duża, można zastosować zaproponowaną przez autorów technikę *przechowywania części rozwiązań*. Wykorzystuje ona ideę ograniczania ilości przechowywanych rozwiązań. Charakteryzuje

[3] Słowo *cache* nie było dotąd używane w kontekście technik projektowania algorytmów. Pamięć *cache* występuje na różnych poziomach architektury komputerów. Są w niej przechowywane często używane dane, co znacznie przyspiesza dokonywane operacje. W przypadku opisywanej techniki w pamięci *cache* będą przechowywane rozwiązania podproblemów i ewentualnie ich parametry.

[4] Jeżeli wszystkie możliwe parametry podproblemów są znane i jest określona relacja pomiędzy pozycjami tabeli a podproblemami, to wystarczy zapamiętywać jedynie rozwiązania.

się więc mniejszą złożonością pamięciową niż klasyczna technika *programowania dynamicznego* czy technika *przechowywania rozwiązań*. Modyfikacja w stosunku do techniki *przechowywania rozwiązań* polega na tym, że nie są przechowywane wszystkie rozwiązania, a jedynie te, które 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*. Ilość przechowywanych rozwiązań zależy od rozmiaru pamięci *cache*. Stosowana strategia ograniczania ilości przechowywanych rozwiązań zależy od charakteru rozwiązywanego problemu. W przypadku gdy istnieją różne strategie dekompozycji problemu pierwotnego o efektywności techniki *przechowywania części rozwiązań* może decydować wybrana strategia dekompozycji, która powinna zapewniać maksymalne wykorzystanie już zarejestrowanych rozwiązań podproblemów.

Zaproponowana technika może znaleźć zastosowanie w przypadkach, gdy złożoność pamięciowa rozwiązań wykorzystujących *programowanie dynamiczne* lub technikę *przechowywania rozwiązań* byłaby zbyt duża. Algorytm faktoryzacji [8, 9] do obliczania niezawodności *K-terminali* może zostać tak zmodyfikowany, aby wykorzystywał technikę *przechowywania części rozwiązań*.

4. Wykorzystanie techniki *przechowywania części rozwiązań* w algorytmie faktoryzacji

Badając standardowy algorytm faktoryzacji można zauważyć, iż w wielu węzłach drzewa obliczeń te same podproblemy (sieci powstałe w wyniku stosowania faktoryzacji i redukcji) rozwiązywane są wiele razy. Zamiast wielokrotnie rozwiązywać ten sam podproblem (obliczać niezawodność tej samej sieci), jak ma to miejsce w przypadku standardowego algorytmu faktoryzacji, znacznie efektywniej byłoby użyć wcześniej obliczony i zapamiętany rezultat obliczeń. Ten bardzo istotny fakt nie został jak dotąd praktycznie wykorzystany.

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 *przechowywania części rozwiązań*. Efektywność stosowanej w algorytmie techniki *przechowywania części rozwiązań* zależy w dużym stopniu od strategii dekompozycji problemu i skuteczności wykrywania równoważności sieci.

4.1. Strategia dekompozycji problemu

Strategia dekompozycji problemu, czyli wybierania krawędzi grafu do faktoryzacji, wykorzystuje priorytety przyporządkowywane krawędziom grafu. Faktoryzacji podlega zawsze ta krawędź grafu, która ma najwyższy priorytet. Testowano różne heurystyki przyporządkowywania priorytetów krawędziom grafu i następująca wydaje się zapewniać najlepsze rezultaty.

W fazie wstępnej dokonywana jest analiza grafu, w wyniku której wszystkim krawędziom grafu zostają przyporządkowane priorytety. 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 Floyd'a. Spośród wierzchołków grafu wybierane są wszystkie możliwe pary węzłów (v_1, v_n), które są najbardziej od siebie oddalone. Jeśli jest tylko jedna taka para to węzły v_1 i v_n są natychmiast określone. Jeżeli jest ich więcej, to wybierana jest ta para, której składniki v_1 i v_n występują najczęściej w wybranych parach wierzchołków (v_1, v_n).

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.

Następnie przeprowadzana jest korekcja współczynników położenia o wpływ wierzchołków przyległych.

P_k - zbiór wierzchołków przyległych do wierzchołka k

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

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. Jeżeli krawędź e_i jest incydentna do wierzchołków o etykietach v_a i v_b, to priorytet przypisany tej krawędzi jest tym większy im wartość wyrażenia $min(v_a,v_b)*|V|+max(v_a,v_b)$ jest większa.

Pozostaje jeszcze problem przyporządkowania priorytetów krawędziom powstałym w wyniku redukcji równoległej i redukcji stopnia drugiego. W obu przypadkach krawędź, która powstaje w miejsce dwóch redukowanych krawędzi e_a i e_b otrzymuje priorytet max(priorytet(e_a), priorytet(e_b)).

Zaproponowana strategia dekompozycji, choć trudna do analitycznej oceny, jest intuicyjnie uzasadniona. Sprzyja ona występowaniu tych samych podproblemów w binarnym drzewie obliczeń, co zostało eksperymentalnie potwierdzone w rozdziale 5.

4.2. Równoważność sieci

Aby móc wykorzystywać raz obliczone rozwiązania podproblemów konieczne jest zaimplementowanie mechanizmu umożliwiającego wykrywanie podproblemów (sieci) równoważnych w sensie niezawodności *K*-terminali.

Definicja 1

Dwie sieci są równoważne w sensie niezawodności *K*-terminali, 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 1

Jeżeli sieci G_K i $H_{K'}$ są równoważne w sensie niezawodności *K*-terminali to ich niezawodności *K*-terminali są równe $R(G_K)=R(H_{K'})$.

Dla danych dwóch sieci reprezentowanych przez grafy G_K i $H_{K'}$ chcemy określić, czy są one równoważne w sensie niezawodności *K*-terminali. Jednym ze sposobów rozwiązania tego problemu jest skonstruowanie kodu dla każdej sieci. Dwie sieci byłyby równoważne wtedy i tylko wtedy, gdyby ich kody były 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. Efektywne testowanie izomorficzności grafów jest jednak wciąż tematem intensywnych badań.

Idealny kod do badania izomorficzności grafów posiadałby następujące własności:

Własność 2

Jeżeli kod odpowiedzialny za testowanie izomorficzności grafów jest taki sam dla grafów *G* i *H* to są one wzajemnie izomorficzne;

Własność 3

Jeżeli graf *G* jest izomorficzny z *H* to ich kody odpowiedzialne za testowanie izomorficzności są sobie równe.

Jednym z możliwych rozwiązań jest testowanie izomorficzności grafów na podstawie ich macierzy 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 3 raczej rzadko byłaby spełniona. Rozwiązaniem problemu jest takie przypisanie etykiet

poszczególnym wierzchołkom, które dawałoby maksymalną albo minimalną wartość kodu. Niestety przebadanie wszystkich możliwych permutacji węzłów jest zbyt kosztowne czasowo. 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. W przypadku grafów, których niezbyt duży procent wierzchołków ma ten sam stopień, zysk może być godny uwagi. W ogólnym przypadku nie jest jednak znana metoda efektywnego testowania izomorficzności grafów. Dlatego konieczne jest pewne osłabienie wymogu co do wykrywania izomorficzności grafów.

O ile własność 2 musi być zawsze spełniona, aby nie zniekształcić ostatecznego wyniku to wystarczy, aby własność 3 była spełniona możliwie często. Jedyną zaś konsekwencją tego jest możliwość niewykrycia izomorficzności grafów, czyli powtórnego obliczania niezawodności niektórych sieci. Praktycznie jednak efektywność algorytmu może się zwiększyć. Osłabienie warunku co do spełnialności własności 3 umożliwia bowiem znaczne przyspieszenie generowania kodów grafów.

Kod sieci składa się z następujących części:

- część odpowiedzialna za możliwie częste wykrywanie izomorficzności grafów (spełnia własność 2 ale nie zawsze własność 3);
- część identyfikująca wyróżniony zbiór węzłów *K*;
- część zawierająca ilość łączy i ich niezawodności;

Ponieważ w wyniku stosowania faktoryzacji i redukcji pewne wierzchołki występujące w grafie wejściowym zostały usunięte, etykiety wierzchołków przetwarzanego grafu nie tworzą ciągłego zbioru liczb 1, 2, ..., |*V*|. Aby wygenerować część kodu grafu, która odpowiada za wykrywanie izomorficzności, 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. W algorytmie zastosowano strategię zachowania pierwotnego porządku etykiet. Zmieniane są jedynie etykiety większe od najmniejszej nieużywanej etykiety grafu. Strategia ta zapewnia, że wygenerowany kod możliwie często spełnia własność 3, co daje wysoki współczynnik trafienia (*hit rate*) w pamięci *cache*. Dzięki temu 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 *przechowywania części rozwiązań* może być mierzona wartością współczynnika trafienia i średnim rozmiarem podproblemów odczytanych z pamięci *cache*.

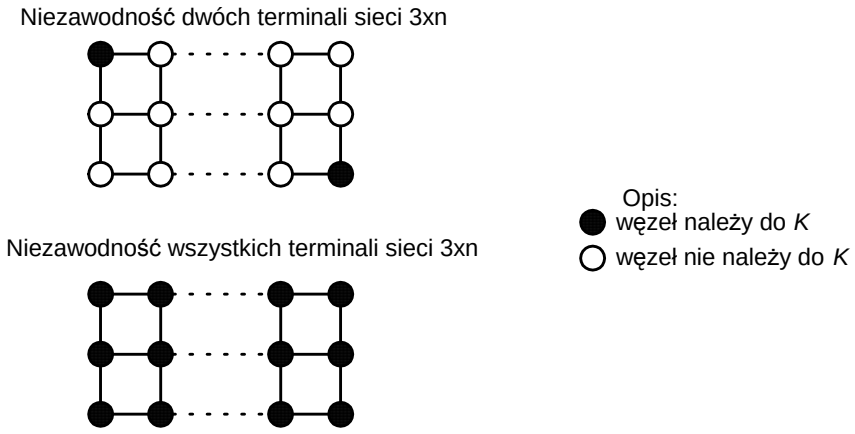
Schemat generowania kodu sieci spełniający te założenia został zaprojektowany i zaimplementowany w algorytmie. Oczywiście można sobie wyobrazić inne schematy kodowania sieci.

5. Porównanie algorytmów obliczania niezawodności sieci

Zgodnie z notacją podaną w rozdziale 1 *Fact* oznacza standardowy algorytm faktoryzacji (wg Page i Perry [8])[5] z zachowującymi niezawodność redukcjami grafu (równoległą, szeregową, pierwszego i drugiego stopnia). Natomiast *Fact&Cache* oznacza zaproponowany przez autorów algorytm faktoryzacji z zachowującymi niezawodność redukcjami grafu, wykorzystujący opisaną w poprzednim rozdziale technikę *przechowywania części rozwiązań* z wykrywaniem sieci równoważnych i zaproponowaną heurystyczną strategią dekompozycji problemu. Zastosowana w algorytmie *Fact&Cache* technika *przechowywania części rozwiązań* używa strategii przechowywania ostatnio zarejestrowanych rozwiązań. Jest ustalona maksymalna liczba przechowywanych rozwiązań i związanych z nimi parametrów podproblemów o określonym rozmiarze a dokładnie liczbie łączy sieci. Ponieważ binarnym drzewie obliczeń podproblemów o małym rozmiarze jest najwięcej, to w efekcie rozwiązania i parametry podproblemów o większym rozmiarze są na ogół dłużej przechowywane niż rozwiązania i parametry podproblemów o mniejszym rozmiarze.

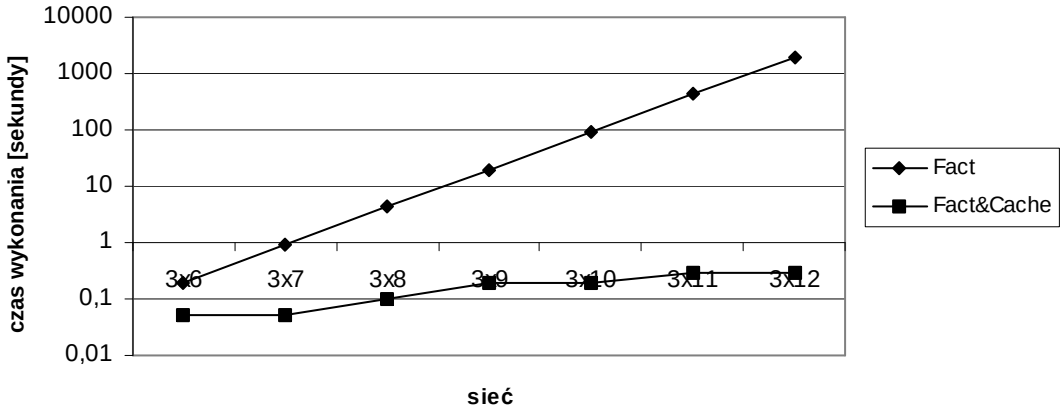
Zarówno algorytm *Fact* jak i *Fact&Cache* zostały praktycznie zaimplementowane w języku C++. Dokonano porównania efektywności algorytmów w przypadku rozwiązywania problemu niezawodności dwóch terminali jak i niezawodności wszystkich terminali dla często rozpatrywanych w literaturze sieci 3xn (gdzie n jest liczbą naturalną) przedstawionych na rysunku 1.

Rysunek 1 Rozpatrywane sieci 3xn i problemy niezawodności

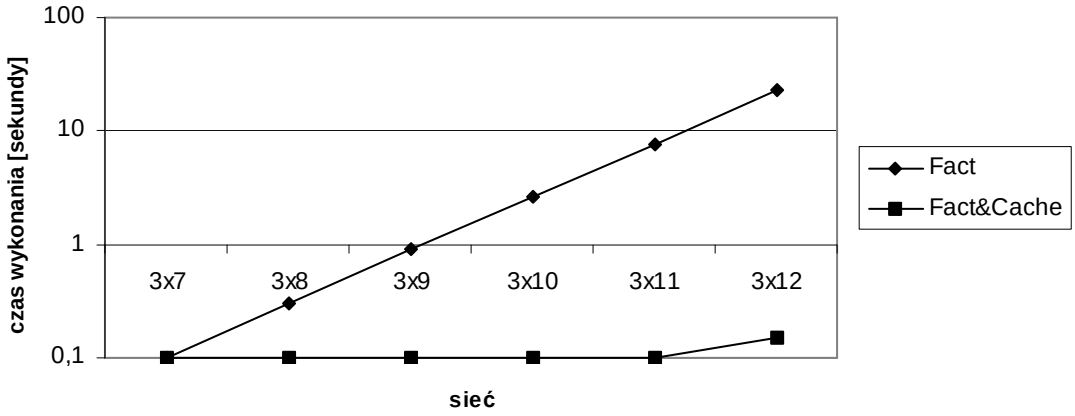


Pomiarów dokonano na komputerze PC Pentium 200 MHz. Rysunki 2 i 3 prezentują czasy wykonania programów wykorzystujących algorytmy *Fact* i *Fact&Cache*. Ze względu na olbrzymie różnice w efektywności obu algorytmów wykonano je w skali logarytmicznej.

[5] Strategia wyboru krawędzi do faktoringu została zmieniona z losowej na stałą (zawsze wybierana jest ostatnia krawędź z listy krawędzi grafu), co daje na ogół lepsze rezultaty.



Rysunek 2 Czas obliczania niezawodności dwóch terminali w przypadku sieci 3xn



Rysunek 3 Czas obliczania niezawodności wszystkich terminali w przypadku sieci 3xn

O efektywności zaproponowanego algorytmu *Fact&Cache* świadczy także rosnący wraz z rozmiarem badanych sieci współczynnik trafienia i średni rozmiar podproblemów odczytanych z pamięci *cache*. Dla badanych sieci zjawisko to występuje zarówno przy rozwiązywaniu problemu niezawodności dwóch terminali jak i niezawodności wszystkich terminali. Dokładne dane przedstawiają tabele 1 i 2.

sieć	współczynnik trafienia [%]	średni rozmiar sieci, których niezawodność odczytano z pamięci *cache*	
		liczba łączy	liczba węzłów
3x6	40,8	9,6	7,6
3x7	43,5	12	9,1
3x8	45	14,5	10,6
3x9	46	17,1	12,1
3x10	46,7	19,6	13,6
3x11	47,2	22,1	15,1
3x12	47,6	24,6	16,6

Tabela 1 Współczynnik trafienia i średni rozmiar podproblemów odczytanych z pamięci *cache* przy obliczaniu niezawodności dwóch terminali sieci 3xn

sieć	Współczynnik trafienia [%]	średni rozmiar sieci, których niezawodność odczytano z pamięci *cache*	
		liczba łączy	liczba węzłów
3x6	34,6	8,9	6,7
3x7	37,5	10,2	7,4
3x8	41	12,5	8,8
3x9	43,8	15,9	10,8
3x10	44,9	18,4	12,3
3x11	45,9	20,9	13,8
3x12	46,4	23,3	15,3

Tabela 2 Współczynnik trafienia i średni rozmiar podproblemów odczytanych z pamięci *cache* przy obliczaniu niezawodności wszystkich terminali sieci 3xn

Ponadto porównano na przykładach zaczerpniętych z [14] oraz [2] czasy wykonania programów wykorzystujących algorytm *Fact&Cache* oraz jak dotąd najefektywniejszą metodę dekompozycji.

sieć	T1[sek]	T2[sek]	T3[sek]
3x10	<0,1	1,7	-
4x7	0,3	4,5	-
Arpanet	<0,1	1,82	-
Sodet1	0,3	7,36	-
EDF2	1,2	12,8	-
PLG	3	-	120

Tabela 3 Porównanie czasów obliczania niezawodności wszystkich terminali przez programy wykorzystujące algorytm *Fact&Cache* i algorytmy dekompozycji

Uwaga: T1 jest czasem wykonania programu wykorzystującego zaproponowany przez autorów algorytm *Fact&Cache* na komputerze PC Pentium 200 MHz. T2 jest czasem wykonania programu wykorzystującego przedstawioną przez Carliera i Lucet w [14] metodę dekompozycji zwaną również metodą brzegową (boundary method) na komputerze PC 486. T3 jest czasem wykonania programu wykorzystującego przedstawioną przez Beichelta i Tittmana w [2] metodę dekompozycji stosowaną wraz z algorytmem faktoryzacji Wooda [9] na komputerze A7150 firmy Robotron.

We wszystkich przypadkach obliczana jest niezawodność wszystkich terminali. Interesującym byłoby porównanie efektywności algorytmów także w przypadku obliczania niezawodności dwóch terminali. Niemniej otrzymane rezultaty są obiecujące, a program realizujący prezentowany algorytm *Fact&Cache* będzie podlegał dalszym modyfikacjom między innymi w zakresie strategii

rejestrowania rozwiązań i związanych z nimi parametrów podproblemów oraz realizacji przeszukiwania rozwiązań.

6. Podsumowanie

W pracy przedstawiono nowy, bardzo efektywny algorytm *Fact&Cache* do obliczania niezawodności sieci *K*-terminali. Wykorzystuje on zaproponowaną przez autorów technikę *przechowywania części rozwiązań* z wykrywaniem sieci równoważnych i heurystyczną strategią dekompozycji problemu. Algorytm został praktycznie zaimplementowany, a uzyskane rezultaty potwierdzają jego bardzo wysoką efektywność.

Bibliografia:

[1] T. Politof, A. Satyanarayana, *Efficient algorithms for reliability analysis of planar networks - A survey*, IEEE Trans. Reliability, 1986 (35), s.252-259.

[2] F. Beichelt, P. Tittmann, *A Generalized Reduction Method for the Connectedness Probability of Stochastic Networks*, IEEE Trans. Reliability, 1991 (40), s.198-204.

[3] F. Beichelt, P. Tittmann, *Reliability analysis of communication networks by decomposition*, Microelectron. Reliab., 1991 (31), s.868-872.

[4] D. R. Shier, *Network Reliability and Algebraic Structures*, Oxford University Press, New York 1991.

[5] M. O. Ball, *Computational complexity of network reliability analysis: An overview*, IEEE Trans. Reliability, 1986 (R-35), 230-239.

[6] 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), s.777-788.

[7] L. G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Computing, 1979 (8), s.410-421

[8] L. B. Page, J. E. Perry, *A practical implementation of the factoring theorem for network reliability*, IEEE Trans. Reliability, 1988 (37) Aug, 259-267.

[9] R. K. Wood, *Factoring Algotithms for Computing K-Terminal Network Reliability*, IEEE Trans. Reliability, 1986 (R-35), s.269-278.

[10] R. K. Wood, *A Factoring Algorithm Using Polygon-to-Chain Reductions for Computing K-Terminal Network Reliability*, Networks, 1985 (15), s.173-190.

[11] M. G. C. Resende, *A Program for Reliability Evaluation of Undirected Networks via Polygon-to-Chain Reductions*, IEEE Trans. Reliability, 1986 (35) Apr, s.24-29.

[12] L. I. P. Resende, *Implementation of factoring algorithm for reliability evaluation of undirected networks*, IEEE Trans. Reliability, 1988 (37) Dec, s.462-468.

[13] R. K. Wood, *Triconnected decomposition for computing K-terminal network reliability*, Networks, 1989 (19), s.203-220.

[14] J. Carlier, C. Lucet, *A decomposition algorithm for network reliability evaluation*, Discrete Appl. Math., 1996, s.141-156.

[15] F. Moskowitz, *The analysis of redundancy networks*, AIEE Trans. Commun. Electron. 1958 (39), s.627-632.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1994.

[17] R. Sedgewick, *Algorithms in C*, Addison-Wesley, 1990.

[18] P. Wróblewski, *Algorytmy, struktury danych i techniki programowania*, Helion, 1997.

[19] L. Banachowski, K. Diks, W. Rytter, *Algorytmy i struktury danych*, WNT, 1996.

[20] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *Projektowanie i analiza algorytmów komputerowych*, PWN 1983.