Politechnika Krakowska
im. Tadeusza Kościuszki

Wydział Fizyki, Matematyki i Informatyki

Michał Wawrzak

Numer albumu: 97916

Fizyka sieci złożonych na przykładzie komórkowvch sieci "AD-HOC".

Physics of complex network using "AD-HOC" cellular networks.

Praca magisterska
na kierunku Fizyka Techniczna

Promotor:
dr Radosław Kycia

Uzgodniona ocena:.....

.....
podpisy promotora i recenzenta

Kraków 2016

Spis treści

1.Wstęp

We współczesnym świecie pojęcie grafu oraz sieci znane jest każdemu oraz przybiera ono różne formy. Otaczają nas one i są obecne praktycznie w każdej dziedzinie życia, z czego często możemy nie zdawać sobie sprawy. Już w 1736 roku grafy stały się narzędziem, dzięki któremu szwajcarski matematyk, Leonhard Euler rozwiązał zagadnienie mostów królewieckich – czy można przejść wszystkie mosty w tym mieście tak, aby przez każdy przejść tylko raz[1]. Od tego czasu znaczenie sieci w życiu ludzkim nieustannie rosło. Prawdziwa rewolucja nastąpiła jednak pod koniec lat dziewięćdziesiątych, a stało się tak w znacznej mierze poprzez rozwój technologii komunikacyjnych, informatycznych oraz spopularyzowanie Internetu jako globalnej sieci.

Wielu z nas nie wyobraża sobie dzisiejszej rzeczywistości bez sieci. Towarzyszy nam ona nieustannie, wykorzystujemy ją do wykonywania operacji bankowych czy komunikowania się poprzez portale społecznościowe. Jednak, pomimo coraz większego uzależnienia teraźniejszego świata od ich topologicznych właściwości, nie do końca rozumiemy zasady sterujące dynamiką tych układów. Przez cały czas rozwoju układów sieciowych, nauka zajmowała się przede wszystkim badaniem sieci regularnych i losowych. Bardzo często do drugiej kategorii zaliczane były sieci o złożonej budowie. Okazało się jednak, że takie podejście jest błędne, ponieważ w tym obszarze można wyróżnić charakterystyczne grupy, które stanowią zupełnie odrębne rodzaje sieci.

Podczas przeprowadzania symulacji przepływu informacji, zauważono, że wystarczy do regularnej sieci dodać kilka przypadkowych połączeń, aby znacznie przyśpieszyć przepływ danych. Dzięki temu odkryciu opracowano algorytm „Małego Świata". Wykorzystuje on mechanizm wyznaczania lokalnych i dalekich kontaktów dla modelu sieci o właściwościach charakterystycznych dla takich sieci. Uzyskane wyniki potwierdzają, iż algorytm ten skutecznie przeszukuje obszary dopuszczalnych wartości znajdując optymalne rozwiązania[2].

Sieci złożone są również obiektem fizyki układów złożonych. Pojęcie złożoności zaczęło funkcjonować w czasie, kiedy badane były systemy, których właściwości nie można było wywieść z właściwości elementów składowych lub układy były na tyle skomplikowane, że należało je opisywać używając języka statystyki matematycznej. Opis takich układów z punktu widzenia fizyki klasycznej był bardzo trudny, ze względu na brak przewidywalności ich zachowań. Pomimo wieloletnich badań nie udało się opisać funkcjonowania sieci w prosty

sposób, ponad to, często jedynym narzędziem do ich analizy pozostają symulacje komputerowe. Złożoność sieci oraz trudności w opisaniu ich zachowań jest dla każdego sprawą oczywistą. Fakt, że zbudowane są one z wielu węzłów, które są w skomplikowany sposób powiązane sprawia, że fizyczny opis tak skonstruowanych układów sieciowych jest bardzo trudny. Jednak pomimo tak zaawansowanej złożoności, sieci potrafią w precyzyjny sposób się rozrastać i dopasowywać do warunków otoczenia - samoorganizować. Przykładem takich zachowań może być usuwanie za długich (nieoptymalnych) połączeń, co prowadzi do powstawania tzw. hubów - węzłów o bardzo dużej ilości połączeń[1].

1.1. Cel i zakres pracy.

Jednym z głównych celów tej pracy jest przeprowadzenie analizy oraz przybliżenie struktury losowych sieci złożonych wykorzystując teorie grafowe, które są podstawowymi zagadnieniami dla tak skonstruowanych układów sieciowych.

Główną częścią tej pracy będzie napisanie programu, dzięki któremu będzie możliwa symulacja zachowań sieci „ad-hoc". Cały zamysł tej symulacji polegał na odzwierciedleniu zachowań ludzi na danym obszarze. Jest to ważne ze względu na fakt, iż wyniki otrzymane w pracy chcę w przyszłości wykorzystać, aby zamodelować sytuację, gdzie sieć tworzy się samoczynnie, przy pomocy urządzeń mobilnych będących w zasięgu swojej widoczności.

Dokonam również procesu selekcji użytecznych algorytmów grafowych, których implementacja pozwala na łatwiejsze badanie sieci, a także na dokładniejszą interpretację otrzymanych wyników. Ponadto zasymuluję zachowania sieci „ad-hoc"(losowej) w różnych warunkach wykorzystując do tego symulację, którą napisałem na potrzeby tej pracy.

Dodatkowo skupię się również na badaniu sieci pod względem jej modułowości. Dzięki wykonanym przeze mnie symulacjom przedstawię charakterystyki sieci dla zadanych warunków, takich jak zasięg, obszar implementacji, czy ilość węzłów. Będzie to opis wyników symulacji struktury sieci ad-hoc oparty na teorii. Wyniki mogą być użyte do projektowania takich sieci.

Przedstawiona praca składa się z siedmiu rozdziałów wraz ze wstępem. Rozdział drugi poświęcony jest sieciom złożonym, ich typom oraz zastosowaniu. Opisuje także teorie grafów, które są podstawą do rozważań na temat sieci złożonych. W rozdziale trzecim przedstawię charakterystykę błądzeń przypadkowych, na których oparłem symulator sieci „ad-hoc". Rozdział czwarty zawiera główną część tej pracy, przedstawione są w nim wyniki

części symulacyjnej, ich interpretacja oraz składowe elementy aplikacji. Znajduje się w nim także krótkie wprowadzenie do sieci Bluetooth oraz opis jak, dzięki wbudowanym protokołom komunikacyjnym powiązać je z aplikacją tworzącą sieci „ad-hoc". Kolejna część to krótkie podsumowanie. Na końcu pracy znajduje się bibliografia.

Praca skupia się na interdyscyplinarnych zagadnieniach z pogranicza matematyki, informatyki i fizyki, szczególnie układów złożonych.

1.2. Metodyka.

Praca ma charakter symulacyjny i polega na modelowaniu sieci ad-hoc przy różnych jej parametrach. Dlatego teoria grafów i sieci będzie obecna w całej pracy. W części symulacyjnej tej pracy wykorzystam wybrane narzędzia komputerowe. Program dzięki któremu zasymuluję działanie sieci „ad-hoc" napiszę w języku C++ w oparciu o zaimplementowane w nim biblioteki. Oprócz zamodelowania takiego układu i stworzenia szeregu zależności dla różnych przypadków, kolejnym celem będzie wizualizacja zachowań, czego dokonam przy pomocy biblioteki Allegro 5[3]. Otrzymane wyniki przeanalizuję, odpowiednio zinterpretuję oraz wykonam poglądowe wykresy zależności w oparciu o program Origin[4].

2.Sieci złożone.

W celu definicji sieci należy przybliżyć pojęcie grafu, którego sieć jest jednym z typów. Jednak ten typ jest ważny z praktycznego punktu widzenia, gdyż zawiera dodatkowe informacje potrzebne do lepszego opisu modelowanych zjawisk, np. przepływu. Dzięki niej możemy określić, dla przykładu jakie zależności zachodzą w danym układzie, w jaki sposób przekazywana jest informacja oraz jaka jest najkrótsza droga dla danego węzła. Znając te informacje można w dużym stopniu modyfikować właściwości sieci lub grafu poprzez dodanie węzła, czy utworzenie dotychczas nieistniejącego połączenia w celu jej optymalizacji. Wówczas wymiana informacji pomiędzy użytkownikami będzie znacznie szybsza. Najważniejszymi przykładami są m.in.: Internet i sieci telekomunikacyjne.

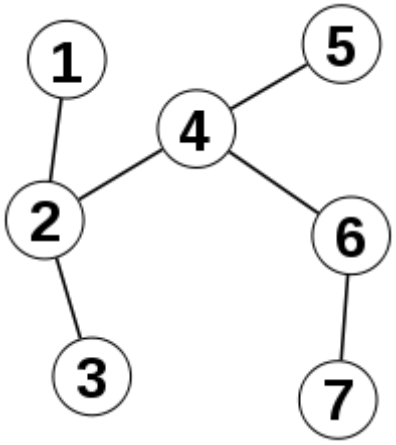
2.1. Definicja grafu, graf spójny, graf skierowany.

Graf to struktura matematyczna służąca do wizualizacji i analizy relacji zachodzących między obiektami. Jest on podstawą do rozważań dla całej teorii grafów. W uproszczeniu, graf to zbiór wierzchołków oraz krawędzi, które je łączą w taki sposób, że początkiem i końcem każdej krawędzi są wierzchołki. W notacji matematycznej graf oznacza się następująco:

$G = (V, E)$ - uporządkowana para wierzchołków i krawędzi;

$V = \{ 1, 2, ..., n \}$ - skończony zbiór wierzchołków grafu;

$E = \{ \{i, j\} : i \neq j \text{ i } i, j \in V \}$ - zbiór krawędzi grafu[5];



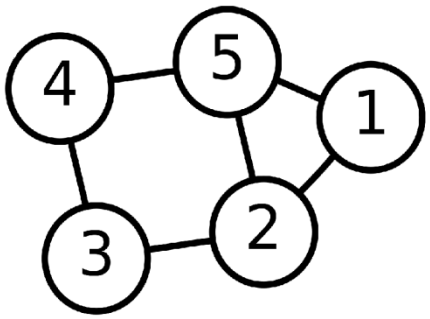
Rys.2.1. Przykład grafu[17]

Graf spójny [6]

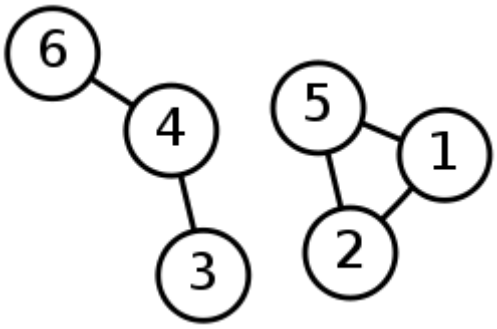
Graf jest *spójny*, jeżeli istnieje ścieżka, która łączy dwa dowolne wierzchołki. Innymi słowy, graf nie rozpada się na niepołączone części, tzn. jest spójny w znaczeniu potocznym. W przypadku, kiedy ten warunek jest spełniony mówimy, że posiada on tylko jedną składową spójną. S*kładowa spójna*, to zbiór największej możliwej liczby wierzchołków, wzajemnie ze sobą połączonych, który można wyodrębnić bez usuwania połączeń. To właśnie ze względu na nią, grafy dzielimy na:

- spójne - posiadające tylko jedną składową spójną,
- niespójne - posiadające przynajmniej dwie składowe spójne.

Ważnym parametrem jest również *stopień wierzchołka*, który określa liczbę bezpośrednich sąsiadów danego węzła w sieci.



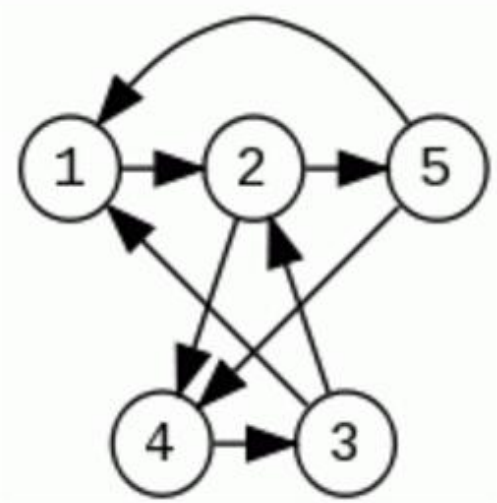
Rys.2.2. Przykład grafu spójnego



Rys.2.3. Przykład grafu niespójnego, który zawiera dwie składowe spójne. [18]

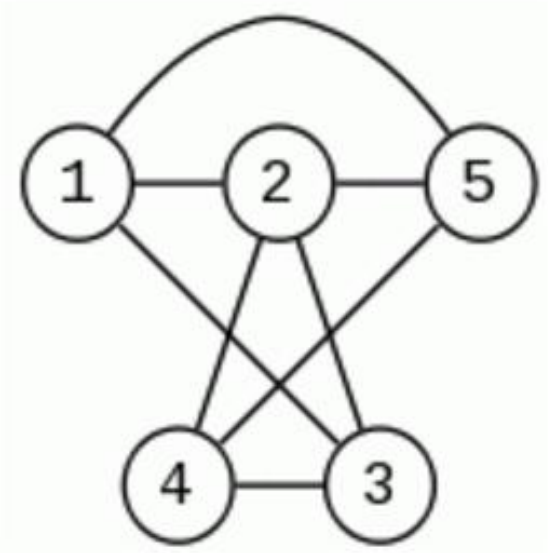
Graf skierowany(czyli sieć)[6]

Występują w nim połączenia, którym przypisany jest ściśle określony kierunek. Oznacza to, że dzięki jednej krawędzi można przejść, np. z wierzchołka A do wierzchołka B ale nie na odwrót jeżeli nie ma krawędzi w przeciwnym kierunku. W takim przypadku jeżeli chcielibyśmy przejść z A do B potrzebujemy dwóch połączeń o przeciwnych kierunkach pomiędzy nimi. Krawędzie w grafie skierowanym umownie oznacza się strzałkami zgodnie z ich kierunkami.



Rys.2.4. Przykład grafu skierowanego [19]

Przeciwieństwem grafu skierowanego jest graf nieskierowany. Nie występują w nim połączenia, które zabraniają przejścia w którąś ze stron - są one "dwukierunkowe". Aby przejść z wierzchołka A do B wystarczy jedno połączenie między nimi. Łatwo można zauważyć, iż jedno połączenie w grafie nieskierowanym, zastępuje dwa w grafie skierowanym. Warto również pamiętać, że z każdego grafu nieskierowanego można zrobić skierowany ale nigdy odwrotnie.



Rys2.5. Przykład grafu nieskierowanego [19]

2.2. Reprezentacja grafu za pomocą macierzy sąsiedztwa.

Graf jest reprezentowany za pomocą macierzy sąsiedztwa. Jest to macierz kwadratowa, a jej kolumny i wiersze numerowane są wierzchołkami grafu. Rozmiar zależy od ilości węzłów w układzie. Zapisywane są w niej wartości logiczne (1 w komórce (i,j) gdy występuje połączenie wierzchołków i-tego z j-tym , a 0 gdy go nie ma). Jest to matematyczny sposób przedstawienia grafu oraz połączeń jakie występują pomiędzy konkretnymi wierzchołkami. W celu jeszcze lepszego zrozumienia tego zagadnienia poniżej przedstawiam macierz sąsiedztwa dla grafu z Rys.2.5.

	I	II	III	IV	V
I	X	1	1	0	1
II	1	X	1	1	1
III	1	1	X	1	0
IV	0	1	1	X	1
V	1	1	0	1	X

Rys.2.6. Przykład macierzy sąsiedztwa

Graf może być reprezentowany również poprzez listy sąsiedztwa. Sposób ten polega na utworzeniu dla każdego wierzchołka listy zawierającej wszystkich jego bezpośrednich sąsiadów. Co ważne, w tej metodzie wierzchołki nie muszą być posegregowane. Poniżej przedstawiony jest przykład listy sąsiedztwa dla grafu z Rys.2.5.

I	II,III,V
II	I,III,IV,V
III	I,II,IV
IV	II,III,V
V	I,II,IV

Rys.2.7. Przykład listy sąsiedztwa

Oba przedstawione przykłady do reprezentacji grafu są równorzędne i można ich używać zamiennie. Lista sąsiedztwa jest efektywniejsza jeżeli chodzi o rozmiar i szukanie sąsiadów węzła, a macierz jeżeli chodzi o czas dostępu. W tej pracy graf reprezentowany jest przez macierz sąsiedztwa ze względu na jego duże rozmiary. W symulacji analizowane są grafy zawierające od 20 do 100 węzłów, przez co ta metoda opisu struktury jest wydajniejsza i bardziej przydatna ze względu na wygodny i szybszy dostęp do elementów macierzy, oraz prostotę sprawdzania konkretnych połączeń. Z kolei listy sąsiedztwa są lepszym wyborem dla grafów o małej liczbie wierzchołków, gdzie zamiast przeszukiwać macierz w celu znajdowania połączeń wystarczy spojrzeć, które węzły leżą w bezpośrednim sąsiedztwie dla badanego wierzchołka[6].

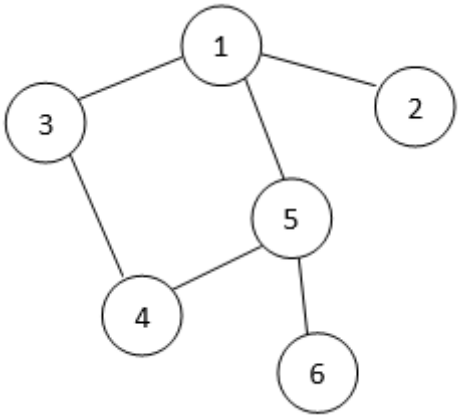
2.3. Składowa spójna grafu oraz sposób jej wyznaczania.

Problem ten pojawiał się już we wcześniejszych rozdziałach, kiedy przedstawione zostały różne typy grafów. Wiemy już, że *składowa spójna* to zbiór największej możliwej liczby wierzchołków, wzajemnie ze sobą połączonych, który można wyodrębnić bez usuwania połączeń.

Do jej wyznaczenia wykorzystujemy dowolny algorytm przeszukiwania. Polega on na przeszukiwaniu grafu od zadanego wierzchołka i wszystkie odwiedzone wówczas wierzchołki muszą należeć do tej samej składowej spójnej. W napisanej przeze mnie symulacji użyłem algorytmu DFS(ang. Depth First Search)[7] - czyli przeszukiwanie grafu w głąb. Działa on na

zasadzie stosu. Na początku wybieramy wierzchołek początkowy i umieścić go na stosie i oznaczyć jako „odwiedzony”. Następnie należy przejść do jego bezpośredniego sąsiada i powtórzyć procedurę. Dzieje się tak dla wszystkich wierzchołków, więc metodę tę można wywoływać poprzez rekurencję. Sytuacja zmienia się w momencie, kiedy dojdziemy do węzła, który nie ma już połączeń z „nieodwiedzonymi” wierzchołkami. W takim przypadku dany węzeł usuwa się ze stosu, oznacza jako "odwiedzony" i procedura jest powtarzana dla kolejnego wierzchołka ze stosu. Natomiast, jeżeli ten węzeł był już ostatnim na stosie, algorytm kończy swoją pracę. Wartym podkreślenia jest fakt, że algorytm zakończy przeszukiwanie tylko, jeżeli w układzie nie ma już "nieodwiedzonego" wierzchołka. Po jego zakończeniu możliwe są dwa typy otrzymanych rezultatów. Jeżeli metoda wywołana dla węzła początkowego "dotrze" do wszystkich pozostałych węzłów w układzie to graf jest spójny i mamy tylko jedną składową spójną.
Druga sytuacja następuje, gdy algorytm nie dotrze do każdego wierzchołka wychodząc od początkowego. Dostaniemy wtedy oprócz "głównej" - w skład której wchodzi wierzchołek początkowy również inne składowe[7].

Przykład działania algorytmu:



Rys.2.8. Graf nieskierowany. [20]

Przeszukiwanie w głąb zaczynamy od wierzchołka nr 1. Umieszczamy go na stosie i oznaczamy jako „odwiedzony” a następnie wrzucamy na stos jego bezpośrednich sąsiadów.

- Wierzchołki na stosie: nr 5, 3 i 2; Wierzchołki „odwiedzone” : nr 1.

Następnie bierzemy wierzchołek nr 2 ze stosu oznaczamy jako „odwiedzony” i z uwagi na to, że nie ma on bezpośrednich „nieodwiedzonych” sąsiadów nie dopisujemy nic do stosu.

- Wierzchołki na stosie: nr 5 i 3 ; Wierzchołki „odwiedzone” : nr 1 i 2.

W kolejnym kroku zajmujemy się węzłem nr 3 i oznaczamy go jako „odwiedzony”, a do stosu dopisujemy wierzchołek nr 4.

- Wierzchołki na stosie: nr 5 i 4; Wierzchołki „odwiedzone” : nr 1, 2 i 3.

Kolejna iteracja, to oznaczenie wierzchołka nr 4 jako „odwiedzony” i dopisanie do stosu węzła nr 5, który już jest na stosie.

- Wierzchołki na stosie: nr 5; Wierzchołki „odwiedzone” : nr 1, 2, 3 i 4.

Potem węzeł nr 5 oznaczamy jako „odwiedzony” i dopisujemy do stosu wierzchołek nr 6.

- Wierzchołki na stosie: nr 6; Wierzchołki „odwiedzone” : nr 1, 2, 3, 4 i 5.

W ostatnim kroku dla tego przykładu oznaczamy węzeł nr 6 jako „odwiedzony” i usuwamy go ze stosu, a z racji tego, że nie ma on połączeń z żadnym z „nieodwiedzonych” wierzchołków nie dopisujemy nic do stosu i zostaje on pusty. W tym momencie algorytm kończy pracę.

2.4 Typy sieci złożonych w fizyce.

Sieci złożone dzielimy na deterministyczne i przypadkowe. Jest to najbardziej naturalna klasyfikacja dla tych układów.

Sieci przypadkowe to takie, w których połączenia oraz węzły są rozłożone w sposób mniej lub bardziej losowy. W skład tych sieci wchodzą wszystkie sieci rzeczywiste oraz ich modele teoretyczne, których ewolucja dopuszcza wystąpienie czynnika losowości. Jest to ważna cecha, która odróżnia sieci przypadkowe od *deterministycznych*, dla których procedury konstrukcyjne nie dopuszczają żadnej przypadkowości[1].

W skład sieci przypadkowych wchodzą sieci ewoluujące. Zostały one odkryte przez Alberta i Barabasi'ego[1] podczas badania struktury sieci WWW. Okazało się wówczas, że sieć WWW jest bezskalowa. Pojęcie *bezskalowości* oznacza, że rozkład połączeń pomiędzy węzłami w takim układzie jest zgodny z rozkładem prawdopodobieństwa Yule-Simona, którego zaobserwowanie było wynikiem tego eksperymentu[1]:

$$P(k) \sim k^{-\alpha}, \tag{2.1}$$

gdzie :

P(k) - prawdopodobieństwo, że wybrany węzeł będzie miał stopień k,
k - stopień węzła,
α - współczynnik potęgowy.

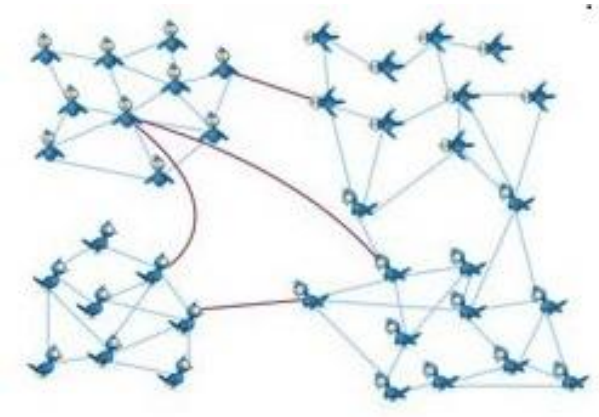
Taka sieć nie rozwija się według narzuconych reguł - lecz jest rezultatem indywidualnych decyzji jej użytkowników.

Odkrycie to pokazało, że wiele innych sieci wykazuje ten sam rozkład stopni wierzchołków oraz, że jest on konsekwencją dwóch mechanizmów: wzrostu sieci i reguły preferencyjnego dołączania węzłów. Zauważono również, iż sieci rzeczywiste nie tworzą się w wyniku czysto przypadkowych procesów, w których w losowy sposób wybiera się węzły oraz ich połączenia. Dzieje się tak dlatego, że sieci te nieustannie ewoluują. Pojawiają się w nich nowe połączenia, nowe węzły, które chętniej łączą się z węzłami już posiadającymi wiele innych połączeń. Mowa tu o zjawiskach, prowadzących do łatwiejszego przyciągania nowych węzłów, do istniejących węzłów z dużą ilością połączeń, niż do tych *słabiej usieciowionych* (o mniejszym stopniu wierzchołka). Wynika z tego, iż mechanizm preferencyjnego dołączania węzłów oznacza, że nowe węzły wchodzące do sieci z większym prawdopodobieństwem będą się łączyć z węzłami, które mają tych połączeń więcej, niż z takimi gdzie liczba połączeń jest mniejsza[1].

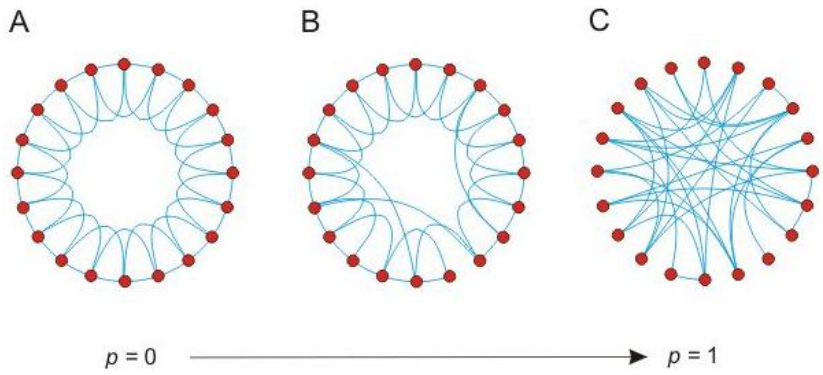
W skład sieci przypadkowych wchodzą również konstrukcje statyczne. Zostały one omówione przez węgierskich matematyków Erdősa i Rényiego [1]. Dzielą się one na:

- Sieci małych światów [8]

  Sieci małych rozmiarów opisuje model Wattsa-Strogatza[8]. Zakłada on, że sieci rzeczywiste mogą być jednocześnie silnie zgronowane (*współczynnik gronowania* C<=1, który jest stosunkiem liczby krawędzi między sąsiadami wierzchołka do wszystkich możliwych krawędzi pomiędzy tymi sąsiadami), rzadkie (kiedy stopień węzła k jest dużo mniejszy od liczby wierzchołków), a także wykazywać efekt małych światów. Współistnienie tych, jak mogłoby się wydawać wzajemnie wykluczających się cech jest możliwe dzięki występowaniu *dalekozasięgowych* połączeń (takich, które łączą węzły nie należące do tej samej podsieci Rys.2.9.). Zdecydowana większość dzisiejszych sieci wykazuje efekt małych światów.



Rys.2.9 Przykład połączeń dalekozasięgowych [21]



Rys.2.10 Tworzenie sieci małego świata[22]

- Grafy przypadkowe[9]

Są to grafy, które definiuje się, podając liczbę węzłów N należących do tego grafu, a następnie przypisuje się połączenia dla każdej z par z zadanym prawdopodobieństwem p. Istnieje kilka modeli grafów losowych:

- Model G(n,k) - w którym znane jest n, a następnie losowane są kolejne krawędzie ze zbioru wszystkich możliwych występujących w układzie z jednakowym prawdopodobieństwem.

- Model G(n,p) - w tym modelu również dana jest liczba węzłów n. Różnicę pomiędzy poprzednim modelem stanowi parametr p. Jest on prawdopodobieństwem krawędziowym określającym prawdopodobieństwo wystąpienia krawędzi w grafie. Może przyjmować wartości od 0 do 1.

- Model G(n,f) - tutaj również znane jest n. Składowa f definiuje maksymalny stopień dla wierzchołka, którego nie można przekroczyć.

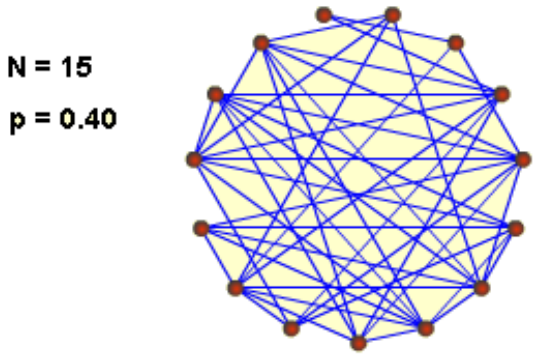
Graf przypadkowy różni się od „sieci” rzeczywistych (w znaczeniu potocznym), najlepiej widać to na dwóch podstawowych parametrach:

- większość sieci posiada potęgowy rozkład stopni wierzchołków, natomiast dla grafu przypadkowego, gdzie N>> k_z (k_z - stopień losowo wybranego wierzchołka) rozkład ten przechodzi w rozkład Poissona[10]:

$$P(k) = \frac{z^k e^{-z}}{k!}; \tag{2.2}$$

- posiada on także niski *współczynnik grupowania* (jest to gęstość bezpośredniego sąsiedztwa dla danego węzła), gdzie dla większości sieci rzeczywistych jest on wysoki[10]:

$$C \approx \frac{z}{N}; \tag{2.3}$$



Rys.2.11. Przykład grafu przypadkowego dla N=15 i p=0.40 [23]

Po przybliżeniu sieci przypadkowych, zajmę się pokrótce sieciami deterministycznymi. Charakteryzują się one całkowitym brakiem losowości oraz odgórnie ustalonym stanem wyjściowym. W tym przypadku ewolucja takiego układu jest ściśle określona i zależna od parametrów początkowych. Układy te opisywane są przez grafy regularne, gdzie od każdego wierzchołka N wychodzi N-1 połączeń. Są one znacznie mniej

skomplikowane od sieci losowych. Dzięki regularnej budowie oraz przewidywalnej ewolucji są one prostsze w opisie fizycznym. Znając warunki początkowe jesteśmy w stanie dokładnie opisać dany układ. W skład sieci deterministycznych wchodzą: sieci regularne, które charakteryzują się powtarzalnością i regularnym ułożeniem węzłów oraz sieci fraktalne, których charakterystyczną cechą jest samopodobieństwo, tzn. podobieństwo całości do jego części składowej. Przykładem sieci regularnych są m.in.: sieci krystaliczne ciał stałych, a sieci fraktalnych - Krzywe Kocha[11].

2.5. Zastosowanie.

Dzięki swoim zaawansowanym właściwościom oraz samoewoluującej topologii, sieci złożone mają bardzo szerokie zastosowanie w wielu dziedzinach m.in.[12]:

- w biologii – gdzie, z potrzeby usystematyzowania wiedzy o niezwykle skomplikowanych i zaawansowanych oddziaływaniach żywych komórek pozwalają na modelowanie tych zależności np. symulowanie połączeń neuronów, gdzie pomimo dużej ilości neuronów, mózg ludzki wykazuje dużą regularność połączeń[12].

- do badań zachowań socjologicznych- gdzie siec jest zbiorem osób i relacji między tymi osobami. Taka sieć może być wykorzystana m.in. do symulacji rozprzestrzeniania się chorób zakaźnych, czy odpowiedniego ukierunkowania reklamy, portale społecznościowe i sieci połączeń profili.

3. Błądzenie przypadkowe.

Jest to pojęcie używane w fizyce, określające zachowanie się ruchu losowego. Polega ono na przemieszczaniu się danego obiektu ze swojego położenia początkowego do innego, losowo wybranego, na którego wybór nie mamy wpływu lub jest on ograniczony. Błądzenie przypadkowe zostało użyte w napisanej prze ze mnie symulacji, gdzie węzły losowały sobie położenia początkowe, a następnie z rozkładów prawdopodobieństwa losowane były ich prędkości, oraz kierunki zgodnie z którymi się poruszały. Całość została napisana tak, aby symulacja była jak najbardziej zbliżona do ruchu przypadkowego. W następnych podrozdziałach omówię rozkłady które najczęściej używane są do modelowania błądzenia przypadkowego

3.1. Rozkład Gaussa[13].

Zwany również rozkładem normalnym, jest niezwykle ważnym rozkładem prawdopodobieństwa w wielu dziedzinach nauki.

Funkcja gęstości rozkładu normalnego to przykład funkcji Gaussa :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}, \tag{3.1}$$

gdzie:

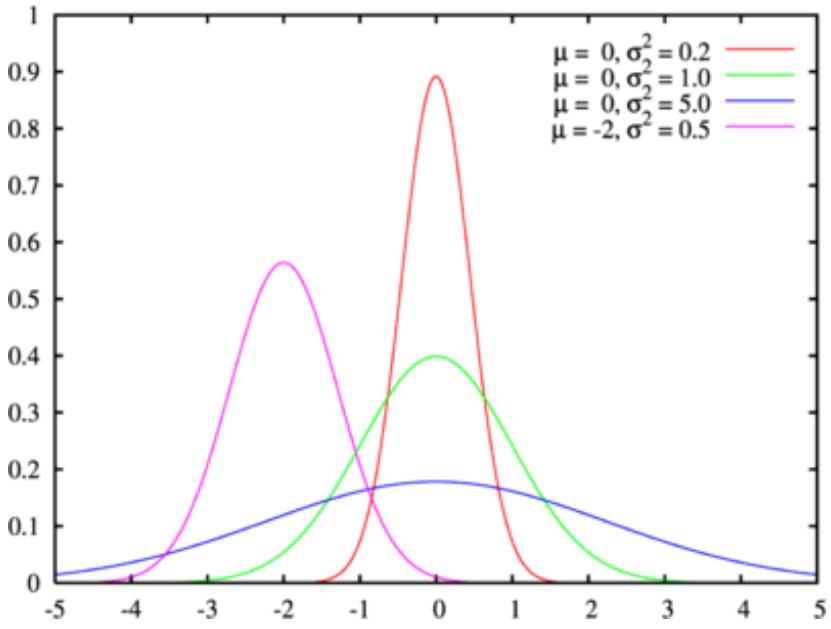
μ – średnia,

σ - odchylenie standardowe.

W znormalizowanym rozkładzie średnia wynosi zero, a odchylenie standardowe jeden(μ=0 i σ=1) i jest to tzw. rozkład normalny, którego funkcja gęstości wynosi:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}; \tag{3.2}$$

Poniżej przedstawiam rozkład gęstości dla rozkładu normalnego, dla przykładowych parametrów μ i σ:



Rys.3.1 Gęstość rozkładu normalnego [24]

Widzimy, że funkcja gęstości jest symetryczna względem wartości średniej oraz, że w tym punkcie osiąga maksimum. Charakterystycznym dla takiego rozkładu jest fakt, że około 68% całkowitego pola pod wykresem znajduje się w odległości jednego odchylenia standardowego od średniej, a aż 95,5% w odległości dwóch. Można również zaobserwować, że im większa wartość odchylenia standardowego, tym wykres jest bardziej "płaski".

Przykład błądzenia dla rozkładu Gaussa. Początek błądzenia rozpoczyna się w punkcie (0,0) czyli w środku układu. Wielkość pudła symulacyjnego to 600 x 400 j.u(jednostek umownych), a ilość kroków równa 10000.

Rys.3.2 Rozkład Gaussa.

Rozkład Gaussa jest przykładem dyfuzji normalnej. Polega ona na tym, że cząstki wykonują bardzo dużo małych przemieszczeń, a praktycznie żadnych dłuższych. Bardzo dobrze widoczne jest to na powyższym rysunku (Rys.3.2).

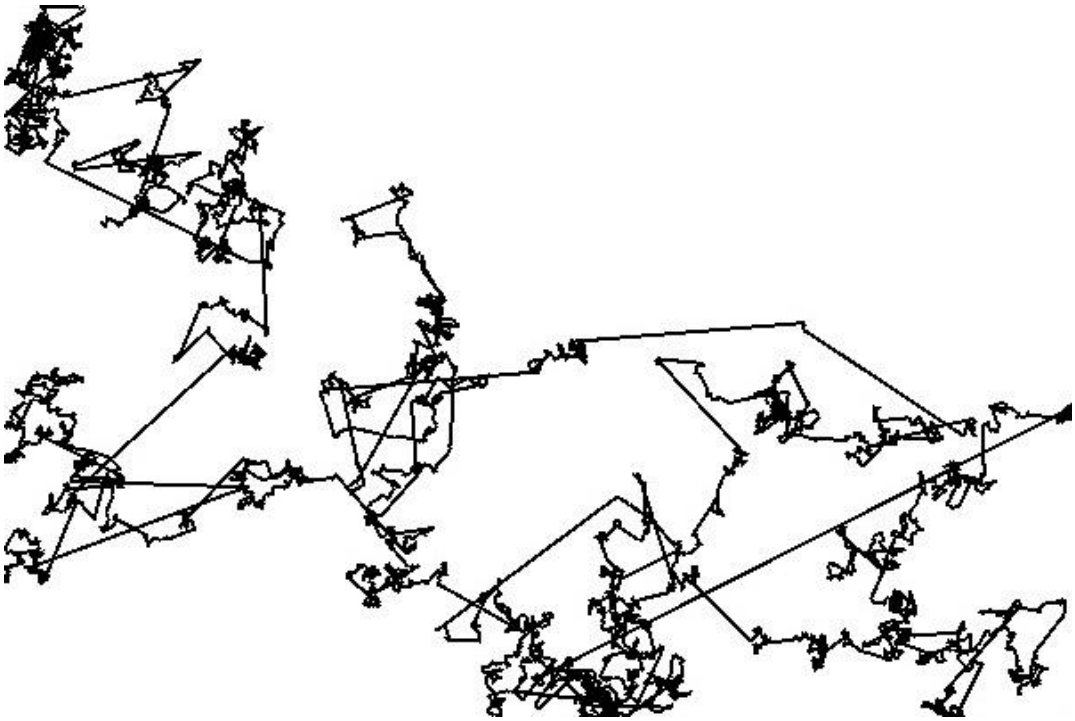
3.2. Loty Levy'ego. [14]

Rozpoczynając analizę Lotów Levy'ego warto na początku skupić się na zjawisku dyfuzji. Polega ono na fizycznym procesie samorzutnego rozprzestrzeniania się cząstek w pewnym ośrodku, w wyniku chaotycznych zderzeń cząsteczek substancji, z cząstkami otoczenia. Jest to jeden z mechanizmów transportu. Jego charakterystyczną cechą jest średnia odległość R, jaką pokona cząstka w czasie t, która jest proporcjonalna do pierwiastka z czasu:

$$R(t) \sim \sqrt{t}\,; \tag{3.3}$$

W tym przypadku, w porównaniu do poprzedniego rozkładu, oprócz dużej ilości małych „skoków” pojawiają się także duże „skoki”. W fizyce takie zjawisko zwane jest lotami Levy'ego.

Przykład błądzenia dla Lotów Levy'ego przedstawia Rys.3.3. Widoczne są charakterystyczne skoki po których następuje krótkie błądzenie wokół punktu końcowego, a następnie kolejny długi skok.

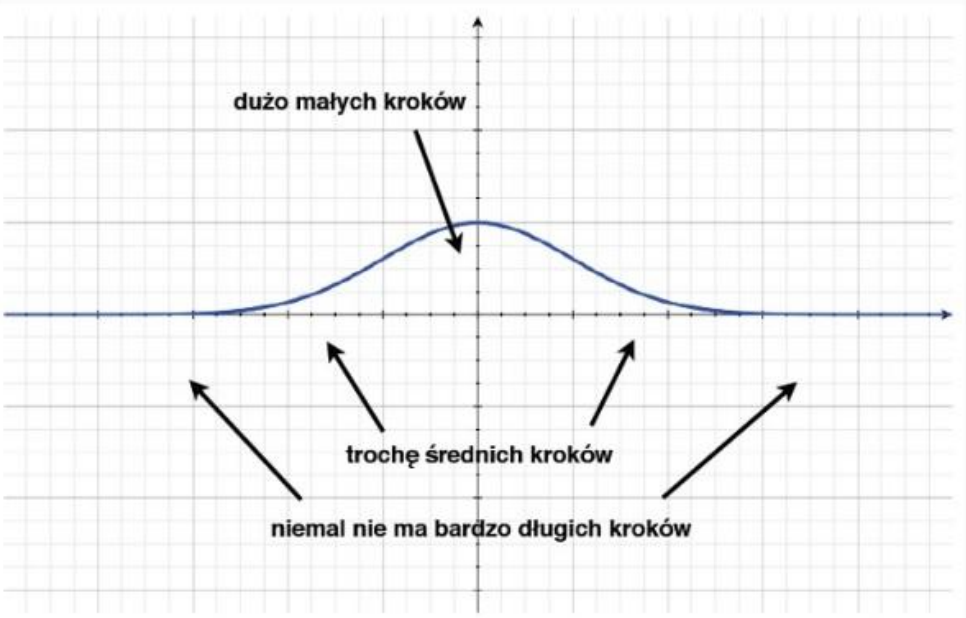


Rys.3.3 Loty levy'ego.

Łatwo można zobaczyć na przedstawionym przykładzie, że występuje tu bardzo dużo małych przemieszczeń ale wyraźnie widoczne są też dalekie przemieszczenia. Okazało się również, że ten fizyczny model przemieszczeń ma swoje odwzorowanie w zachowaniach łownych albatrosów[14]. Po dokładnych analizach dotyczących życia tych ptaków, zauważono, że zachowują się one niemalże identycznie. Przebywają daleki dystans w poszukiwaniu jedzenia oraz odpowiednich warunków do życia, a gdy już znajdą odpowiednie miejsce, wykonują małe przemieszczenia w obrębie danego obszaru.

3.3. Porównanie rozkładów Gaussa i Levy'ego.

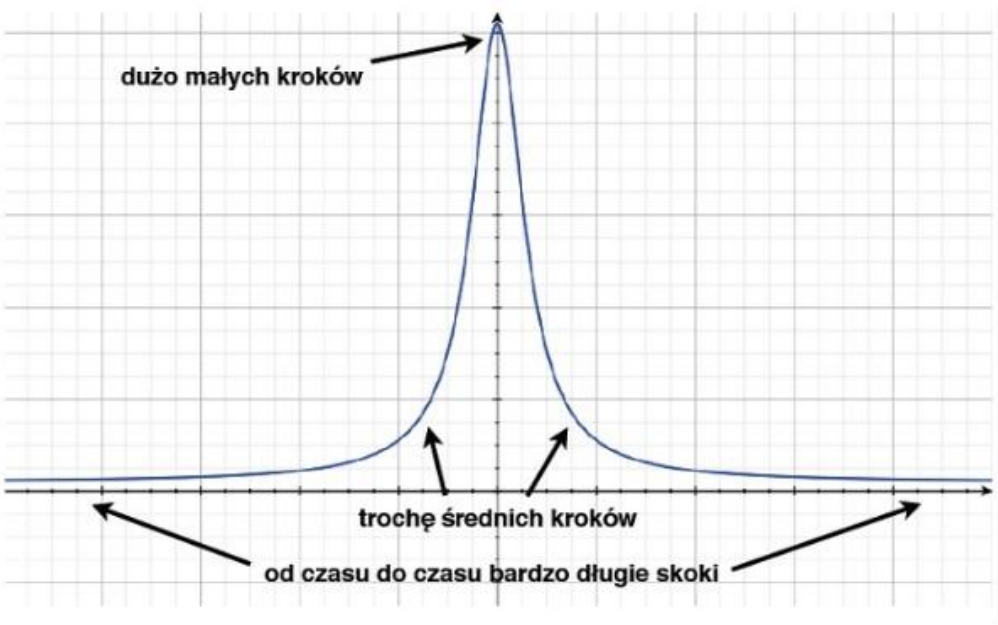
Zjawiska obu dyfuzji można przedstawić w formie rozkładów prawdopodobieństwa. W dyfuzji normalnej mamy do czynienia z dużą ilością małych kroków, a także niewielką ilością kroków średnich. Można to przedstawić w formie wykresu, gdzie na osi odciętych będzie przesunięcie cząstki w jednym kroku czasowym, a na osi rzędnych zaznaczymy jak często dane przesunięcie występuje - Rys. 3.4



Rys.3.4 rozkład dla Gaussa[25].

Powyższy wykres kształtem przypomina krzywą Gaussa, która jest niezwykle ważna w wielu dziedzinach nauki. Ma ona bardzo duży wpływ na rozkład prawdopodobieństw.

Inaczej sytuacja wygląda dla dyfuzji anomalnej. Gdzie tak, jak wcześniej występuje duża ilość małych kroków, ale oprócz nich występują kroki średnie i długie - Rys. 3.5.



Rys.3.5 Rozkład Levy'ego[25].

Wykres ten reprezentuje rozkład Levy'ego, a jego cechą charakterystyczną są długie „ogony” dla dużych bezwzględnych wartości na osi odciętych.

Znaczenie obu rozkładów dla symulacji i modelowania zachowań w sieci.

W rozkładzie Gaussa widzimy duże skupienie węzłów wokół węzłów "głównych". Na podstawie tego można modelować sytuacje, gdzie wszystkie węzły są zależne od węzła głównego i poruszają się razem z nim. Na bazie tego rozkładu można zamodelować np. łączność w jednostce wojskowej działającej na polu walki, gdzie żołnierze skupieni są wokół swoich dowódców.

Z kolei w rozkładzie Levy'ego, widzimy mniejszą zależność między węzłami, a także ich duże skoki. Dzięki temu bardziej nadaje się on do symulacji człowieka z urządzeniem mobilnym, którego zamodelowanie jest podstawowym celem tej pracy. Dzięki lotom Levy'ego modelujemy np. wyjście do pracy, spędzenie tam kilku godzin, powrót do domu i poruszanie się po określonym położeniu. Jak widzimy, bardziej realistyczne zachowania osób z pewnością łatwiej będzie opisać za pomocą dyfuzji anomalnej.

4. Część symulacyjna.

Jest to główna część pracy. Zajmę się w niej przede wszystkim programem symulującym zachowanie, ewolucję i tworzenie sieci „ad-hoc", który został napisany na potrzeby tej pracy. Przedstawię założenia, warunki początkowe oraz cele, które chciałem osiągnąć. Opiszę także wykorzystane w nim funkcje, oraz przedstawię odpowiadające im fragmenty kodu. Listing całego programu zamieściłem w załączniku do pracy.

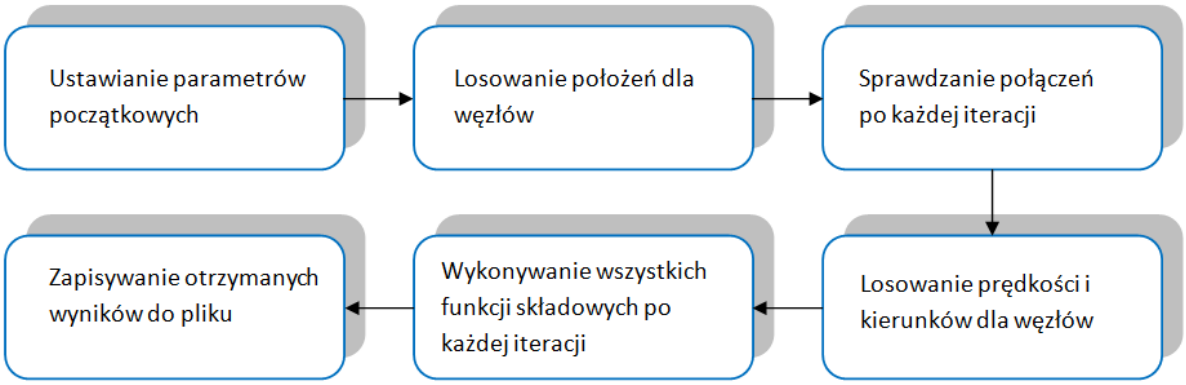
Następnie przejdę do interpretacji otrzymanych wyników, które zaprezentuję jako wykresy zależności dla odpowiednich warunków. Warunki oraz przypadki wybrałem w oparciu o rozważania, które z nich najlepiej zobrazują charakterystykę tak utworzonych sieci. Podstawowymi parametrami są m.in.: ilość połączeń po określonym czasie, średnia liczba połączeń w zależności od ilości węzłów, ich zasięgu, a także rozmiar obszaru symulacji oraz procent pokrycia obszaru przez utworzoną sieć. Te oraz inne zależności zostaną przedstawione przeze mnie w następnych rozdziałach.

Należy także pamiętać, że sieci są modelami statystycznymi, co za tym idzie otrzymane wyniki poddałem uśrednianiu przed ich interpretacją.

4.1. Symulacja tworzenia i ewolucji sieci.

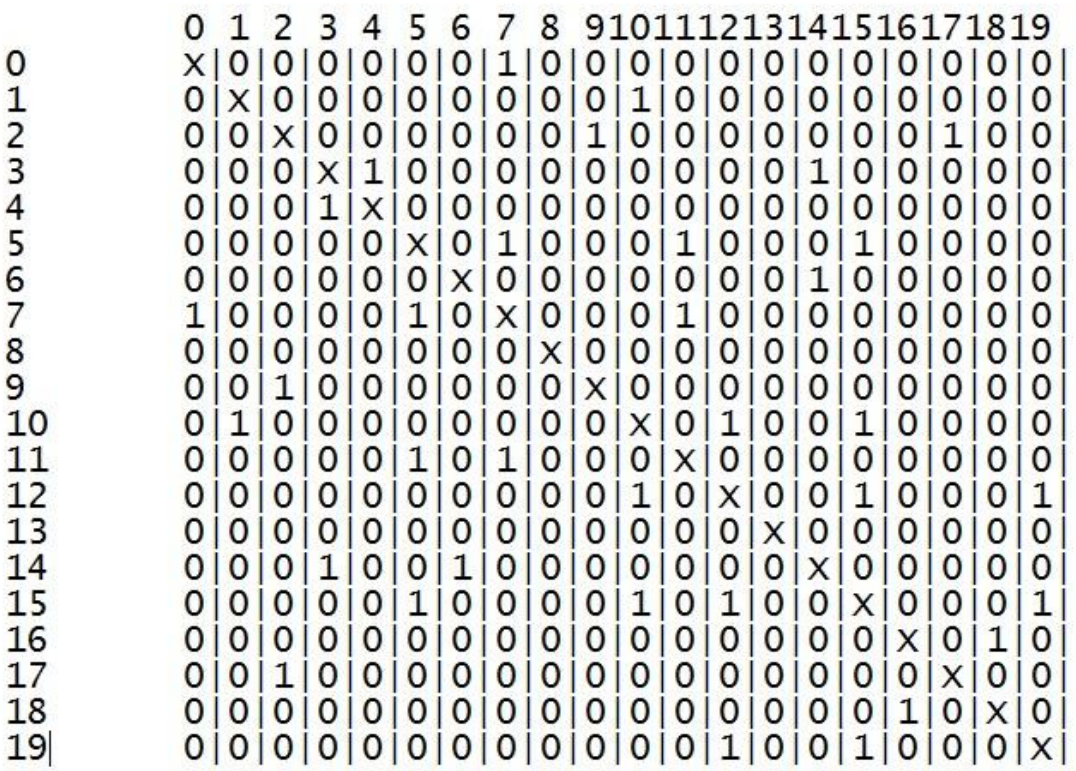
Główną częścią tej pracy było napisanie programu, dzięki któremu będzie możliwa symulacja zachowań sieci „ad-hoc". Cały zamysł tej symulacji polegał na odzwierciedleniu zachowań ludzi na danym obszarze. Jest to ważne ze względu na fakt, iż chcemy zamodelować sytuację, gdzie sieć tworzy się samoczynnie, przy pomocy urządzeń mobilnych będących w zasięgu swojej widoczności. Innymi słowy, jeżeli kilka osób znajduje się odpowiednio blisko siebie, ich urządzenia mobilne łączą się w sieć, przez co urządzenie staje się węzłem takiego układu. Właśnie z tego względu zastosowane zostały rozkłady Gaussa i Levy'ego, które najlepiej symulują zachowania ludzi, a także wielu gatunków zwierząt.

Kod źródłowy programu został napisany w języku C++, a wizualizacja tego zagadnienia została zrobiona w oparciu o bibliotekę Allegro. Aplikacja ta ma za zadanie jak najdokładniej odwzorować przykładową sieć „ad-hoc" tworzoną całkowicie losowo.



Rys.4.1 Schemat blokowy aplikacji

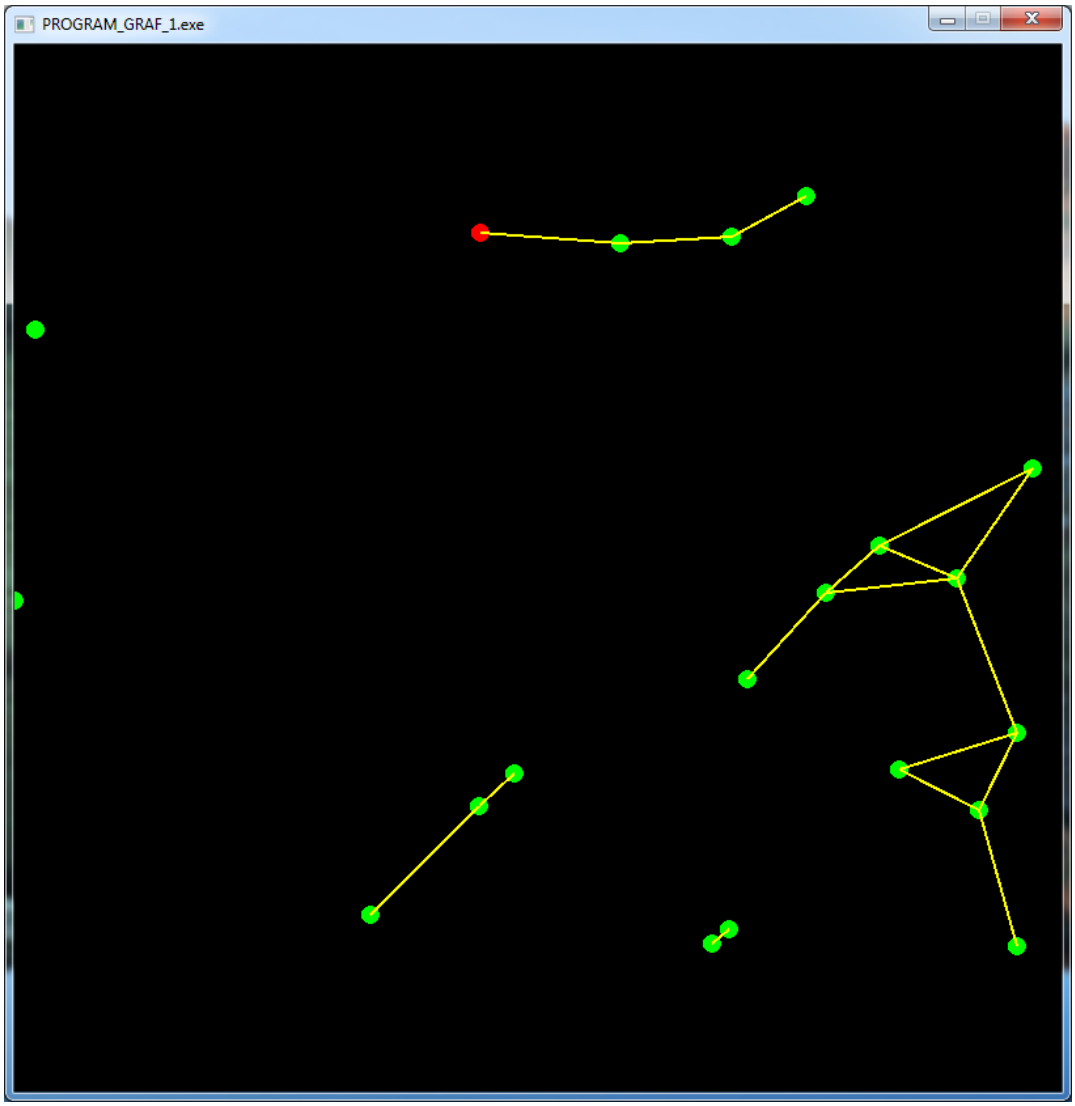
Zasada działania polega na określeniu, jak duży ma być obszar symulacji (L), ile węzłów (N) ma się w nim znajdować oraz jak daleki zasięg (R) będą miały. Obszar symulacji przyjąłem jako kwadrat, dlatego do opisania jego wielkości wystarczy jedna zmienna. Te trzy parametry, to warunki początkowe dla tej symulacji. Po ich podaniu, każdy węzeł losuje swoje położenie a następnie sprawdza, czy w jego zasięgu znajduje się inny. W przypadku, kiedy odległość między nimi wynosi mniej, niż 2R, powstaje połączenie. Sprawdzanie sąsiedztwa oparte jest o funkcję „neighbour", która po każdej iteracji sprawdza sąsiadów dla każdego węzła w sieci. W kolejnych krokach iteracyjnych, dla wszystkich węzłów losowane jest przyspieszenie oraz jego kierunek. Wartości te otrzymujemy dzięki zastosowaniu rozkładów Gaussa oraz lotów Levy'ego. Na ich podstawie wyliczamy, w jakim kierunku oraz z jaką prędkością węzły będą się poruszać. Funkcja „neighbour" wykonywana jest po każdej iteracji, dokładnie śledząc całą ewolucję układu. Kolejnym krokiem było utworzenie macierzy sąsiedztwa. Jest to macierz o rozmiarach N x N, w której zapisane jest każde połączenie dla każdego węzła. Jest to macierz zerojedynkowa, 1-jest połączenie, 0 - nie ma połączenia. Oczywiście "x" oznacza, że węzeł nie może się połączyć sam ze sobą. Poniżej zamieszczona jest przykładowa macierz dla 20 węzłów:



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	x	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	x	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
2	0	0	x	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
3	0	0	0	x	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
4	0	0	0	1	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	x	0	1	0	0	0	1	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	x	0	0	0	0	0	0	0	1	0	0	0	0	0
7	1	0	0	0	0	1	0	x	0	0	0	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	x	0	0	0	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	x	0	0	0	0	0	0	0	0	0	0
10	0	1	0	0	0	0	0	0	0	0	x	0	1	0	0	1	0	0	0	0
11	0	0	0	0	0	1	0	1	0	0	0	x	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	1	0	x	0	0	1	0	0	0	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	x	0	0	0	0	0	0
14	0	0	0	1	0	0	1	0	0	0	0	0	0	0	x	0	0	0	0	0
15	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	x	0	0	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	0	1	0
17	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	x	0
19	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	x	

Rys.4.2. Przykład macierzy sąsiedztwa dla 20 węzłów

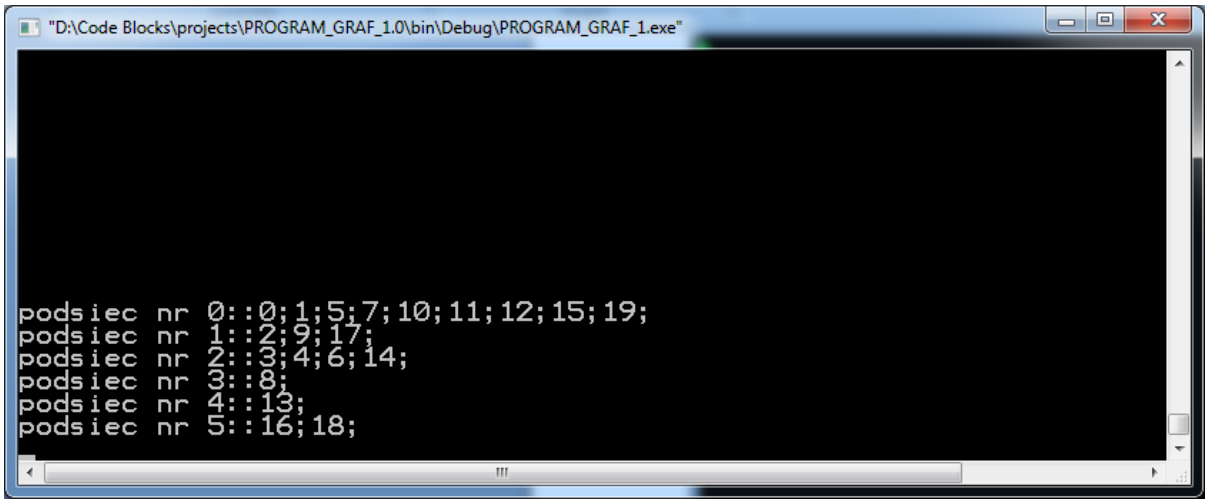
Posiadając informacje o wszystkich połączeniach w układzie, kolejnym krokiem było wyznaczenie składowych spójnych, czyli wyznaczenie części, które są ze sobą połączone. Są to mniejsze grafy, które można wyodrębnić bez usuwania połączeń. W rozwiązaniu tego problemu pomógł algorytm przeszukiwania w głąb – DFS, który omówiłem w rozdziale 2.3. Znajdował on najdłuższe możliwe połączenia i na tej podstawie dzielił na poszczególne „podsieci”. Poniżej przedstawiam prostą wizualizację tego problemu:



Rys.4.3. Wizualizacja połączeń węzłów na podstawie macierzy sąsiedztwa.

Od razu możemy zauważyć, że jest to graficzne przedstawienie macierzy połączeń. Dla spójności algorytmu założyłem, że pojedyncze węzły to najmniejsza „podsieć”.

Dzięki zaimplementowaniu algorytmu DFS, wiadomo było, który węzeł wchodzi w skład konkretnej podsieci oraz jak duży mają one zasięg.



Rys.4.4. Przykład rozpisanych składowych spójnych oraz węzłów, które do nich należą.

Była to kluczowa, lecz nie jedyna funkcjonalność potrzebna do późniejszej analizy. Ponadto, program zwracał liczbę wszystkich połączeń w układzie w zadanym momencie. Odpowiedzialna za to funkcja zliczała wszystkie połączenia jakie wykryła funkcja „neighbour". Co więcej, program pokazywał liczbę wszystkich możliwych połączeń, ze wzoru (N(N-1))/2(liczba wszystkich połączeń pomiędzy N punktami), a także średnią liczbę połączeń występujących w obszarze w czasie rzeczywistym.

Mając wszystkie te wartości, można było przejść do analizy tak utworzonych sieci. Wyniki, charakterystyki i obserwacje znajdują się w części symulacyjnej tej pracy.

Poniżej przedstawiłem najważniejsze funkcje programu wraz z odpowiadającym im kodem źródłowym.

- Funkcja „randXY" odpowiedzialna jest za losowanie położeń początkowych dla każdego węzła.

```
void randXY(double *x, double *y, int n, double l)
  {
    for(int i=0; i<n; i++)
    {
      x[i] = l*((rand()%RAND_MAX) / (1.0 * RAND_MAX));
      y[i] = l*((rand()%RAND_MAX) / (1.0 * RAND_MAX)); } }
```

- Funkcja „dist" to funkcja odpowiedzialna za błądzenie przypadkowe oparte na rozkładzie Levy'ego

```
float dist (double c, double alpha) {

 const gsl_rng_type * T;
 gsl_rng * r;
 gsl_rng_env_setup();
 struct timeval tv;
 gettimeofday(&tv,0);
  unsigned long mySeed = tv.tv_sec + tv.tv_usec;
  T = gsl_rng_default;
 r = gsl_rng_alloc (T);
   gsl_rng_set(r, mySeed);
 double u;
          u = gsl_ran_levy(r,c,alpha);
gsl_rng_free (r);
 return (float)u;
}
```

- Funkcja „randv” to funkcja losująca prędkości i ich kierunki dla każdego z węzłów. Oparta jest ona o funkcję „dist” która odpowiada za rozkład Levy'ego

```
void randv(double *x, double *y, int n, double a,double b)
   {
      for(int i=0; i<n; i++)
      {

         x[i] = dist(5.0,2.0);
         y[i] = dist(5.0,2.0);
      }

      double maxx=0;
      double maxy=0;

      for(int i=0; i<n; i++)
      {
         if(x[i] >maxx)maxx = x[i];
         if(y[i] >maxy)maxy = y[i];
      }
      if (maxx>maxy)maxy=maxx;
      else maxx=maxy;

      for(int i=0; i<n; i++)
      {

         x[i]/= maxx;
         y[i]/=maxy;
         x[i] =2*b*x[i] - b;
         y[i] =2*b*y[i] - b;
 }
```

- Funkcja „neighbour” sprawdza po każdej iteracji wszystkich sąsiadów dla każdego węzła, i w przypadku gdy odległość między nimi jest mniejsza niż 2R, tworzy połączenie.

```
void neighbour(double *x,double *y,double SIGMA)
 {
 double rijX,rijY,R_sq;
 double rijX2,rijY2,Circ_2;
 Circ_2=SIGMA;
 for(int is=0; is<N; is++)
{
 for(int js=0; js<N; js++)
 {
            rijX=x[is]-x[js];
           rijY=y[is]-y[js];

          rijX2=rijX*rijX;
          rijY2=rijY*rijY;
          R_sq=sqrt(rijX2+rijY2);

           if(R_sq<Circ_2 ) neib[is][js]=1;
            else neib[is][js]=0; }}}
```

Na podstawie tej funkcji wyliczana jest także macierz sąsiedztwa (Przykład Rys.4.1)

```
neighbour(x,y,R);
        for(int g=0; g<N; g++)
         for(int h=0; h<N; h++)
                  neib_template[g][h] = neib[g][h];
```

- Funkcja „CountConn” odpowiedzialna jest za podanie wszystkich występujących połączeń w układzie, dla danego kroku czasowego.

```
int CountConn()
{
 int allConn =0;
 for (int i=0; i<N; i++)
    for(int j=i; j<N; j++)
      if (neib[i][j]!=0 && j!=i) allConn++;
return allConn;
 }
```

- Funkcja „znajdź_podsiec” oparta jest na wektorach. Jest to struktura danych reprezentująca tablicę, pozwalająca na modyfikację rozmiaru tablicy w trakcie jej życia. Funkcja ta ma za zadanie znajdować składowe spójne, jakie występują na

danym obszarze. Dzięki niej, wiemy ile jest składowych spójnych, oraz które węzły do nich należą. (Przykład - Rys.4.2)

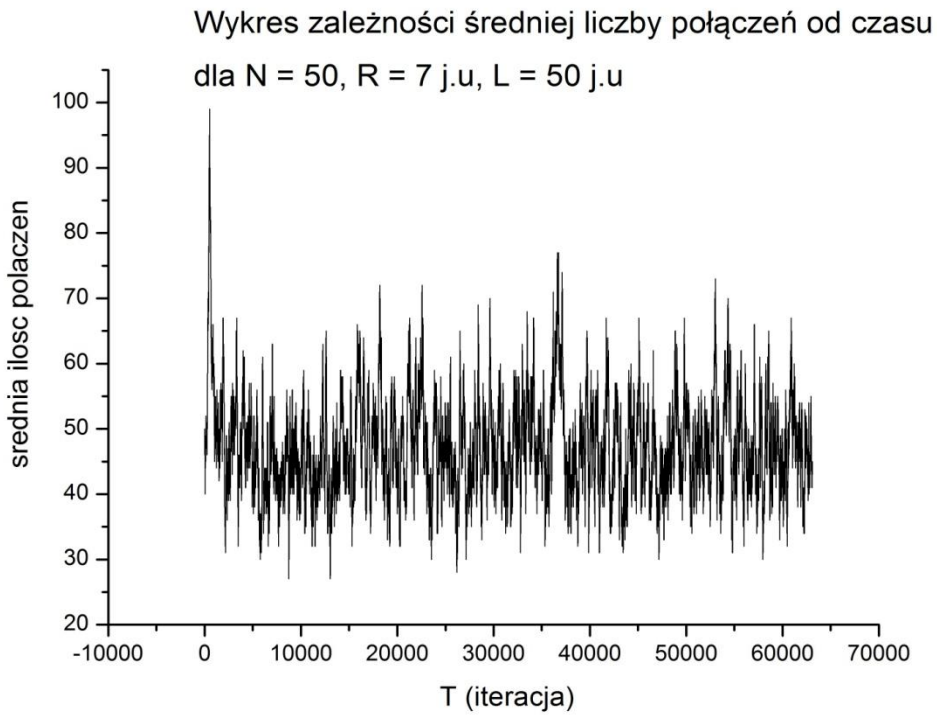
```
vector<vector<int>> znajdz_podsiec(int ** neib, int N)
 {
  vector<vector<int>> output;
 vector<int> buff;
 bool * wezly;
 wezly = new bool [N];
 for (int i =0; i<N; i++)
    wezly[i] = true;
      bool go = true;
 while(go)
{
 go=false;
 for (int i=0; i<N; i++)
        {
         if(wezly[i])go=true;
 }
       for(int i=0; i<N; i++)
       {
        if (wezly[i])
       {
        spjp(i);
       for(int j=0; j<N; j++)
       {
              if (D[j]<nieskonczonosc && D[j]>=0)
             {
        buff.push_back(j);
        wezly[j] =false;
              }
       }
        output.push_back(buff);
       buff.erase(buff.begin(),buff.end()); }}}
```

W tym rozdziale przybliżyłem najważniejsze funkcje, z których składa się aplikacja. Sposób w jaki one ze sobą oddziałują oraz ich konkretne zastosowanie pokażę w kodzie zamieszczonym na końcu pracy.

4.2. Rezultaty symulacji.

W tym rozdziale przedstawię otrzymane i opracowane przeze mnie wyniki, które dobrze charakteryzują samoorganizację i ewolucję sieci „ad-hoc”.

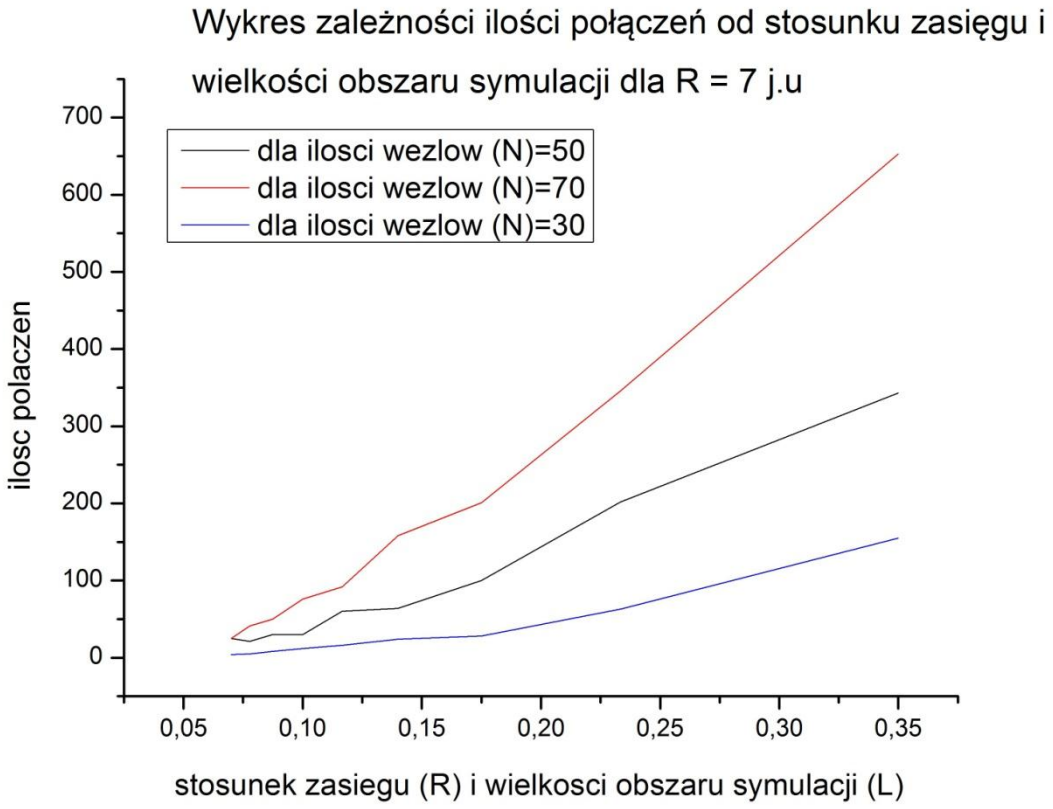
Pierwszym aspektem, którym się zająłem, było wyznaczenie średniej liczby połączeń w danym kroku czasowym. Do wykonania tej analizy przyjąłem ilość węzłów (N) = 50, zasięg każdego węzła (R) = 7 jednostek umownych (j.u), a rozmiar obszaru symulacyjnego wynosił 50 x 50 j.u. Prędkości węzłów zostały wylosowane zgodnie z rozkładem Levy'ego, który uzyskałem przy pomocy funkcji gsl_ran_levy z biblioteki GNU Scientific Library[15]. Funkcja ta potrzebuje dwóch parametrów: c- odpowiadający za skalę i alfa który jest wykładnikiem eksponenty. W moim przypadku współczynniki rozkładu Levy'ego wynosiły odpowiednio: c= 5 i alfa= 2. Dla tak wybranych wartości parametrów powstała optymalna liczba dużych i małych kroków, która symuluje ruch osób na małym obszarze.



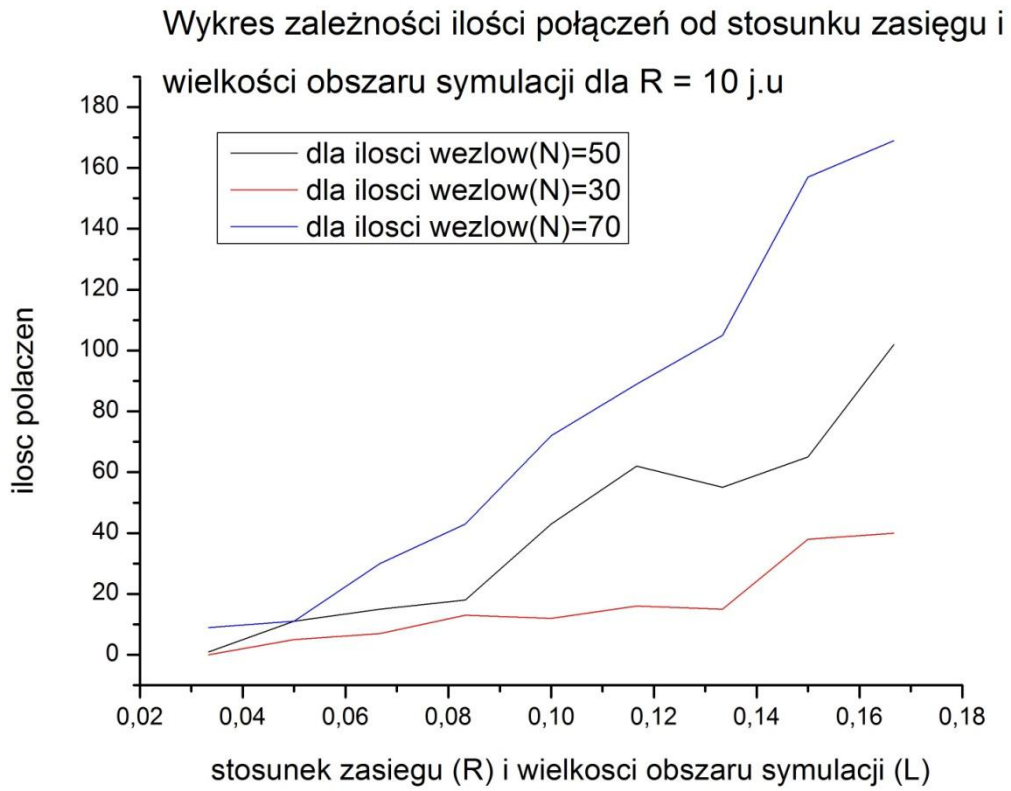
Rys.4.1 Wykres zależności średniej liczby połączeń od czasu.

Ilość połączeń wyraźnie oscyluje wokół progu 50 połączeń, co pokazuje, że dla tak dobranych parametrów początkowych statystyka ta jest poprawna.

Następnym krokiem było wykonanie analizy zależności ilości połączeń występujących w układzie, od stosunku zasięgu węzłów i wielkości obszaru symulacyjnego. Do tych statystyk przyjąłem liczbę węzłów (N) = 30, 50 i 70 i zasięg (R) = 7 i 10 j.u.

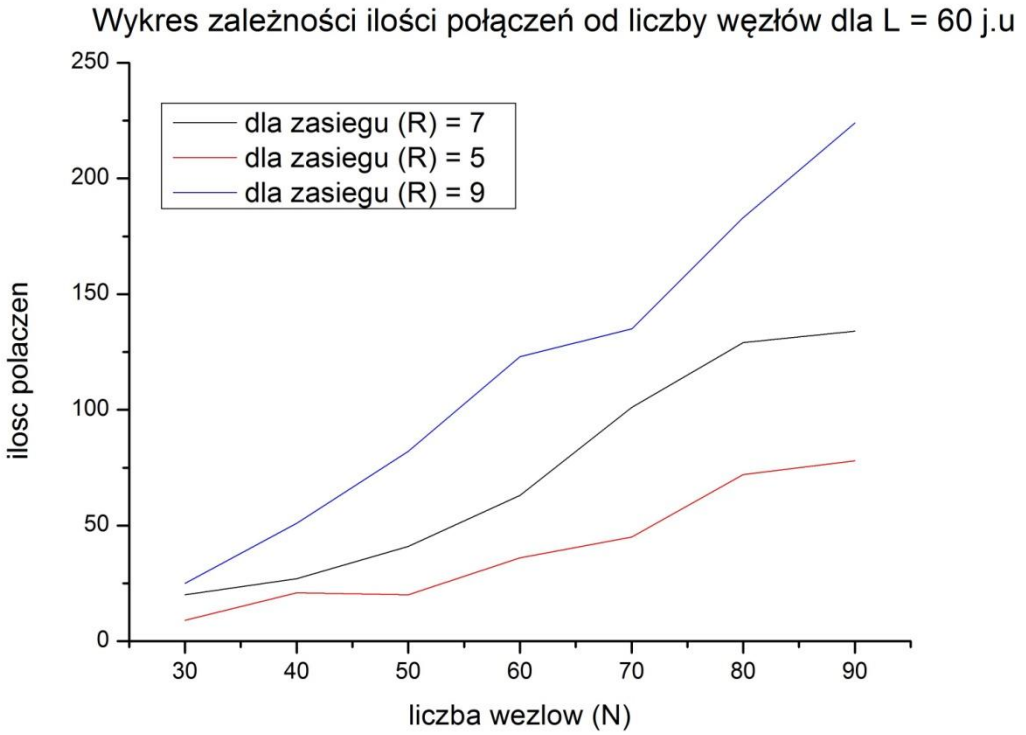


Rys.4.2 Wykres zależności ilości połączeń od stosunku zasięgu i wielkości obszaru symulacji.



Rys.4.3 Wykres zależności ilości połączeń od stosunku zasięgu i wielkości obszaru symulacji.

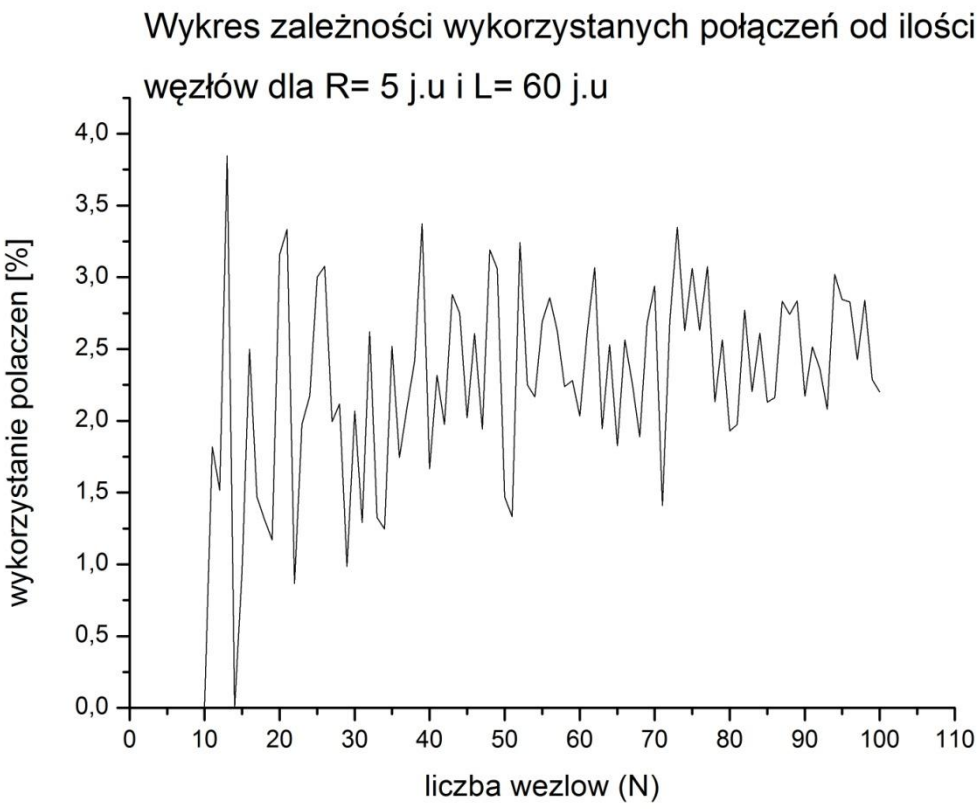
Otrzymane wyniki potwierdzają poprawność symulacji. Im stosunek zasięgu i wielkości obszaru symulacji jest większy, tym większa jest liczba połączeń w układzie. Oznacza to, że w miarę zwiększania zasięgu, bądź zmniejszania obszaru, ilość połączeń będzie rosła. W ramach uzupełnienia tej statystyki, poniżej przedstawiam wykres zależności ilości połączeń, w zależności od liczby węzłów dla zasięgu (R) = 5, 7 i 9 j.u oraz wielkości obszaru symulacji równej 60 x 60 j.u.



Rys.4.4 Wykres zależności ilości połączeń od liczby węzłów.

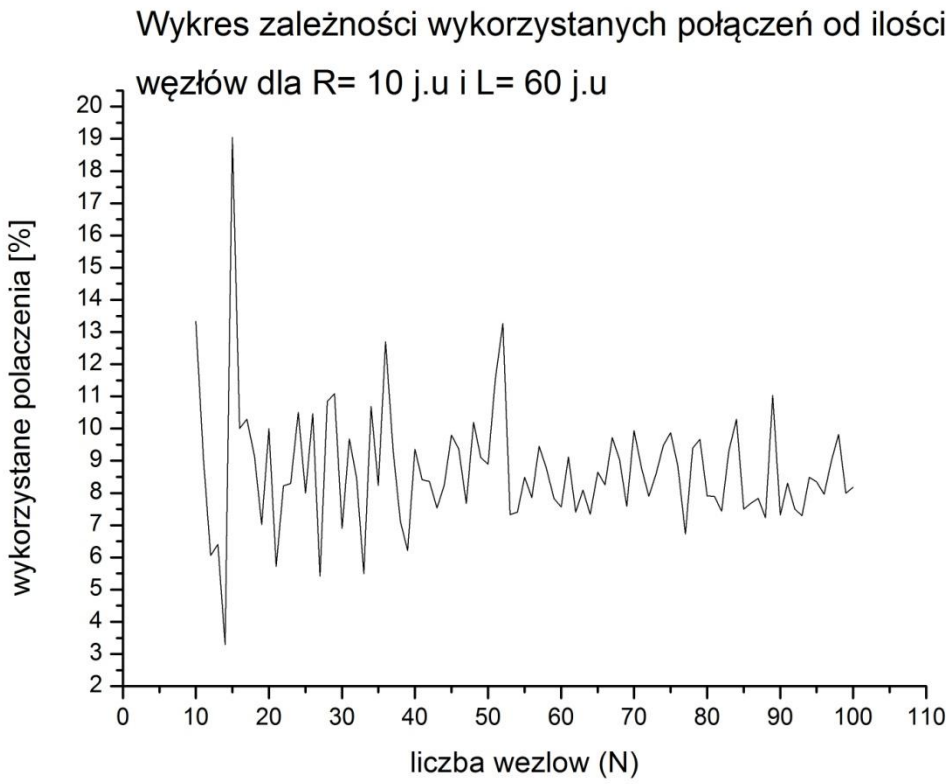
Tutaj również możemy zauważyć, iż w miarę zwiększania ilości węzłów, zasięgu, bądź obu tych czynników ilość połączeń będzie rosła.

Kolejnym etapem analizy sieci było określenie, jaki jest stopień wykorzystania połączeń w takim układzie. Do przedstawienia tego zagadnienia użyłem zależności, która przedstawia procent wykorzystanych połączeń od ilości węzłów w obszarze. Dzięki tej statystyce można określić, jaki jest procent wykorzystanych połączeń, tzn. jaki jest stosunek utworzonych połączeń w takiej sieci do wszystkich możliwych połączeń, jakie były by w przypadku kiedy wszystkie węzły były by ze sobą połączone. Wykorzystane połączenia, to w tym przypadku ilość wszystkich połączeń podzielona przez ilość wszystkich możliwych połączeń w układzie, liczona ze wzoru (N(N-1))/2, gdzie N to oczywiście liczba węzłów. Poniżej zamieszczam trzy wykresy przedstawiające tą zależność. Wykonane są one dla obszaru o rozmiarach 60 x 60 j.u oraz dla zasięgu (R) = 5,7 i 10 j.u.



Rys.4.5 Wykres zależności wykorzystanych połączeń od ilości węzłów.

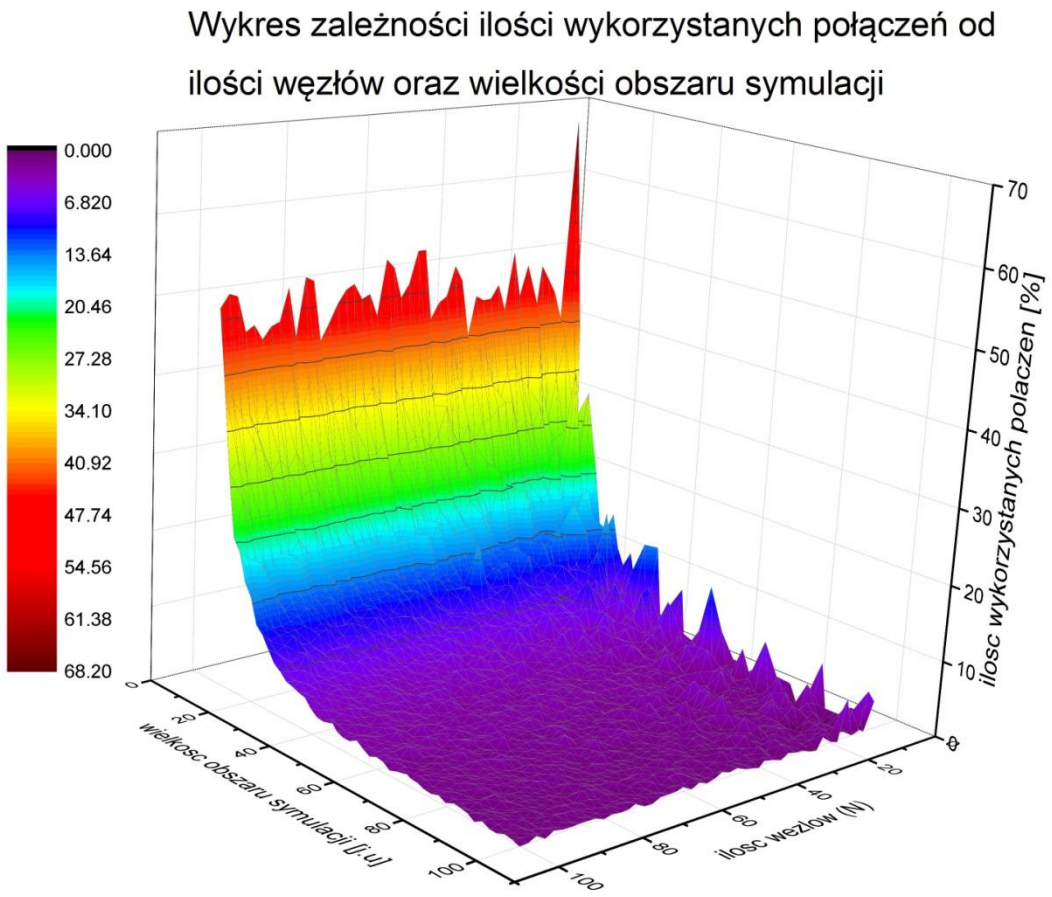
Rys.4.6 Wykres zależności wykorzystanych połączeń od ilości węzłów.



Rys.4.7 Wykres zależności wykorzystanych połączeń od ilości węzłów.

Widać wyraźnie, że wraz ze wzrostem zasięgu dla każdego węzła, wykorzystanie połączeń jest lepsze. Duży wpływ ma na to również powierzchnia obszaru, im będzie ona mniejsza tym większe będzie wykorzystanie połączeń ze względu na to, że węzły będą znajdować się bliżej siebie. W przypadku symulacyjnym, największy procent wykorzystanych połączeń wynosił ok. 19% dla zasięgu równego 10 j.u. Oczywiście, jeżeli zasięg byłby większy, również i procent uległby zwiększeniu. Jednak w tej pracy chciałem zasymulować tworzenie sieci w oparciu o sieci Bluetooth, których największy zasięg oscyluje wokół 10 metrów.

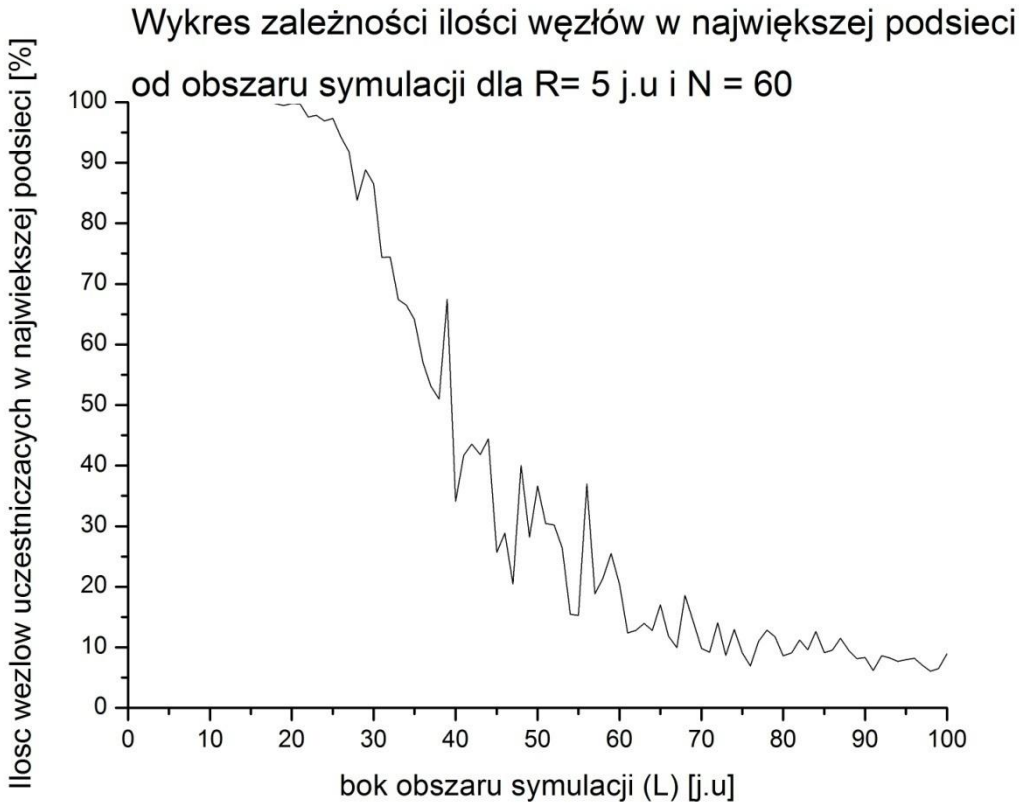
Na zakończenie tej analizy przedstawiam wykres 3D, który dokładnie pokazuje wykorzystanie połączeń w zależności od ilości węzłów (N) i wielkości obszaru (L).



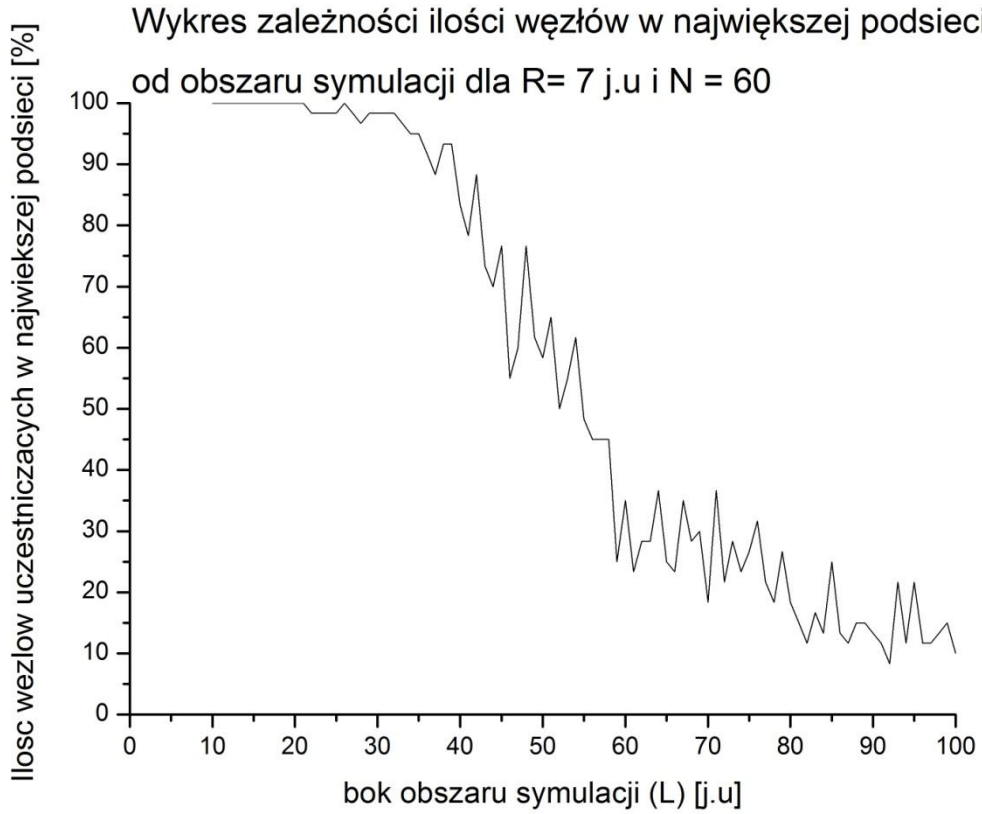
Rys.4.8 Wykres zależności wykorzystanych połączeń od ilości węzłów i wielkości obszaru symulacji.

Łatwo jest zauważyć, że największy procent wykorzystanych połączeń jest w przypadkach, kiedy ilość węzłów (N) jest duża, a obszar na którym się znajdują (L) jest mały.

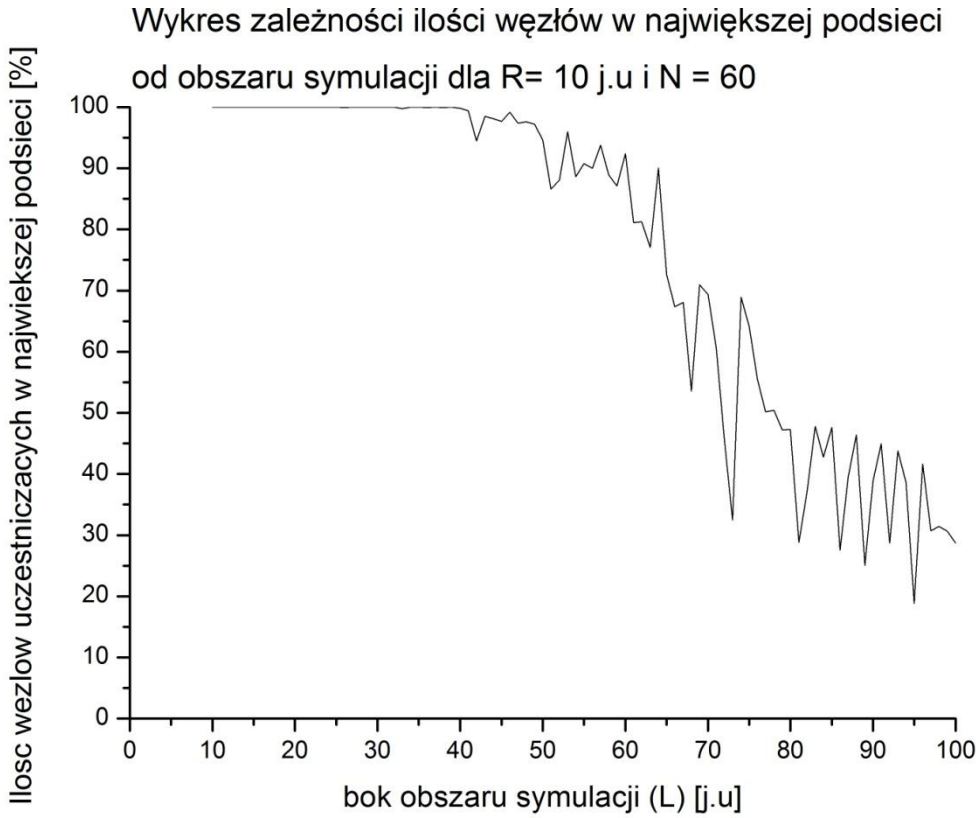
W dalszej części analizy otrzymanych wyników zająłem się sprawą występowania wielu składowych spójnych, czyli tzw. podsieci. Celem tej charakterystyki było pokazanie jaka ilość węzłów, ze wszystkich znajdujących się w obszarze, wchodzi w skład największej podsieci. Największa podsieć w układzie to ta składowa spójna, do której należy najwięcej wierzchołków. Do utworzenia takiej zależności potrzebowałem średniej liczby węzłów w największej podsieci, (którą otrzymałem z programu symulacyjnego), a także całkowitą liczbę węzłów w obszarze (N). Ilość węzłów uczestniczących w największej podsieci, to nic innego, jak stosunek średniej ilości węzłów występujących w największej podsieci oraz całkowitej liczby węzłów w obszarze. Poniżej przedstawiam wykresy, które dokładnie obrazują tą zależność. Wykonane są one dla 60 węzłów oraz zasięgu = 5, 7 i 10 j.u.



Rys.4.9 Wykres zależności ilości węzłów w największej podsieci od obszaru symulacji.



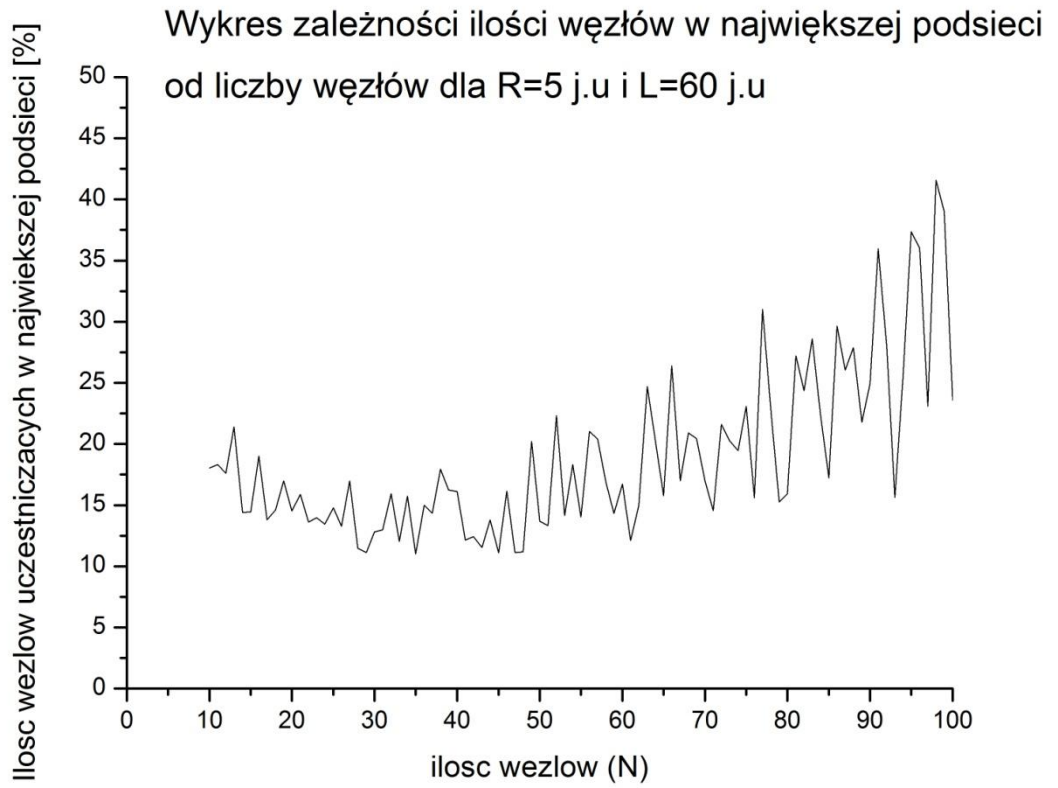
Rys.4.10 Wykres zależności ilości węzłów w największej podsieci od obszaru symulacji.



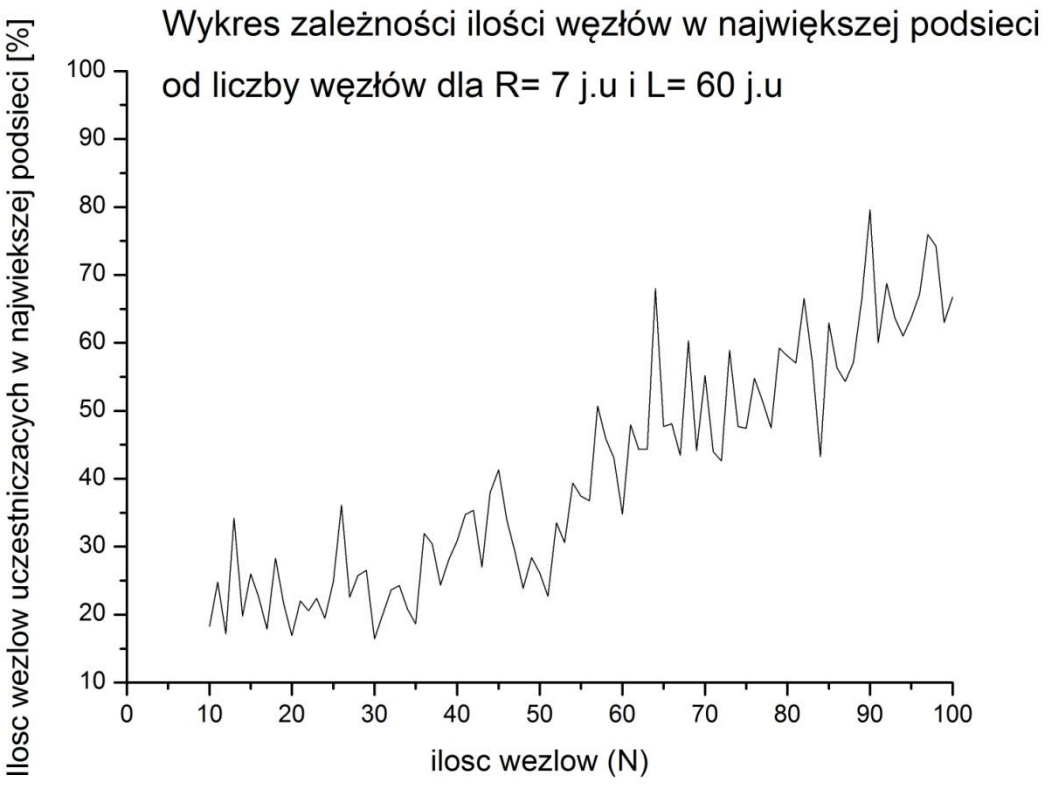
Rys.4.11 Wykres zależności ilości węzłów w największej podsieci od obszaru symulacji.

Charakterystyczne dla tych wykresów jest ich początkowe nasycenie, które zależy od zasięgu. Oznacza to, że w takiej sytuacji, w obszarze jest tylko jedna składowa spójna, zawierająca wszystkie węzły w układzie. Dla zasięgu (R) = 5 j.u, taka sytuacja jest dla obszarów 20 x 20 j.u i mniejszych, natomiast dla zasięgu równego 10 j.u, tylko jedna składowa spójna występuje dla obszarów 40 x 40 j.u i mniejszych. Wartym zauważenia jest również fakt, że spadek, który występuje po nasyceniu, jest bardziej gwałtowny dla małych zasięgów, a mniej gwałtowny dla większych. Doskonale pokazuje to zależność, że im większy jest zasięg tym większa jest ilość węzłów w największej podsieci.

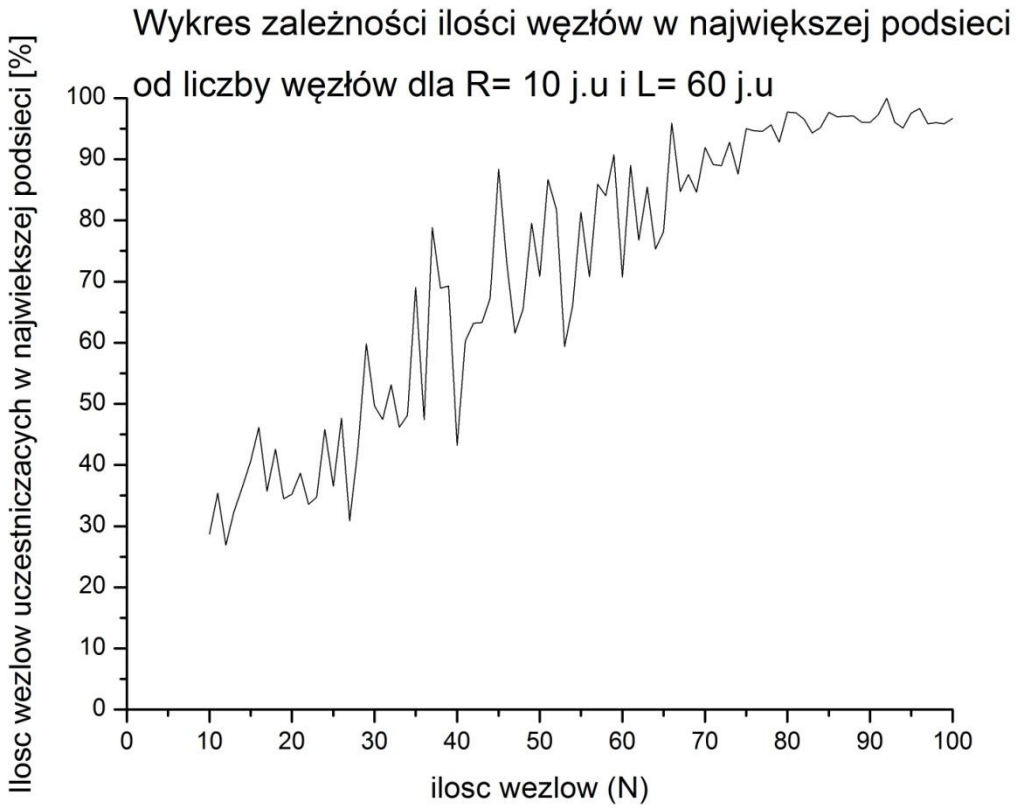
Poniżej zamieszczam także wykresy dla tej samej charakterystyki, w których w tym przypadku zmienia się ilość węzłów, a wielkość obszaru pozostaje stała i wynosi 60 x 60 j.u.



Rys.4.12 Wykres zależności ilości węzłów w największej podsieci od liczby węzłów.



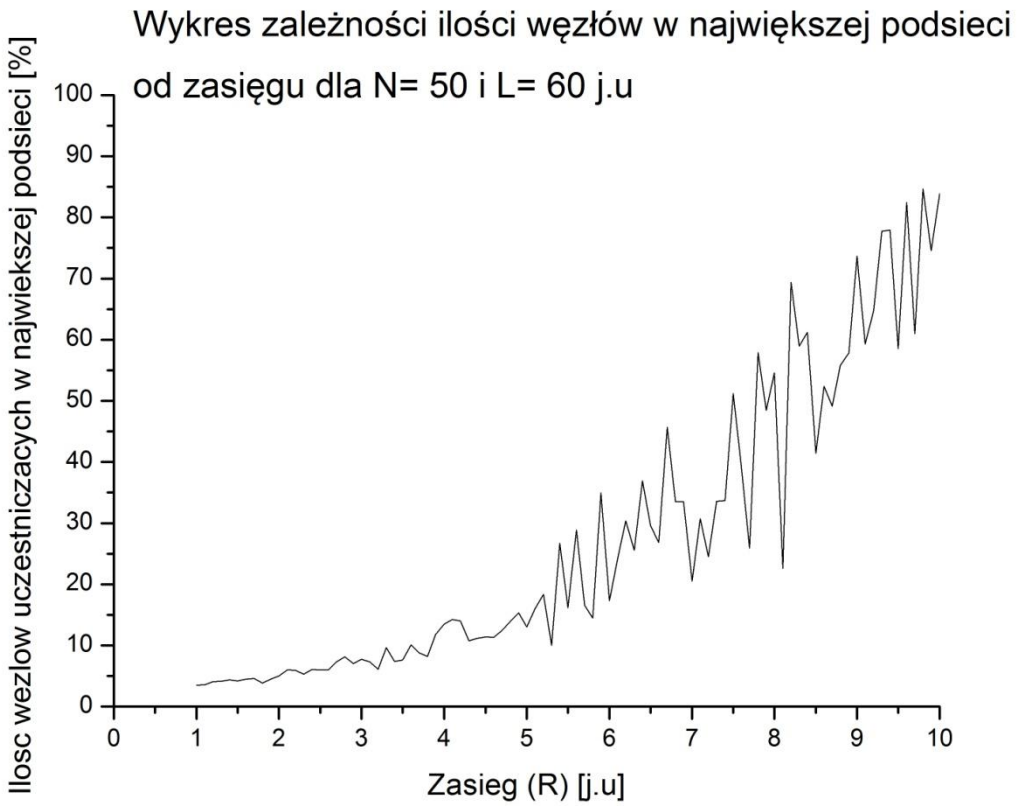
Rys.4.13 Wykres zależności ilości węzłów w największej podsieci od liczby węzłów.



Rys.4.14 Wykres zależności ilości węzłów w największej podsieci od liczby węzłów.

Można tutaj zauważyć wyraźne przesunięcie krzywej zależne od zasięgu. Dla małych zasięgów zaczyna się ona od mniejszych wartości na osi OY i widać mniejszy i bardziej łagodny wzrost w porównaniu z większymi zasięgami. W tym przypadku, duży wpływ na ilość węzłów w największej podsieci ma także całkowita liczba węzłów. Wyraźnie widać, że wraz z jej wzrostem wzrasta również ilość węzłów w największej podsieci, oczywiście jest to uzależnione także od zasięgu, im jest on większy, tym szybszy jest wzrost.

Dla dopełnienia tej charakterystyki przedstawiam również wykres dla stałej liczby węzłów (N) = 50 i stałego obszaru równego 60 x 60 j.u.



Rys.4.15 Wykres zależności ilości węzłów w największej podsieci od zasięgu.

Wykres ten jednoznacznie pokazuje, że ilość węzłów w największej podsieci zależy od zasięgu.

Analizując przedstawione charakterystyki, można z całą pewnością stwierdzić, że najważniejszym parametrem definiującym sieci jest zasięg. To od niego zależy, ile będzie połączeń w układzie, ile będzie składowych spójnych, a nawet jaki będzie procent wykorzystania połączeń. Ma on największy wpływ na zachowania sieci i jej ewolucję. Nie możemy jednak zapominać o parametrach odpowiedzialnych za wielkość obszaru i ilość węzłów. Ich prawidłowy dobór jest tak samo istotny. Wyobraźmy sobie sytuację, gdzie obszar symulacyjny jest bardzo duży, a znajduje się w nim tylko kilka węzłów. W takim przypadku, nawet duży zasięg nie pozwoli na ich połączenie. Widać więc wyraźnie, że odpowiedni dobór tych parametrów jest bardzo ważny i dopiero w takich warunkach modyfikacja zasięgu przynosi zamierzone rezultaty.

4.3. Zastosowanie.

Tak stworzony model sieci „ad-hoc” można powiązać z sieciami Bluetooth. Dzięki tym sieciom możliwa jest bezprzewodowa komunikacja oraz wymiana danych na niewielkie odległości w dowolnym miejscu na Ziemi. Dzięki tej technologii, urządzenia będące w zasięgu mogą się porozumieć bez konieczności używania kabli, wykorzystując do tego pojedynczy interfejs radiowy. Sieć Bluetooth została zaprojektowane z myślą o łączności bezprzewodowej na małe odległości. Średni zasięg takiej sieci to ok. 10 metrów[16].

Właśnie ze względu na to, wszystkie charakterystyki, które przedstawiłem w tej pracy są dla zasięgu mniejszego, bądź równego 10 metrów. Nie bez powodu symulacje sieci „ad-hoc” oparłem właśnie na tej technologii. Będę chciał rozwijać wykonaną przeze mnie pracę w celu stworzenia aplikacji mobilnej, która będzie łączyć urządzenia mobilne w sieć oraz zapewniać komunikację między nimi. Będę chciał tego dokonać przy pomocy środowiska Android Studio i w nim napisać kod źródłowy aplikacji, wykorzystując zaimplementowane protokoły sieci Bluetooth.

W takiej sytuacji wszystkie charakterystyki przedstawione w tej pracy będą aktualne, ale nie dla węzłów, lecz ludzi używających tak stworzonej aplikacji. Będzie możliwa wymiana informacji na dalekie odległości, która przechodzić będzie przez poszczególne urządzenia znajdujące się w zasięgu — należące do danej podsieci (składowej spójnej).

Technologie łączności przy użyciu sieci „ad-hoc” są obecnie w fazie intensywnych badań i rozwoju. Są stosowane w sytuacjach krytycznych - na terenach działań militarnych, klęsk żywiołowych lub w przypadku, gdy sieć komórkowa nie jest dostępna. Dlatego taki teoretyczny opis i symulator sieci jest niezwykle ważny do weryfikacji założeń dotyczących konstrukcji sieci.

5. Podsumowanie.

Przedstawiona praca zawiera opis wielu zagadnień związanych z szeroko rozumianym pojęciem sieci. Znalazło się tutaj wyjaśnienie podstawowych terminów z teorii grafów, właściwości sieci złożonych oraz rozkładów prawdopodobieństwa. Każde z tych zagadnień jest istotne z punktu widzenia tworzenia sieci „ad-hoc", jej struktury oraz ewolucji. Dodatkowo został także przeprowadzony przegląd teorii grafowych, dotyczących zagadnienia samoorganizujących się sieci.

Dokonano również selekcji użytecznych algorytmów grafowych dla zagadnień opisanych w tej pracy. Użyte zostały one także w programie symulacyjnym. Dzięki zastosowaniu tych algorytmów, otrzymane wyniki oraz ich interpretacje były bardziej dokładne, co pomogło w odwzorowaniu rzeczywistych zachowań sieci.

Wszystkie te elementy, zarówno teoretyczne jak i praktyczne, w dużej mierze przyczyniły się do napisania programu symulacyjnego, który bardzo dokładnie obrazował zachowanie się sieci na danym obszarze. Oparty był także o zadane prawa rozkładu prawdopodobieństwa, które pozwalały na ewolucję sieci i zmianę jej struktury. Dzięki tak napisanej symulacji, otrzymano wiele zależności charakteryzujących samoorganizujące się sieci.

Uzyskane charakterystyki jednoznacznie pokazują, że najważniejszymi parametrami, przebadanymi przeze mnie, dla sieci „ad-hoc" są: ilość węzłów znajdujących się na danym obszarze, wielkość tego obszaru oraz zasięg jaki ma każdy z węzłów. To właśnie od tych trzech warunków zależy, jak dana sieć będzie się tworzyć, a także jakie będzie miała warunki do rozwoju. W tej pracy przedstawiono szereg charakterystyk zależnych od każdego z wymienionych parametrów. Łatwo można zauważyć, że zmiana jednego z nich ma znaczny wpływ na strukturę sieci, a co za tym idzie, przy tworzeniu takich układów zawsze należy pamiętać o ich odpowiednim doborze.

Przeprowadzone badania oraz analizy dla samoorganizujących się sieci „ad-hoc" pokazały, że sieci te mogą znaleźć szerokie zastosowanie w wielu dziedzinach życia. Łatwość ich tworzenia oraz przesyłania informacji może sprawić, że sieci te uzyskają miano jednych z najważniejszych typów sieci przyszłych czasów- dla zastosowań kryzysowych

6. Bibliografia.

[1]. A. Fronczak, P. Fronczak, *Świat sieci złożonych. Od fizyki do internetu* PWN, Warszawa 2009.

[2]. http://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-article-BSW4-0103-0012, (Dostęp lipiec 2016).

[3]. http://allegro5tutorial.blogspot.com, (Dostęp lipiec 2016).

[4]. http://www.originlab.com/index.aspx?go=Company/NewsAndEvents/PressRoom&pid=1771, (Dostęp lipiec 2016).

[5]. J. Wojciechowski, K. Pieńkosz *Grafy i sieci* PWN, Warszawa 2013.

[6]. http://iair.mchtr.pw.edu.pl/~bputz/aisd_cpp/lekcja7/segment1/main.htm, (Dostęp lipiec 2016).

[7]. http://www.algorytm.org/algorytmy-grafowe/przeszukiwanie-grafu-wszerz-bfs-i-w-glab-dfs.html, (Dostęp lipiec 2016).

[8]. http://www.if.pw.edu.pl/~agatka/moodle/spoleczne.html, (Dostęp lipiec 2016).

[9]. http://www.algorytm.org/klasyczne/grafy-i-ciag-dalszy.html, (Dostęp lipiec 2016).

[10]. http://www.mif.pg.gda.pl/nkm/files/grafylos.pdf, (Dostęp lipiec 2016).

[11]. http://www.neuroinf.pl/Members/danek/swps/Article.2006-06-16.5517/getFile, (Dostęp lipiec 2016).

[12]. http://szemek.github.io/networks/#/8, (Dostęp lipiec 2016).

[13]. http://home.agh.edu.pl/~jurczyk/wms/RozkladGaussa.pdf, (Dostęp lipiec 2016).

[14]. http://www.eduscience.pl/artyku%C5%82y/loty-levy-ego-czyli-jak-poluj%C4%85-albatrosy, (Dostęp lipiec 2016).

[15]. https://www.gnu.org/software/gsl/manual/gsl-ref.pdf str. 323, (Dostęp lipiec 2016).

[16]. A. M. Brent, Ch. Bisdikian , Ph. D. *Bluetooth uwolnij się od kabli* Wydawnictwo Helion, Gliwice 2003.

[17].https://upload.wikimedia.org/wikipedia/commons/thumb/d/d3/Graph_theory_tree.svg/200px-Graph_theory_tree.svg.png, (Dostęp lipiec 2016).

[18]. https://upload.wikimedia.org/wikipedia/commons/thumb/2/28/6n-graf2.svg/250px-6n-graf2.svg.png, (Dostęp lipiec 2016).

[19]. http://www.algorytm.org/klasyczne/grafy-i-ich-reprezentacje.html, (Dostęp lipiec 2016).

[20]. http://kochanowski.iq.pl/~mim/wiki-image/graf.gif, (Dostęp lipiec 2016).

[21]. http://www.if.pw.edu.pl/~agatka/moodle/rys/strongties.jpg, (Dostęp lipiec 2016).

[22]. http://www.if.pw.edu.pl/~agatka/moodle/spoleczne.html, (Dostęp lipiec 2016).

[23]. http://www.if.pw.edu.pl/~agatka/moodle/modele_KS_ER.html, (Dostęp lipiec 2016).
[24].https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/Normal_distribution_pdf.png/400px-Normal_distribution_pdf.png, (Dostęp lipiec 2016).
[25]. http://www.eduscience.pl/artyku%C5%82y/loty-levy-ego-czyli-jak-poluj%C4%85-albatrosy, (Dostęp lipiec 2016).

Appendix:

Poniżej zmieściłem kod programu symulacyjnego w celu weryfikacji.

```
#include <iostream>
#include <fstream>
#include <cmath>

#include <vector>
#include <random>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include<algorithm>
#include <stdio.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_rng.h>
#include <sys/time.h>

#define nieskonczonosc 1000
using namespace std;

const double pi = 3.141592653589793;

class compute
{

public:
   int N;
   int L ;
   double R ;

   int **neib;

   float dist (double c, double alpha) {

   const gsl_rng_type * T;
   gsl_rng * r;
   gsl_rng_env_setup();
   struct timeval tv;
   gettimeofday(&tv,0);
   unsigned long mySeed = tv.tv_sec + tv.tv_usec;
   T = gsl_rng_default;
   r = gsl_rng_alloc (T);
   gsl_rng_set(r, mySeed);
   double u;
            u = gsl_ran_levy(r,c,alpha);
   gsl_rng_free (r);
   return (float)u;
}
   void neighbour(double *x,double *y,double SIGMA)
   {
      double rijX,rijY,R_sq;
      double rijX2,rijY2,Circ_2;
      Circ_2=SIGMA; // licbza z przodu duza, gdy system jest rzadki
      for(int is=0; is<N; is++)
      {
         for(int js=0; js<N; js++)
         {
            rijX=x[is]-x[js];
            rijY=y[is]-y[js];

            rijX2=rijX*rijX;
            rijY2=rijY*rijY;
            R_sq=sqrt(rijX2+rijY2);

            if(R_sq<Circ_2 ) neib[is][js]=1;
            else neib[is][js]=0;
         }
      }
   }

   void randXY(double *x, double *y, int n, double a)
   {
      for(int i=0; i<n; i++)
```

```
    {

        x[i] = a*((rand()%RAND_MAX) / (1.0 * RAND_MAX));
        y[i] = a*((rand()%RAND_MAX) / (1.0 * RAND_MAX));
    }
}

void randv(double *x, double *y, int n, double a,double b)
{
    for(int i=0; i<n; i++)
    {

        x[i] = dist(5.0,2.0);
        y[i] = dist(5.0,2.0);
    }

    double maxx=0;
    double maxy=0;

    for(int i=0; i<n; i++)
    {
        if(x[i] >maxx)maxx = x[i];
        if(y[i] >maxy)maxy = y[i];
    }
    if (maxx>maxy)maxy=maxx;
    else maxx=maxy;

    for(int i=0; i<n; i++)
    {

        x[i]/= maxx;
        y[i]/=maxy;

        x[i] =2*b*x[i] - b;
        y[i] =2*b*y[i] - b;
    }
    fstream plik;
    plik.open("velocity_data.txt");

    for (int h=0; h<N; h++)
    {
        plik<<h<<"\t"<<x[h]<<"\t"<<y[h]<<endl;

    }
    plik.close();
}

void write(double *x,double *y, int N)
{
    for (int i=0; i<N; i++)
        cout<<x[i]<<"\t"<<y[i]<<endl;
}

void writeGrid()
{
    fstream plik_MAC;
    plik_MAC.open("Macierz.txt",ios::out| ios::app);
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<N; j++)
        {
            if(i==j)
                plik_MAC <<"X|";
            else
                plik_MAC<<neib[i][j]<<"|";
        }
        plik_MAC<<endl;
    }
}

void GetPartData(int parN)
{
    int nN=0;
    for (int i=0; i<N; i++)
        if (neib[parN][i]!=0 && parN !=i) nN++;
    cout<<"czastka nr "<<parN<<" ma "<<nN<<" sasiadow"<<endl;
}
```

```
   int CountConn()
   {
      int allConn =0;
      for (int i=0; i<N; i++)
         for(int j=i; j<N; j++)
            if (neib[i][j]!=0 && j!=i) allConn++;
      return allConn;
   }
//================================================================================
==================

   int    * D;
   class   cGraf
   {

   public:

      int     **A;
      cGraf(int n)
      {
         A = new int*[n];
         for(int i =0;i<n;i++)
            A[i]=new int [n];
      }
      int n;

      vector<int> Nastepniki(int x)
      {
         vector<int> wynik;

         for (int i=0; i<n; i++)
            if ((A[x][i]!=nieskonczonosc)&&(A[x][i]!=0))
               wynik.push_back(i);

         return(wynik);
      }
   };

   class cWierzcholek
   {

   public:
      int V;
      int* Odl;

      friend bool operator < (const cWierzcholek &p1, const cWierzcholek &p2)
      {
         return(*(p1.Odl)>*(p2.Odl));
      }
   };

   void spjp(int jj)
   {
      FILE               *plik;
      char               s[5];
      int               i,j,k;
      cGraf              Graf(N);

      vector<cWierzcholek>      Q;
      cWierzcholek           wierzcholek;
      vector<int>           nastepniki;

//============================MAIN=====================================
      Graf.n=N;

      for ( int i=0; i<N; i++ )
      {
         for ( int j=0; j<N; j++ )
         {
//             cout<<neib[i][j]<<"|";
            if (neib[i][j]==1 && i!=j) Graf.A[i][j]=1;
            if (neib[i][j]==0 && i!=j) Graf.A[i][j]=nieskonczonosc;
            if (i==j) Graf.A[i][j]=0;
//             cout<<Graf->A[i][j]<<"|";
         }
         //cout<<endl;
      }
```

```
        for (i=0; i<Graf.n; i++)
        {
            D[i]=Graf.A[jj][i];
            wierzcholek.V=i;
            wierzcholek.Odl=&D[i];
            Q.push_back(wierzcholek);
        }

        vector<cWierzcholek>::iterator it;

        make_heap(Q.begin(),Q.end(),less<cWierzcholek>());

        wierzcholka wynosi 0)
        pop_heap(Q.begin(),Q.end());
        Q.pop_back();

        while (Q.empty()!=true)
        {
            make_heap(Q.begin(),Q.end(),less<cWierzcholek>())  ;
            wierzcholek=Q[0];
            pop_heap(Q.begin(),Q.end());
            Q.pop_back();
            nastepniki=Graf.Nastepniki(wierzcholek.V);
            for (i=0; i<nastepniki.size(); i++)
                D[nastepniki[i]]=min(D[nastepniki[i]],D[wierzcholek.V]+Graf.A[wierzcholek.V][nastepniki[i]]);
        }
    }

    vector<vector<int> > znajdz_podsiec(int ** neib, int N)
    {
        vector<vector<int> > output;
        vector<int> buff;

        bool * wezly;
        wezly = new bool [N];

        for (int i =0; i<N; i++)
            wezly[i] = true;

        bool go = true;

        while(go)
        {
            go=false;
            for (int i=0; i<N; i++)
            {
                if(wezly[i])go=true;

            }

            for(int i=0; i<N; i++)
            {
                if (wezly[i])
                {
                    spjp(i);

                    for(int j=0; j<N; j++)
                    {
                        if (D[j]<nieskonczonosc && D[j]>=0)
                        {
                            buff.push_back(j);
                            wezly[j] =false;
                        }
                    }
                    output.push_back(buff);
                    buff.erase(buff.begin(),buff.end());
                }
            }
        }
        return (output);
    }

//-=--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

    compute(int n,int l ,double r)
    {
```

```
    N = n;
    L = l;
    R = r;

    neib = new int*[n];
    for (int i =0;i<N;i++)
        neib[i] = new int[n];

    D = new int [n];
}

int operate()
{
    int size_x = 800;
    int size_y = 800;

    srand(time(NULL));

    double *x;
    double *y;
    x= new double [N];
    y= new double [N];

    double a_x,b_x,a_y,b_y;

    double *vx;
    double *vy;
    vx= new double [N];
    vy= new double [N];
    randv(vx,vy,N,1.3,0.1);
    randXY(x,y,N,L);

    neighbour(x,y,R);

    int iter =0;

    fstream plik_ONE,plik_TWO,plik_THREE,plik_Four;
    plik_ONE.open("ilscPolPoCzas.txt",ios::out);
    plik_TWO.open("gestosc.txt",ios::out | ios::app);
    plik_THREE.open("zimiennepudlo.txt",ios::out | ios::app);
    plik_Four.open("nnnj.txt",ios::out | ios::app);

    double two =CountConn();
    //plik_TWO<<N<<"\t"<<two/(L*L)<<endl;
    plik_TWO<<N<<"\t"<<two<<endl;
    plik_THREE<<L<<"\t"<<R<<"\t"<<R/L<<"\t"<<CountConn()<<endl;


    int ** neib_template;
    neib_template = new int *[N];
    for (int g=0; g<N; g++) neib_template[g] = new int [N];

    for(int g=0; g<N; g++)
        for(int h=0; h<N; h++)
            neib_template[g][h] = neib[g][h];

    //=================================================================
    vector<vector<int> > podsiec;
podsiec = znajdz_podsiec(neib_template,N);


for(int k =0;k<podsiec.size();k++)
{
    cout<<"podsiec nr "<<k<<"::";
    for(int t=0;t<podsiec[k].size();t++)
        cout<<podsiec[k][t]<<";";
    cout<<endl;
}

    bool start = false;
    int najwieksza_podsiec;
    int sumowanie = 0;
    int iterator_sum=0;
    for(int hh = 0 ;hh<1000;hh++)
    {
        ALLEGRO_EVENT ev;
```

```
        al_wait_for_event(event_queue,&ev);
        if(ev.type==ALLEGRO_EVENT_TIMER)
        {
           redraw=true;
        }
        else if(ev.type==ALLEGRO_EVENT_DISPLAY_CLOSE)
        {

           break;
        }
        else if(ev.type == ALLEGRO_EVENT_KEY_DOWN && ev.keyboard.keycode == ALLEGRO_KEY_ESCAPE)
        {
           break;
        }
        else if(ev.type == ALLEGRO_EVENT_KEY_DOWN && ev.keyboard.keycode == ALLEGRO_KEY_S)
        {
           start= true;
        }
        else if(ev.type == ALLEGRO_EVENT_KEY_DOWN && ev.keyboard.keycode == ALLEGRO_KEY_P)
        {
           start= false;
           podsiec = znajdz_podsiec(neib_template,N);

           for(int k =0; k<podsiec.size(); k++)
           {
              cout<<"podsiec nr "<<k<<"::";
              for(int t=0; t<podsiec[k].size(); t++)
                 cout<<podsiec[k][t]<<";";
              cout<<endl;
           }

           writeGrid();
        }

        if(/*start && redraw && al_is_event_queue_empty(event_queue)*/true)
        {
           redraw = false;

           double temp_x =0.0;
           double temp_y = 0.0;
           al_set_target_bitmap(bitm);
           al_clear_to_color(kBLACK);


           for(int i=0; i<N; i++)
           {
              x[i]+=vx[i];
              y[i]+=vy[i];
           }

           //warunki brzegowe
           for(int i = 0; i<N; i++)
           {
              if(x[i]>=L)vx[i]=-vx[i];
              if(x[i]<=0.)vx[i]=-vx[i];
              if(y[i]>=L)vy[i]=-vy[i];
              if(y[i]<=0.)vy[i]=-vy[i];

           }


           for(int i = 0; i < N; i++)
           {
              //shape[i].setPosition(x+i*10,y+i*10);

              temp_x=(int)((x[i]/L)*size_x) ;
              temp_y=(int)((y[i]/L)*size_y);
              if(i==6) al_draw_filled_circle(temp_x,temp_y,7.,kRED);
              else al_draw_filled_circle(temp_x,temp_y,7.,kGREEN);

           }
//=====================================================

           neighbour(x,y,R);
           for(int g=0; g<N; g++)
              for(int h=0; h<N; h++)
                 neib_template[g][h] = neib[g][h];
```

```
//=========================================================
            for(int i=0; i<N; i++)
               for(int j=0; j<N; j++)
               {
                  if(neib[i][j] ==1)
                  {
                     a_x=(int)((x[i]/L)*size_x) ;
                     a_y=(int)((y[i]/L)*size_y);
                     b_x=(int)((x[j]/L)*size_x) ;
                     b_y=(int)((y[j]/L)*size_y);
                     al_draw_line(a_x,a_y,b_x,b_y,kYELLOW,2.);
                  }
               }


            al_set_target_bitmap(al_get_backbuffer(display));

            al_draw_bitmap(bitm,0,0,0);
            al_flip_display();

            if(iter%10)plik_ONE<<iter<<"\t"<<CountConn()<<endl;

            if(iter%10)
            {
               iterator_sum+=1;
               podsiec = znajdz_podsiec(neib_template,N);
               najwieksza_podsiec=0;
               for(int k =0; k<podsiec.size(); k++)
               {
                  //cout<<"podsiec nr "<<k<<"::";
                  if(najwieksza_podsiec<podsiec[k].size()) najwieksza_podsiec=podsiec[k].size();

                  //cout<<podsiec[k][t]<<";";
                  //plik_Four<<k<<endl;

               }
               sumowanie+=najwieksza_podsiec;
               // plik_Four<<sumowanie<<endl;

            }
            //cout<<najwieksza_podsiec<<endl;

            iter+=1;
         }
      }

      double sum;
      sum = (double)sumowanie/iterator_sum;
      //plik_Four<<"N R L irS Sr AllConn"<<endl;
      plik_Four<<N<<"\t"<<R<<"\t"<<L<<"\t"<<iterator_sum<<"\t"<<sum<<"\t"<<CountConn()<<endl;
//       al_destroy_timer(timer);
//       al_destroy_display(display);
//       al_destroy_event_queue(event_queue);
      plik_ONE.close();
      plik_THREE.close();
      plik_TWO.close();
      plik_Four.close();
      int cc = CountConn();
      return cc;
   }

};

main(int argc, char **argv)
{
   fstream ppp;
   int l,n;
   double r;
   l=atoi(argv[1]);
   n=atoi(argv[2]);
   r=atof(argv[1]);
            r=r/10;

   ppp.open("3d.txt",ios::out |ios::app);
   double out;
   int temp;
   double T;
```

```
                        cout<<r<<" _R!"<<endl;
        out = 0;
        compute symulation (n,l,r);
        temp = symulation.operate();
        out = n*(n-1);
              T = temp/out;
        ppp<<n<<"\t"<<l<<"\t"<<T<<"\t"<<temp<<"\t"<<out<<endl;
        cout<<n<<";"<<l<<":"<<r<<" _done!"<<endl;
    ppp.close();
    return 0;
}
```