

Spis treści

1 Wstęp	3
2 Przegląd literatury	5
2.1 Cele badania przestrzeni rozwiązań	5
2.2 Próbkowanie przestrzeni rozwiązań	6
2.3 O przestrzeni rozwiązań	8
2.3.1 Sieć optimów lokalnych	8
2.3.2 Wierzchołki	8
2.3.3 Krawędzie	9
2.3.4 Struktury lejowe	10
2.3.5 Metryki przestrzeni rozwiązań	10
2.3.6 Problem komiwojażera	12
2.3.7 Operacja 2-exchange	12
3 Badania eksperymentalne	15
3.1 Zaimplementowane algorytmy	15
3.1.1 Próbkowanie dwufazowe	15
3.1.2 Snowball	15
3.1.3 Przegląd zupełny	18
3.2 Instancje testowe	19
3.3 Opis eksperymentu	21
3.4 Wyniki	21
3.4.1 Porównanie wartości metryk dla małych instancji	21
3.4.2 Badanie stabilności	22
4 Opis implementacji	51
4.1 Program próbujący	51
4.1.1 Uruchamianie i parametry	51
4.1.2 Pliki wejściowe i wyjściowe	52
4.2 Program obliczający wartości miar	54
4.2.1 Uruchamianie i parametry	54
4.2.2 Pliki wejściowe i wyjściowe	54
4.3 Generatory instancji testowych	54
4.3.1 Pliki wyjściowe	55
4.4 Skrypty pomocnicze	55
4.5 Wykorzystane biblioteki	56
5 Podsumowanie	57
Literatura	57

Rozdział 1

Wstęp

Rozdział 2

Przegląd literatury

2.1 Cele badania przestrzeni rozwiązań

Analiza przestrzeni rozwiązań pozwala na lepsze poznanie niektórych cech problemu, a także zbadanie, w jakim stopniu cechy te zależne są od rodzaju badanej instancji. Przedmiotem badań z tej dziedziny najczęściej są znane problemy NP-trudne, takie jak problem komiwojażera czy problem kwadratowego przydziału. Jednym z proponowanych zastosowań analizy przestrzeni rozwiązań jest wykorzystanie jej do wyboru najlepszego algorytmu heurystycznego dla danej instancji problemu. W pracy *Local Optima Networks in Solving Algorithm Selection Problem for TSP*[1] sieci optimów lokalnych wygenerowane na podstawie próbkowania przestrzeni rozwiązań wykorzystano do nauczenia modeli regresji, których zadaniem było przewidzenie, który algorytm heurystyczny da lepsze wyniki dla danej instancji problemu. Badanie wykazało, że analiza przestrzeni rozwiązań może zstać z powodzeniem wykorzystana w tym celu. Problemem pozostaje natomiast długi czas trwania próbkowania przestrzeni, zwykle dłuższy niż czas działania samego algorytmu heurystycznego. W pracy *Mapping the global structure of TSP Fitness landscapes*[7] zbadano przestrzeń rozwiązań dla różnych instancji problemu komiwojażera. Zauważono, że instancje wygenerowane losowo zwykle mają mniejszą neutralność i mniej globalnych optimów od instancji rzeczywistych. Zaobserwowano również, że sposób rozłożenia miast (w klastrach, równomierny) wpływa na wzajemną korelację wartości różnych miar.

W ostatnich latach w podobny sposób przeprowadzono analizy przestrzeni konfiguracji wieloparametrowych algorytmów optymalizacyjnych. W pracy *Understanding Parameter Spaces using Local Optima Networks: A Case Study on Particle Swarm Optimization*[2] wykorzystano sieci lokalnych optimów, oraz pochodne struktury CMLON do analizy i wizualizacji przestrzeni parametrów algorytmu roju cząstek. Analiza wykazała istnienie dużej ilości lokalnych optimów, niską neutralność, oraz istnienie wielu ścieków (ang sinks) nie znajdujących się w optimum globalnym. Sugeruje to, że naiwne metody dobierania parametrów mogą łatwo doprowadzić do suboptimalnej konfiguracji i w efekcie nie otrzymania najlepszych wyników. Podobne badanie wykonano dla przestrzeni parametrów procesu AutoML w pracy *Understanding AutoML Search Spaces with Local Optima Networks*[10]. AutoML jest procesem automatyzacji konfiguracji procesów (ang. pipelines) uczenia maszynowego obejmującym m. in. wybór zastosowanego przetwarzania wstępnego, właściwego algorytmu uczenia, oraz jego hiperparametrów. W pracy zbadano przestrzeń konfiguracji AutoML dla zadania klasyfikacji. Jako funkcję celu przyjęto uśrednioną wartość metryki F-score dla danej konfiguracji procesu. Wartości F-score uzyskano poprzez przetestowanie procesu na kilku zbiorach danych. Dla każdej z badanych przestrzeni konfiguracji utworzono trzy sieci LON. Każda z sieci była oparta o inny mo-

del krawędzi. Przeszukano wszystkie możliwe rozwiązania, ale ze względu na złożoność obliczeniową sieć LON budowano poprzez przeglądanie ograniczonego sąsiedztwa każdego z wierzchołków. Badanie wykonano dla kilku wielkości sąsiedztwa - 20, 30, 50 i 100 sąsiadów. Zauważono wpływ modelu krawędzi na powstałą sieć - w sieciach z krawędziami typu basin-transition nie zauważono obecności ścieków, natomiast istniało wiele źródeł. Sieci oparte o escape edges miały z kolei dużo ścieków i mało źródeł. W przypadku perturbation edges w sieci nie było ścieków ani źródeł niezależnie od rozmiaru sąsiedztwa. Zauważono, że w badanych przypadkach optima globalne skupiały się w pewnym rejonie przestrzeni rozwiązań, w niewielkiej od siebie odległości. Nie były rozłożone równomierne. Sugeruje to, że w problemie konfiguracji procesu uczenia maszynowego, najlepsze ze wszystkich możliwych konfiguracji niewiele się od siebie różnią.

2.2 Próbkowanie przestrzeni rozwiązań

Przegląd zupełny przestrzeni rozwiązań problemów trudnych obliczeniowo jest w praktyce niemożliwy, poza bardzo małymi instancjami problemu. Z tego powodu analizę przestrzeni rozwiązań przeprowadza się na części przestrzeni zbadanej w procesie próbowania. W tym miejscu pojawiają się pytanie: w jaki sposób próbować przestrzeń, aby cechy jej zbadanego fragmentu jak najlepiej odzwierciedlały cechy całej przestrzeni?

Próbkowanie przestrzeni rozwiązań wykonuje się zwykle poprzez zastosowanie pewnej odmiany iteracyjnego przeszukiwania lokalnego (ang. ILS - Iterated Local Search), a wyniki zapisuje się w postaci sieci optimów lokalnych. Najczęściej stosowane algorytmy próbowania oparte o ILS można podzielić na trzy kategorie: Próbkowanie dwufazowe, *Markov-chain* oraz *Snowball*.

Próbkowanie dwufazowe składa się z dwóch faz - najpierw próbowane są wierzchołki poprzez zastosowanie metody iteracyjnego przeszukiwania lokalnego[8]. Metoda ta polega na generowaniu losowego rozwiązania, a następnie wykorzystaniu algorytmu optymalizacji lokalnej do znalezienia najbliższego lokalnego optimum. Następuje po tym wylosowanie nowego punktu i powtórzenie procedury. Druga faza to faza próbowania krawędzi, polegająca na poddaniu znalezionych wcześniej rozwiązań operacji perturbacji, a następnie wykonaniu optymalizacji lokalnej. Jeśli otrzymane w ten sposób rozwiązanie jest innym lokalnym optimum znajdującym się w zbiorze wierzchołków, to dodawana jest odpowiednia krawędź. Metoda ta po raz pierwszy została zaprezentowana w pracy[4] do analizy problemu kwadratowego przypisania. W pracy[1] algorytm oparty o tę samą metodę został wykorzystany do badania przestrzeni problemu komiwojażera. Zastosowana tam odmiana wykorzystuje algorytm 2-opt do optymalizacji lokalnej i procedurę *2-exchange* jako operację perturbacji.

Metoda *Markov-chain* została po raz pierwszy zaprezentowana w pracy [6]. Zaczyna się od wybrania losowego rozwiązania i jego optymalizacji. Następnie otrzymane optimum lokalne zostaje poddane operacji perturbacji i otrzymywane w ten sposób rozwiązanie staje się nowym punktem startowym. Procedura jest powtarzana przez określoną liczbę iteracji, z każdą iteracją zapisywane są informacje o nowych krawędziach i wierzchołkach. Metoda została wykorzystana w pracy[7] do badania przestrzeni problemu komiwojażera. Do optymalizacji lokalnej wykorzystany został algorytm Lin-Kerninghan, a jako operacja perturbacji procedura *Double-bridge*.

Metoda *Snowball* składa się z dwóch etapów wykonywanych naprzemiennie. Pierwszy polega na kilkukrotnym poddaniu rozwiązania początkowego perturbacji w celu uzyskania sąsiednich rozwiązań, a następnie ich optymalizacji. Procedura jest rekurencyjnie po-

wtarzana dla znalezionych w ten sposób lokalnych optimów aż do osiągnięcia pewnej z góry ustalonej głębokości. Jedno z lokalnych optimów sąsiadujących z rozwiązaniem początkowym zostaje wybrane jako rozwiązanie początkowe kolejnej iteracji. Dodatkowo w pamięci przechowywana jest lista rozwiązań początkowych - jeśli rozwiązanie już się w nim znajduje, to nie może być wybrane ponownie. Jeśli nie istnieje rozwiązanie sąsiednie spełniające ten warunek, za kolejne rozwiązanie początkowe przyjmowane jest lokalne optimum otrzymane z optymalizacji rozwiązania losowego. Algorytm *Snowball* wywodzi się z technik wykorzystywanych w badaniach z dziedziny socjologii, a w kontekście badania przestrzeni rozwiązań problemów optymalizacyjnych został zaprezentowany po raz pierwszy w pracy [14].

W pracy[11] zostało przeprowadzone statystyczne porównanie algorytmów próbkiowania *Snowball* oraz *Markov-chain*. Eksperyment polegał na spróbkowaniu 30 instancji problemu kwadratowego przypisania, a następnie użyciu zebranych danych do stworzenia modeli regresji przewidujących jakość rozwiązań, które zostaną uzyskane przez algorytm optymalizacji uruchomiony na instancji. Wybranymi algorytmami heurystycznymi były *Robust Taboo Search* Taillarda oraz *Improved ILS* Stützla. Z zebranych danych utworzono grafy LON i zbadano takie właściwości, jak średnia wartość funkcji celu, średni stopień wychodzący wierzchołków w grafie, promień grafu, liczba lokalnych optimów i liczba krawędzi. Stworzono modele regresji typu liniowego oraz lasu losowego. Obliczono wartości korelacji pomiędzy właściwościami przestrzeni rozwiązań a jakością rozwiązań zwracanych przez algorytmy optymalizacji. Badania wykazały, że dane pozyskane z próbkowania *Markov-chain* były w większym stopniu skorelowane z jakością rozwiązań algorytmów optymalizacji, a modele regresji utworzone na ich podstawie dokonywały lepszej predykcji niż te utworzone na podstawie danych ze *Snowball*. Z kolei *Snowball* okazał się bardziej przewidywalny i łatwiejszy w doborze parametrów.

Podobne badanie przeprowadzono w pracy[12], tym razem na większej liczbie instancji - 124 instancji ze zbioru QAPLIB i dodatkowych 60 instancji o rozmiarze N=11. Dla tych dodatkowych instancji, oprócz próbkowania wykonano również przegląd zupełny. Zauważono, że algorytm *Snowball* produkuje gęstszą sieć od algorytmu *Markov-chain*. Dostrzeżono wadę algorytmu *Snowball* - duży wpływ parametrów próbkowania na wartości metryk opartych o gęstość i wzory połączeń krawędzi, oraz metryk opisujących strukturę lejowe. Stworzono modele regresji przewidujące jakość rozwiązań generowanych przez heurystyki *Taboo Search* i ILS. Model przewidujący odpowiedź ILS oparty o dane z algorytmu Snowball był nieco lepszy od modelu opartego o dane z algorytmu ILS. Między predykcjami modeli przewidujących odpowiedź algorytmu TS nie było znaczającej różnicy. Wśród wszystkich czterech testowanych modeli cechą przestrzeni będącą najlepszym predyktorem okazała się średnia wartość funkcji dopasowania (ang. mean fitness). Porównanie spróbkowanych sieci LON małych instancji z sieciami uzyskanymi poprzez przegląd zupełny wykazało, że zarówno *Markov-chain* jak i *Snowball* dobrze aproksymują liczbę wierzchołków i krawędzi w sieci. *Snowball* odnalazł prawie wszystkie optima lokalne obecne w pełnej sieci. *Markov-chain* znalazł ich mniej, ale nadal dawał zadowalający wynik.

Z innych prac z dziedziny warto wymienić[9], w którym porównano algorytmy wstępujące (hillclimb) typu *best-improvement* i *first-improvement* w próbkowaniu przestrzeni NK. Wykorzystanie algorytmu typu *first-improvement* prowadziło do powstania gęstszej, bardziej kompletnej sieci optimów lokalnych. Pętle obecne w sieci miały mniejszą wagę w przypadku algorytmu *first-improvement* co sugeruje, że łatwiej wychodzi z optimum lokalnego. Dodatkową zaletą tego algorytmu jest krótszy czas wykonywania, spowodowany brakiem konieczności przeglądu za każdym razem całego sąsiedztwa rozwiązania początkowego.

Z kolei w pracy[5] zbadano wpływ siły perturbacji (stopnia, w jakim operacja perturbacji zmienia rozwiązanie początkowe) na właściwości spróbkowanej przestrzeni. Wybranym algorytmem próbującym był algorytm typu *Markov-chain* pochodzący z artykułu[7]. Operacją perturbacji tego algorytmu jest procedura double-bridge. Badania przeprowadzono na 180 instancjach o znanych rozwiązaniach optymalnych, uzyskanych przy pomocy generatora DIMACS TSP . Celem było znalezienie takiego parametru k - siły perturbacji - aby uzyskać jak największy wskaźnik sukcesu. Za wskaźnik sukcesu przyjęto stosunek liczby przebiegów algorytmu, które znalazły globalne optimum do wszystkich przebiegów. Nie znaleziono uniwersalnej najlepszej wartości parametru k. Zauważono jednak, że niższa siła perturbacji sprawdzała się dla instancji, w których miasta rozmieszczone są w klastrach. Dla instancji z miastami rozmieszczonymi równomiernie zwiększenie wartości k powodowało zmniejszenie ilości suboptimalnych lejów w przestrzeni. Zaobserwowano również, że instancje z miastami ułożonymi w klastrach były generalnie prostsze do rozwiązania, niż instancje z rozkładem równomiernym, a także, że rozkład optimów lokalnych w przestrzeni rozwiązań ma większy wpływ na trudność znalezienia optymalnego rozwiązania dla danej instancji, niż ich ilość.

2.3 O przestrzeni rozwiązań

Przestrzeń rozwiązań (przestrzeń przystosowania, ang. Fitness Landscape) jest pojęciem wywodzącym się z biologii ewolucyjnej. W kontekście biologicznym jest to model opisujący relację między genotypem i fenotypem organizmów, a ich przystosowaniem (ang. fitness), które jest miarą opisującą sukces reprodukcyjny[3].

W kontekście optymalizacji Fitness landscape to trójka (S, V, f) , gdzie:

- S jest zbiorem wszystkich możliwych rozwiązań - przestrzenią przeszukiwania,
- V jest funkcją przypisującą każdemu rozwiązaniu $s \in S$ zbiór sąsiadów $V(s)$,
- f jest funkcją $f : S \rightarrow \mathbb{R}$ przypisującą danemu rozwiązaniu wartość przystosowania - zwykle jest to wartość funkcji celu dla danego rozwiązania.

2.3.1 Sieć optimów lokalnych

Sieć optimów lokalnych(ang. Local Optima Network, LON) jest konstruktem zaprezentowanym po raz pierwszy w artykule[13], i rozwiniętym w pracy[8].

Jest to graf $G = (N, E)$ przedstawiający występujące w przestrzeni rozwiązań optima lokalne (zbiór wierzchołków N) i relacje między nimi (zbiór krawędzi E).

2.3.2 Wierzchołki

Wierzchołki w sieci optimów lokalnych reprezentują optima lokalne w przestrzeni rozwiązań. Do optimów zaliczamy minima i maksima; w problemach optymalizacyjnych zazwyczaj poszukujemy tych pierwszych. Minimum lokalne to takie rozwiązanie s , w którego sąsiedztwie $V(s)$ nie znajduje się żadne rozwiązanie x , dla którego $f(x) < f(s)$. Każde rozwiązanie w przestrzeni rozwiązań można przyporządkować do pewnego lokalnego minimum. Aby znaleźć lokalne minimum $n \in N$, do którego "prowadzi" dane rozwiązanie $s \in S$, wykonuje się lokalną optymalizację z tym rozwiązaniem przyjętym jako punkt startowy. W dalszej części pracy takie przyporządkowanie będzie oznaczane jako

$h(s) \rightarrow n \in N$. Wierzchołkom w sieci LON można przypisać wagę równą wartości funkcji celu w danym optimum lokalnym.

2.3.3 Krawędzie

Krawędzie w sieci lokalnych optimów mogą być zdefiniowane na jeden z kilku sposobów. W literaturze[8][10] zostały opisane trzy różne modele: *basin-transition*, *escape edges* i *perturbation edges*.

Basin-transition

Basen przyciągania optimum lokalnego n jest zdefiniowany jako zbiór:

$$b_i = \{s \in S \mid h(s) = n\}$$

Rozmiarem basenu jest liczność tego zbioru oznaczana jako $|b_i|$. Dla każdej pary rozwiązań w przestrzeni można obliczyć prawdopodobieństwo przejścia z jednego rozwiązania do drugiego $p(s \rightarrow s')$. Dla rozwiązań reprezentowanych permutacją o długości M , prawdopodobieństwo takie wynosi:

$$p(s \rightarrow s') = \frac{1}{M(M-1)/2}, \quad \text{jeżeli } s' \in V(s),$$

$$p(s \rightarrow s') = 0, \quad \text{jeżeli } s' \notin V(s),$$

Mając informacje o prawdopodobieństwach przejścia między poszczególnymi rozwiązaniami można obliczyć prawdopodobieństwo przejścia od rozwiązania s do dowolnego rozwiązania należącego do basenu b_j :

$$p(s \rightarrow b_j) = \sum_{s' \in b_j} p(s \rightarrow s')$$

Całkowite prawdopodobieństwo przejścia z basenu jednego optimum lokalnego do drugiego wynosi więc:

$$p(b_i \rightarrow b_j) = \frac{1}{|b_i|} \cdot \sum_{s \in b_i} p(s \rightarrow b_j)$$

To całkowite prawdopodobieństwo stanowi wagę krawędzi w grafie.

Krawędzie typu *basin-transition* tworzą gęstszą sieć od krawędzi typu *escape edges*.

Escape Edges

Escape Edges zdefiniowane są przy pomocy funkcji dystansu d zwracającej najmniejszą odległość między dwoma rozwiązaniami, oraz liczby całkowitej D . Krawędź e_{ij} między lokalnymi optimami n_i i n_j istnieje, jeśli istnieje rozwiązanie s takie, że:

$$d(s, n_i) \leq D \wedge h(s) = n_j \tag{2.1}$$

Wagą takiej krawędzi jest ilość rozwiązań spełniających powyższy warunek.

Perturbation edges

W tym modelu wagę krawędzi pomiędzy lokalnymi optimami n_i i n_j uzyskuje się poprzez kilkukrotne wykonanie operacji perturbacji(mutacji) na n_i , a następnie optymalizacji lokalnej otrzymanego rozwiązania. Liczba przypadków, w których po optymalizacji otrzymujemy rozwiązanie n_j podzielona przez liczbę prób stanowi wagę krawędzi.

$$w_{ij} = \frac{|\{opt(mut(n_i)) = n_j\}|}{trials}$$

TODO: czy zapis jest poprawny \wedge ?

2.3.4 Struktury lejowe

W sieci optimów lokalnych możemy wyróżnić sekwencje złożone z optimów lokalnych, których wartość przystosowania jest niemalejąca. Sekwencje te zwane są sekwencjami monotonicznymi[7]. Sekwencje monotoniczne zmierzające do tego samego ścieku (ang. sink, wierzchołek bez krawędzi wychodzących) tworzy strukturę zwaną lejem. Wspomniany ściek stanowi spód leja (ang. funnel bottom), a liczba wierzchołków zawartych w tej strukturze określa jej rozmiar. Ponadto w przestrzeni rozwiązań można wyróżnić lej pierwszorzędny(ang. primary funnel), kończący się w globalnym optimum i leje drugorzędne, kończące się w optimach lokalnych. Jeden wierzchołek w grafie może przynależeć jednocześnie do wielu lejów.

Leje w sieci optimów lokalnych można zidentyfikować poprzez usunięcie z grafu krawędzi prowadzących od lepszych rozwiązań do gorszych, zidentyfikowanie ścieków, odwrócenie krawędzi i wykonanie przeszukiwania wgłąb lub wszerz w celu odnalezienia wierzchołków należących do leja.

2.3.5 Metryki przestrzeni rozwiązań

- **num_sinks** - liczba ścieków. Ściek (ang. sink) jest wierzchołkiem grafu nie posiadającym krawędzi wychodzących. Pętle nie są uwzględniane.
- **num_sources** - liczba źródeł. Źródło (ang. source) jest wierzchołkiem grafu nie posiadającym krawędzi wchodzących. Pętle nie są uwzględniane.
- **num_sub sinks** - Liczba wierzchołków, które nie posiadają krawędzi wychodzących do rozwiązań o niższej wartości funkcji celu.
- **edge_to_node** - stosunek liczby krawędzi do liczby wierzchołków w grafie.
- **avg_fitness** - średnia wartość funkcji celu lokalnych optimów w sieci.
- **distLO** - średnia odległość rozwiązań od rozwiązania z najniższą wartością funkcji celu. Odległość jest zdefiniowana jako odwrotność wagi krawędzi łączących rozwiązania. Rozwiązania nie połączone krawędzią z najlepszym rozwiązaniem nie sąbrane pod uwagę.
- **conrel** - Stosunek liczby rozwiązań połączonych krawędzią z najlepszym rozwiązaniem do liczby pozostałych rozwiązań.
- **avg_out_degree, max_out_degree** - średni i maksymalny stopień wychodzący rozwiązań w grafie. Stopień wychodzący wierzchołka to liczba wychodzących z niego krawędzi. Pętle nie sąbrane pod uwagę.

- **avg_in_degree, max_in_degree** - średni i maksymalny stopień wchodzący rozwiązań w grafie. Stopień wchodzący wierzchołka to liczba wchodzących do niego krawędzi. Pętle nie są brane pod uwagę.
- **assortativity** - współczynnik różnorodności grafu. Różnorodność (ang. assortativity) grafu skierowanego jest zdefiniowana następującym wzorem:

$$\text{assortativity} = \frac{1}{\sigma_o \sigma_i} \sum_{(j,k) \in E} \deg(j) \cdot \deg(k) \cdot (e_{jk} - q_j^o q_k^i)$$

Gdzie:

- e_{ij} - część krawędzi łączących wierzchołki i i j w stosunku do liczby wszystkich krawędzi (ułamek z zakresu 0 do 1),
- $q_i^o = \sum_{j \in V} e_{ij}$
- $q_i^i = \sum_{j \in V} e_{ji}$
- σ_o - odchylenie standardowe q^o
- σ_i - odchylenie standardowe q^i
- **clustering_coeff** - współczynnik klasteryzacji grafu (ang. clustering coefficient, transitivity) opisuje prawdopodobieństwo istnienia połączenia pomiędzy sąsiednimi wierzchołkami. Współczynnik klasteryzacji opisany jest wzorem:

$$cc = \frac{N_{triangles}}{N_{triples}}$$

Gdzie $N_{triangles}$ to liczba trójkątów w grafie, a $N_{triples}$ to liczba połączonych trójkę. Trójkąt to trójka wierzchołków (x, y, z) taka, że $(x, y), (y, z), (x, z) \in E$. Połączona trójka to trójka wierzchołków (x, y, z) taka, że $(x, y), (y, z) \in E$.

- **density** - gęstość grafu. Jest to stosunek liczby krawędzi w grafie do maksymalnej liczby krawędzi, jaka mogłaby istnieć w tym grafie. Dana jest wzorem:

$$\text{density} = \frac{|E|}{|V|(|V| - 1)}$$

- **cliques_num** - Liczba klik. Klika w grafie jest podzbiorem zbioru wierzchołków, w którym wszystkie wierzchołki są sąsiednie - istnieje krawędź pomiędzy każdą parą wierzchołków należącą do zbioru.
- **maximal_cliques_num** - Liczba klik maksymalnych w grafie. Klika maksymalna to klika, której nie można powiększyć poprzez dołączenie do niej sąsiedniego wierzchołka.
- **largest_clique_size** - rozmiar największej kliki.
- **reciprocity** - Wzajemność. Wzajemność jest miarą zdefiniowaną tylko dla grafów skierowanych. Jest to stosunek wierzchołków wzajemnie połączonych do wierzchołków, które są połączone krawędzią tylko w jednym kierunku. Dana jest wzorem:

$$\text{reciprocity} = \frac{|(i, j) \in E \mid (j, i) \in E|}{|(i, j) \in E \mid (j, i) \notin E|}$$

- **funnel_num** - liczba lejów w przestrzeni rozwiązań
- **mean_funnel_size** - średnia wielkość leja
- **max_funnel_size** - wielkość największego leja
- **go_path_ratio** - stosunek liczby wierzchołków ze ścieżką do najlepszego rozwiązania do liczby wszystkich wierzchołków.
- **avg_go_path_len** - średnia długość ścieżki do najlepszego rozwiązania. Wierzchołki bez ścieżki do najlepszego rozwiązania nie sąbrane pod uwagę Długość ścieżki definiowana jest jako liczba krawędzi wchodzących w skład ścieżki pomiędzy wierzchołkami.
- **max_go_path_len** - długość najdłuższej ścieżki do najlepszego rozwiązania.
- **num_cc** - Liczba spójnych podgrafów grafu
- **largest_cc** - Wielkość (liczba wierzchołków) największego spójnego podgrafa.
- **largest_cc_radius** - Promień największego spójnego podgrafa. Promień grafu to najmniejsza acentryczność wierzchołka wśród wszystkich wierzchołków grafu. Acentryczność (ang. eccentricity) wierzchołka to największa z odległości wierzchołka do innych wierzchołków grafu.

2.3.6 Problem komiwojażera

Problem komiwojażera (ang. Traveling Salesman Problem, TSP) jest znanym problemem optymalizacyjnym sformułowanym w następujący sposób: mając do dyspozycji listę miast i odległości między nimi, należy odnaleźć najkrótszą ścieżkę przechodzącą przez wszystkie miasta zaczynającą i kończącą się w ustalonym punkcie. Problem ten jest problemem NP-trudnym i z tego powodu do rozwiązywania większych jego instancji konieczne jest stosowanie algorytmów heurystycznych.

2.3.7 Operacja 2-exchange

Operacja 2-exchange jest operacją wybrania dwóch wierzchołków i zamiany krawędzi prowadzących do następnych wierzchołków. Procedura jest wykorzystywana w algorytmie heurystycznym 2opt, w którym w ten sposób "rozplatane" są skrzyżowane krawędzie.

Algorytmy przeszukiwania przestrzeni rozwiązań zaprezentowane w tej pracy wykorzystują operację 2-exchange jako operację mutacji. Mutacja permutacji polega na D-krotnym wykonaniu operacji 2-exchange na losowych wierzchołkach, gdzie D to maksymalna odległość z równania 2.1.

Operacja 2-exchange oraz operator mutacji zostały przedstawione na listingu 1.

Algorithm 1: Operacja 2exchange - pseudokod

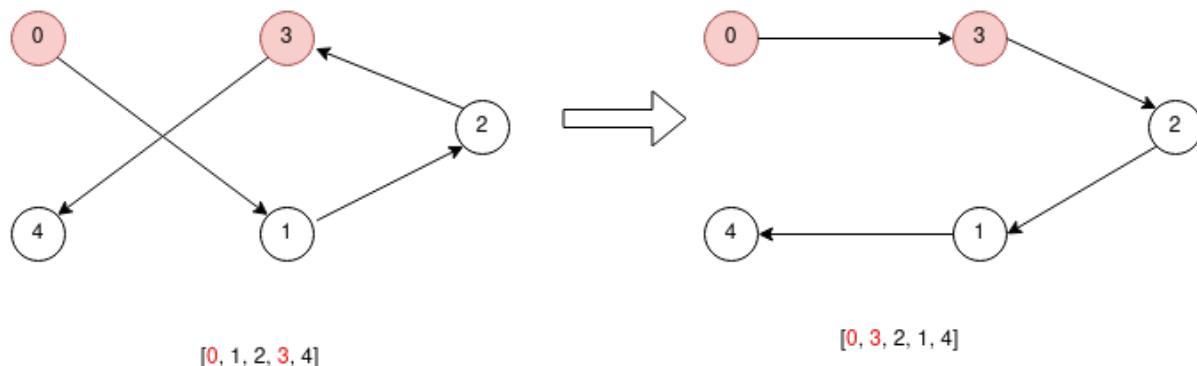
```

function 2exchange( $a, b, perm$ ):
     $a \leftarrow a + 1;$ 
    while  $a < b$  do
        zamien( $perm[a], perm[b]$ );
         $a \leftarrow a + 1;$ 
         $b \leftarrow b + 1;$ 
    end
    return  $perm$ ;
end

function 2exchangeMutacja( $perm, D$ ):
    for  $i \leftarrow 1$  to  $D$  do
         $a \leftarrow losowaZZakresu(0, length(perm) - 3);$ 
         $b \leftarrow losowaZZakresu(a + 2, length(perm) - 1);$ 
         $perm \leftarrow 2exchange(a, b, perm);$ 
    end
    return  $perm$ ;
end

function 2exchangeWszystkiePermutacje( $perm$ ):
     $perms = \{\}$ ;
    for  $a \leftarrow 0$  to  $n - 3$  do
        for  $b \leftarrow a + 2$  to  $n - 1$  do
             $perm \leftarrow 2exchange(a, b);$ 
             $perms \leftarrow perms \cup \{perm\};$ 
        end
    end
    return  $perms$ ;
end

```



Rysunek 2.1 Przykład procedury 2-exchange

Rozdział 3

Badania eksperymentalne

3.1 Zaimplementowane algorytmy

3.1.1 Próbkowanie dwufazowe

Próbkowanie dwufazowe swoją nazwę zawdzięcza procesowi próbkowania składającemu się z dwóch oddzielnego faz - próbkowania wierzchołków oraz próbkowania krawędzi - wykonywanych jedna po drugiej. Istotną zaletą tego podejścia jest jego stosunkowo prosta implementacja.

Zaimplementowany algorytm pochodzi z pracy[1]. Został on przygotowany specjalnie do próbkowania przestrzeni rozwiązań problemu komiwojażera. Próbkowanie wierzchołków odbywa się poprzez generowanie losowych rozwiązań, a następnie ich optymalizacji algorytmem 2-opt. Próbkowanie krawędzi polega na wielokrotnym poddaniu każdego ze znalezionych wcześniej lokalnych optimów n_i operacji perturbacji typu 2-exchange, a następnie poddaniu powstałego rozwiązania optymalizacji algorytmem 2-opt typu *first-improvement* uzyskując w ten sposób lokalne optimum n_j . Następnie dodawana jest krawędź między n_i a n_j , lub - jeśli już taka istnieje - jej waga jest zwiększana o 1.

Algorytm przyjmuje trzy parametry: pożądaną liczbę wierzchołków do wygenerowania (n_{max}), maksymalną liczbę prób generowania wierzchołka (n_{att}) oraz maksymalną liczbę prób generowania krawędzi (e_{att}). Implementacja zastosowana w tej pracy dodatkowo powtarza cały proces kilkukrotnie, za każdym razem zapisując zebrane próbki do pliku.

Algorytm w postaci pseudokodu został przedstawiony na listingu 2.

3.1.2 Snowball

Próbkowanie typu Snowball wywodzi się z techniki używanej w badaniach z dziedziny socjologii, w której ludzie należący do próby z populacji rekrutują kolejnych uczestników badania spośród swoich znajomych. W kontekście badania przestrzeni rozwiązań technika ta została zaprezentowana w pracy[14], gdzie została wykorzystana do próbkowania przestrzeni problemu kwadratowego przypisania (QAP).

Próbkowanie składa się z etapów procedury *snowball* próbującej "wgłębić" i losowego spaceru (ang. *random walk*). Próbkowanie *snowball* polega na wybraniu rozwiązania startowego i przeszukaniu jego najbliższego sąsiedztwa. Następnie operacja ta jest powtarzana dla każdego rozwiązania w tym sąsiedztwie. Proces powtarza się aż do osiągnięcia z góry ustalonej głębokości przeszukiwania. Następnie rozpoczyna się procedura losowego spaceru - wybierane jest kolejne rozwiązanie startowe ze zbioru sąsiadów poprzedniego rozwiązywania startowego (lub rozwiązanie losowe, jeśli to sąsiedztwo jest puste) i proces

snowball rozpoczyna się od nowa. Procedura jest powtarzana aż osiągnięty zostanie z góry ustalony limit długości spaceru.

Zaimplementowany algorytm jest próbą adaptacji tej techniki do zadania przeszukiwania przestrzeni problemu komiwojażera. Do najważniejszych modyfikacji należy zastąpienie funkcji optymalizacji lokalnej *hillclimb* optymalizacją *2opt*, implementacja odpowiedniej funkcji celu oraz operacji mutacji typu *2-exchange*.

Algorytm w postaci pseudokodu został przedstawiony na listingu 3.

Algorithm 2: Próbkowanie dwufazowe - pseudokod

Data:

n_{max} - żądana liczba wierzchołków
 n_{att} - liczba prób generowania wierzchołków
 e_{att} - liczba prób generowania krawędzi
 n_{runs} - liczba powtórzeń
 D - stała D krawędzi

```

 $N \leftarrow \{\};$ 
 $E \leftarrow \{\};$ 
for  $i \leftarrow 1$  to  $n_{runs}$  do
   $| probkujWierzcholki(N, n_{max}, n_{att});$ 
   $| probkujKrawedzie(N, E, e_{att});$ 
   $| zapiszDoPliku(N, E);$ 
end

function probkujWierzcholki( $N, n_{max}, n_{att}$ ):
  for  $i \leftarrow 1$  to  $n_{max}$  do
    for  $i \leftarrow 1$  to  $n_{att}$  do
       $| s \leftarrow losoweRozwiazanie();$ 
       $| s \leftarrow 2opt(s);$ 
       $| N \leftarrow N \cup \{s\};$ 
    end
  end
end

function probkujKrawedzie( $N, E, e_{att}$ ):
  foreach  $n \in N$  do
    for  $i \leftarrow 1$  to  $e_{att}$  do
       $| s \leftarrow 2exchangeMutacja(n, D);$ 
       $| s \leftarrow 2optFirstImprovement(s);$ 
      if  $s \in N$  then
         $| | E \leftarrow E \cup \{(n, s)\};$ 
         $| | w_{ns} \leftarrow w_{ns} + 1;$ 
      end
    end
  end
end

```

Algorithm 3: Próbkowanie snowball - pseudokod**Data:**

w_{len} - długość losowego spaceru
 m - liczba prób przeszukania sąsiedztwa
 $depth$ - głębokość przeszukiwania
 D - stała D krawędzi
 s_{tresh} - interwał zapisu

```

 $s_1 \leftarrow losoweRozwiazanie();$ 
 $n_1 \leftarrow 2opt(s_1);$ 
 $N \leftarrow \{n_1\};$ 
 $E \leftarrow \{\};$ 
for  $j \leftarrow 1$  to  $n_{runs}$  do
    for  $i \leftarrow 1$  to  $w_{len}$  do
         $snowball(n_i, m, depth);$ 
         $n_{i+1} \leftarrow losowySpacer(n_i);$ 
    end
end
 $zapiszDoPliku(N, E);$ 

function  $snowball(n, m, depth):$ 
    if  $d > 0$  then
        for  $i \leftarrow 1$  to  $m$  do
             $s \leftarrow 2opt(2exchangeMutacja(n, D));$ 
             $N \leftarrow N \cup \{s\};$ 
            if  $|N| \ mod \ s_{tresh} = 0$  then
                 $zapiszDoPliku(N, E);$ 
            end
            if  $(n, s) \in E$  then
                 $w_{ns} \leftarrow w_{ns} + 1;$ 
            else
                 $E \leftarrow E \cup \{(n, s)\};$ 
                 $w_{ns} \leftarrow 1;$ 
                 $snowball(s, m, d - 1);$ 
            end
        end
    end
end

function  $losowySpacer(n_i):$ 
     $neighbours \leftarrow \{s : (n_i, s) \in E \wedge s \notin \{n_0 \dots n_i\}\};$ 
    if  $neighbours \neq \emptyset$  then
         $n_{i+1} \leftarrow losowyElementZeZbioru(neighbours);$ 
    else
         $s \leftarrow losoweRozwiazanie();$ 
         $n_{i+1} \leftarrow 2opt(s);$ 
         $N \leftarrow N \cup \{n_{i+1}\};$ 
        if  $|N| \ mod \ s_{tresh} = 0$  then
             $zapiszDoPliku(N, E);$ 
        end
    end
    return  $n_{i+1}$ 
end

```

3.1.3 Przegląd zupełny

Ze względu na złożoność problemu komiwojażera przegląd zupełny można zastosować tylko do bardzo małych instancji problemu. Przegląd polega na wygenerowaniu wszystkich możliwych rozwiązań danej instancji, wykonaniu na nich optymalizacji 2-opt w celu znalezienia optimów lokalnych a następnie znalezieniu krawędzi oraz obliczeniu ich wag. Dla każdego z rozwiązań generowane są wszystkie permutacje, które mogą powstać poprzez D-krotne wykonanie na rozwiązaniu operacji 2-exchange. Jeśli wśród tych permutacji znajduje się jedno ze znalezionych wcześniej lokalnych optimów, oznacza to, że spełniony jest warunek 2.1 i dodawana jest nowa krawędź lub zwiększa zostaje waga istniejącej.

Algorithm 4: Przegląd zupełny

```

 $S \leftarrow \{\};$ 
 $P \leftarrow \text{wygenerujWszystkiePermutacje}();$ 
foreach  $p \in P$  do
     $lo \leftarrow 2\text{opt}(p);$ 
     $S \leftarrow S \cup \{(p, lo)\};$ 
end
foreach  $(p, lo) \in S$  do
    foreach  $n \in N$  do
        if  $wZasiegu2Exchange(p, n, D)$  then
            if  $(n, lo) \in E$  then
                 $w_{n,lo} \leftarrow w_{n,lo} + 1;$ 
            else
                 $E \leftarrow E \cup \{(n, lo)\};$ 
                 $w_{n,lo} = 1;$ 
            end
        end
    end
end
function  $wZasiegu2Exchange(p, n, D):$ 
     $permutacje \leftarrow \{p\};$ 
    for  $i \in 1..D$  do
         $nowe\_perm \leftarrow \{\};$ 
        foreach  $perm \in permutacje$  do
             $pochodne\_perm \leftarrow 2exchangeWszystkiePermutacje(perm);$ 
            foreach  $poch \in pochodne\_perm$  do
                if  $poch = n$  then
                    return  $true;$ 
                end
                 $nowe\_perm \leftarrow nowe\_perm \cup \{poch\};$ 
            end
        end
         $permutacje \leftarrow nowe\_perm;$ 
    end
    return  $false;$ 
end

```

3.2 Instancje testowe

Do badań wykorzystano instancje testowe wygenerowane losowo oraz wybrane instancje ze zbioru TSPLIB. Zaimplementowano trzy generatory tworzące różne typy instancji testowych: z miastami rozłożonymi równomiernie, z miastami rozłożonymi w klastrach oraz z miastami ułożonymi na siatce. Wygenerowano instancje testowe każdego z trzech typów instancji losowych o rozmiarach 7, 8, 9, 10, 11 oraz 20, 50, 80 i 100. Uzyskano w ten sposób 27 instancji problemu. Ze zbioru TSPLIB wybrano instancje o podobnych rozmiarach: **burma14**, **ulysses22**, **att48**, **berlin52**, **pr76**, **eil76**, **rat99**, **bier127**. W sumie badanie przeprowadzono na 35 instancjach problemu.

Miasta rozmieszczone równomiernie

Generator losowo rozmieszcza miasta na wirtualnej planszy o ustalonym rozmiarze. Współrzędne miast generowane są losowo z rozkładu równomiernego. W dalszej części dokumentu instancje wygenerowane tym generatorem będą nazywane **uniform_<liczba miast>**.

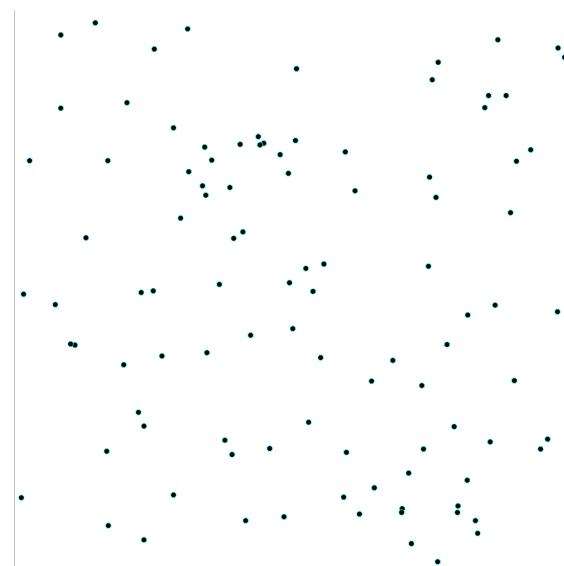
Przykład wygenerowanej instancji został przedstawiony na rysunku 3.1.

Miasta rozmieszczone w klastrach

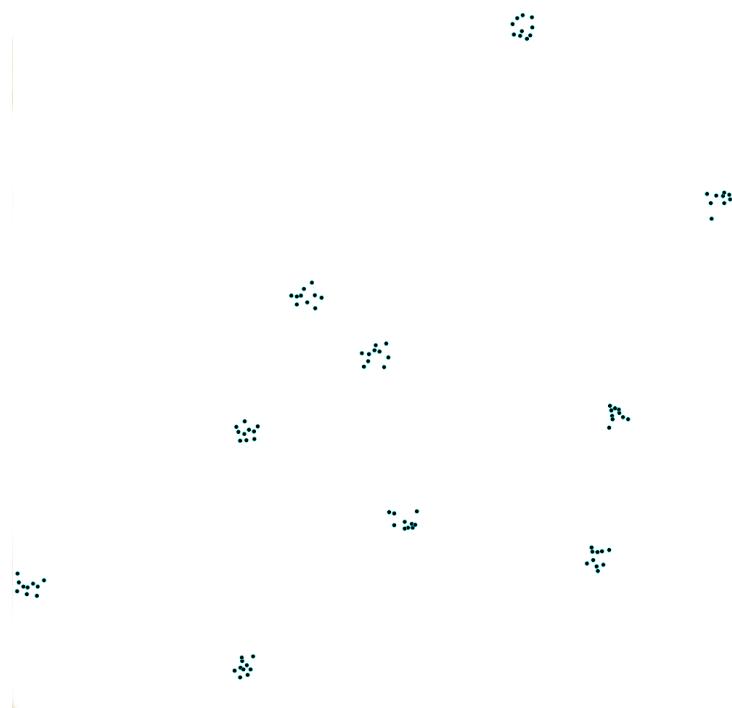
Miasta umieszczane są blisko siebie w kilku grupach oddzielonych większymi odległościami. W dalszej części dokumentu instancje wygenerowane tym generatorem będą nazywane **cliques_<liczba miast>**. Przykład wygenerowanej instancji został przedstawiony na rysunku 3.2.

Miasta rozmieszczone na siatce

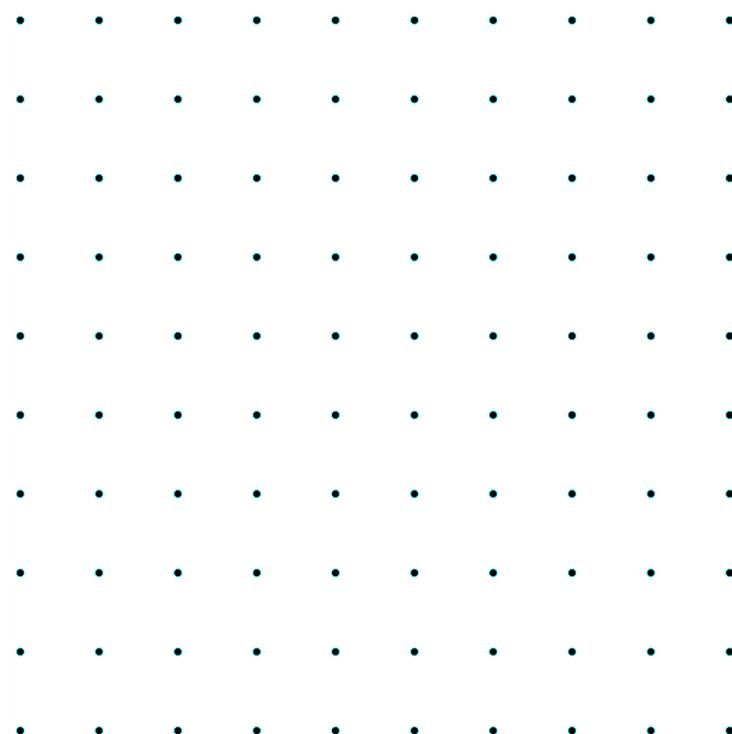
Miasta umieszczane są na siatce, w stałej odległości od swoich sąsiadów. W dalszej części dokumentu instancje wygenerowane tym generatorem będą nazywane **grid_<liczba miast>**. Przykład wygenerowanej instancji został przedstawiony na rysunku 3.3.



Rysunek 3.1 Wizualizacja przykładowej wygenerowanej instancji z miastami rozmieszczenymi równomiernie dla 100 miast



Rysunek 3.2 Wizualizacja przykładowej wygenerowanej instancji z miastami rozmieszczonymi w klastrach dla 100 miast



Rysunek 3.3 Wizualizacja przykładowej wygenerowanej instancji z miastami rozmieszczonymi na siatce dla 100 miast

3.3 Opis eksperymentu

Dla każdej z instancji problemu wykonano próbkowanie przestrzeni przy użyciu algorytmów dwufazowego oraz *snowball*. Dla małych instancji (do 11 miast) wykonano również przegląd zupełny, w celu porównania wartości miar uzyskanych z próbkowania z wartościami prawdziwymi.

Próbkowanie algorytmem dwufazowym przeprowadzono z następującymi parametrami:

- n_{max} - żądana liczba wierzchołków - 1000
- n_{att} - liczba prób generowania wierzchołków - 1000
- e_{att} - liczba prób generowania krawędzi - 1000
- n_{runs} - liczba powtórzeń - 100

Próbkowanie przeprowadzono przez ustaloną z góry liczbę powtórzeń, zapisując stan przestrzeni po zakończeniu każdego powtórzenia. Dla małych instancji próbkowanie przeprowadzono: //TODO: parametry dla próbkowania małych instancji, jak już to zrobisz.

Próbkowanie algorytmem *snowball* wykonano z następującymi parametrami:

- w_{len} - długość losowego spaceru - 10000
- m - liczba prób przeszukania sąsiedztwa - 100
- $depth$ - głębokość przeszukiwania - 3

Próbkowanie algorytmem *snowball* prowadzono do zakończenia losowego spaceru lub osiągnięcia liczby 100000 wierzchołków. Dla bardzo małych instancji (o rozmiarze 6 do 11) zapisywano stan przestrzeni za każdym razem, gdy dodany został nowy wierzchołek. Dla instancji: **burma14**, **ulysses22**, **grid_20**, **uniform_20**, **cliques_20**, oraz **cliques_50**, zapis wykonywano co 100 nowych wierzchołków. Dla pozostałych instancji stan próbkowania zapisywany był co 1000 znalezionych wierzchołków.

Po zakończeniu próbkowania dla każdego zapisanego stanu przestrzeni obliczono wartości badanych miar. W ten sposób uzyskano wartości miar dla różnych etapów próbkowania przestrzeni.

Obliczono: //TODO: Średnia, odchylenie standardowe? Utworzono wykresy przedstawiające zależność wartości miar od liczby spróbkowanych wierzchołków.

3.4 Wyniki

3.4.1 Porównanie wartości metryk dla małych instancji

Dla wygenerowanych instancji o rozmiarze 6-11 wartości miar uzyskane z próbkowania porównano z wartościami uzyskanymi z przeglądu zupełnego. Dla wartości każdej miary obliczono wzajemny błąd z wzoru 3.1.

$$\frac{|s - t|}{t} \cdot 100\% \quad (3.1)$$

Gdzie s to wartość z przestrzeni spróbkowanej, a t to wartość z przestrzeni z przeglądu zupełnego. Porównano wartości wszystkich metryk wymienionych w sekcji 2.3.5, oprócz

Tabela 3.1 Błąd względny podany w % dla instancji o równomiernie ułożonych miastach

miara \ instancja	uniform_7	uniform_8	uniform_9	uniform_10	uniform_11
distLO	99.88	99.92	99.86	98.28	98.05
avg_loop_weight	218748.47	184274.79	105710.00	24220.36	21559.15

Tabela 3.2 Błąd względny podany w % dla instancji o miastach ułożonych w klastrach

miara \ instancja	cliques_7	cliques_8	cliques_9	cliques_10	cliques_11
distLO	99.97	99.61	99.83	99.29	98.71
avg_loop_weight	362238.26	109015.44	102432.09	32200.04	13969.65

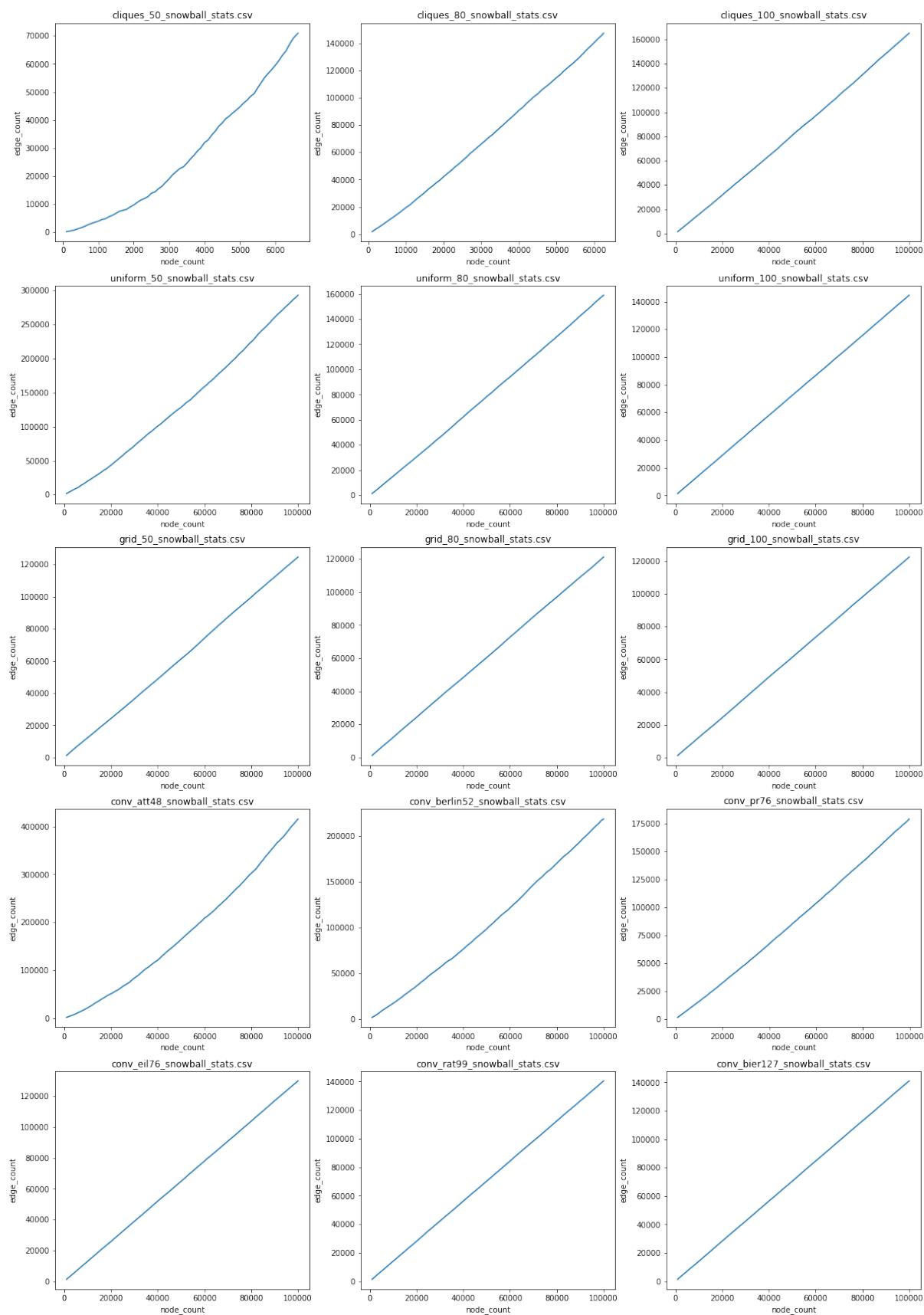
Tabela 3.3 Błąd względny podany w % dla instancji o miastach ułożonych na siatce

miara \ instancja	grid_7	grid_8	grid_9	grid_10	grid_11
edge_count	0.00	0.00	0.00	0.00	1.91
edge_to_node	0.00	0.00	0.00	0.00	1.91
distLO	99.96	99.92	97.90	85.06	79.36
avg_loop_weight	369966.52	182173.06	12715.99	6865.94	2117.10
avg_go_path_len	0.00	0.00	0.00	0.00	2.00
mean_funnel_size	0.00	0.00	0.00	0.00	1.01
avg_out_degree	0.00	0.00	0.00	0.00	1.98
avg_in_degree	0.00	0.00	0.00	0.00	1.98
density	0.00	0.00	0.00	0.00	1.91
avg_path_len	0.00	0.00	0.00	0.00	0.92
cliques_num	0.00	0.00	0.00	0.00	52.14
maximal_cliques_num	0.00	0.00	0.00	0.00	18.01
largest_clique_size	0.00	0.00	0.00	0.00	10.53
reciprocity	0.00	0.00	0.00	0.00	0.16

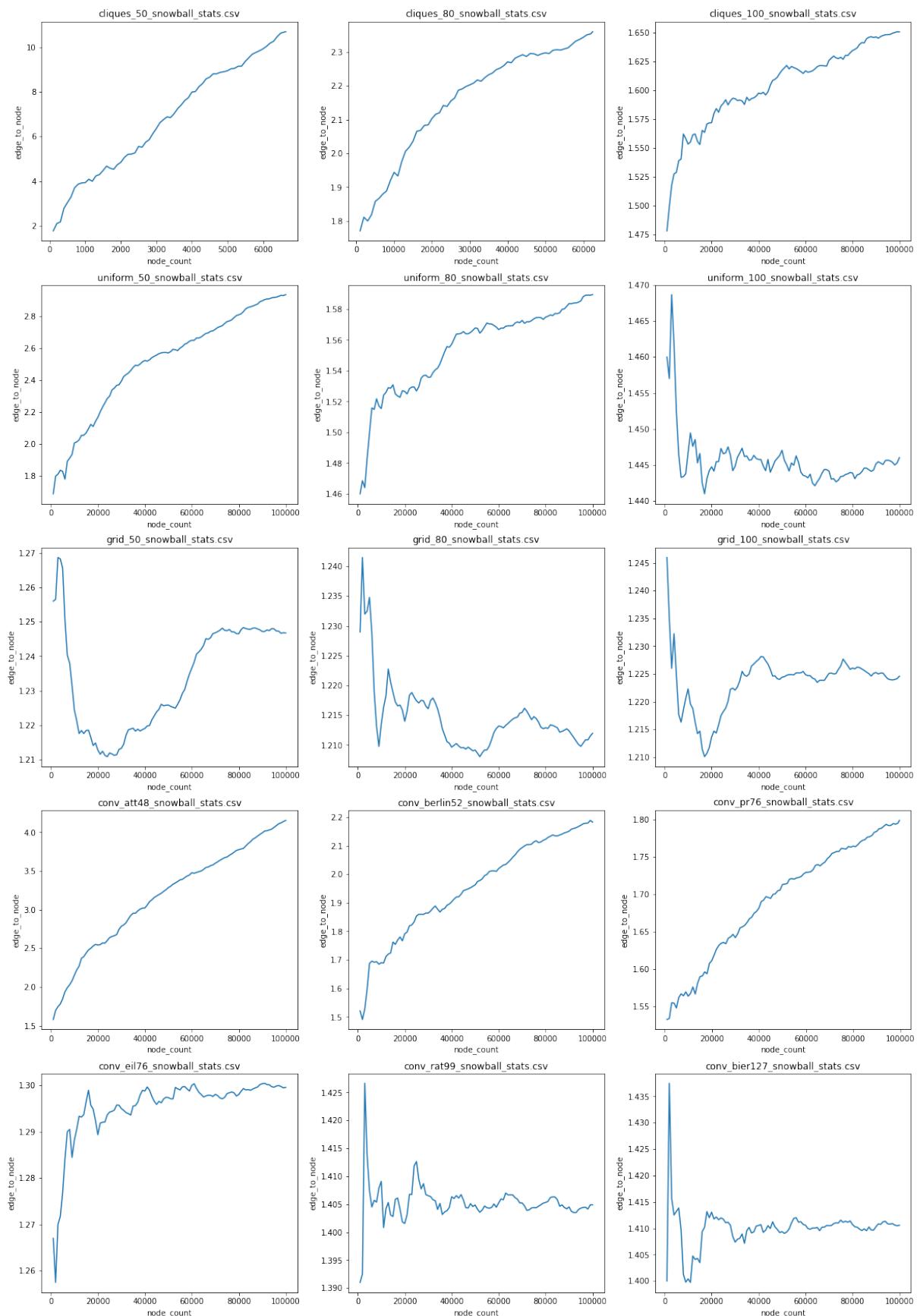
różnorodności (assortativity). Otrzymane wartości błędów przedstawiono w tabelach: 3.1, 3.2, 3.3 W tabelach pominięto metryki, dla których wartość błędu była równa 0.

TODO: Komentarz

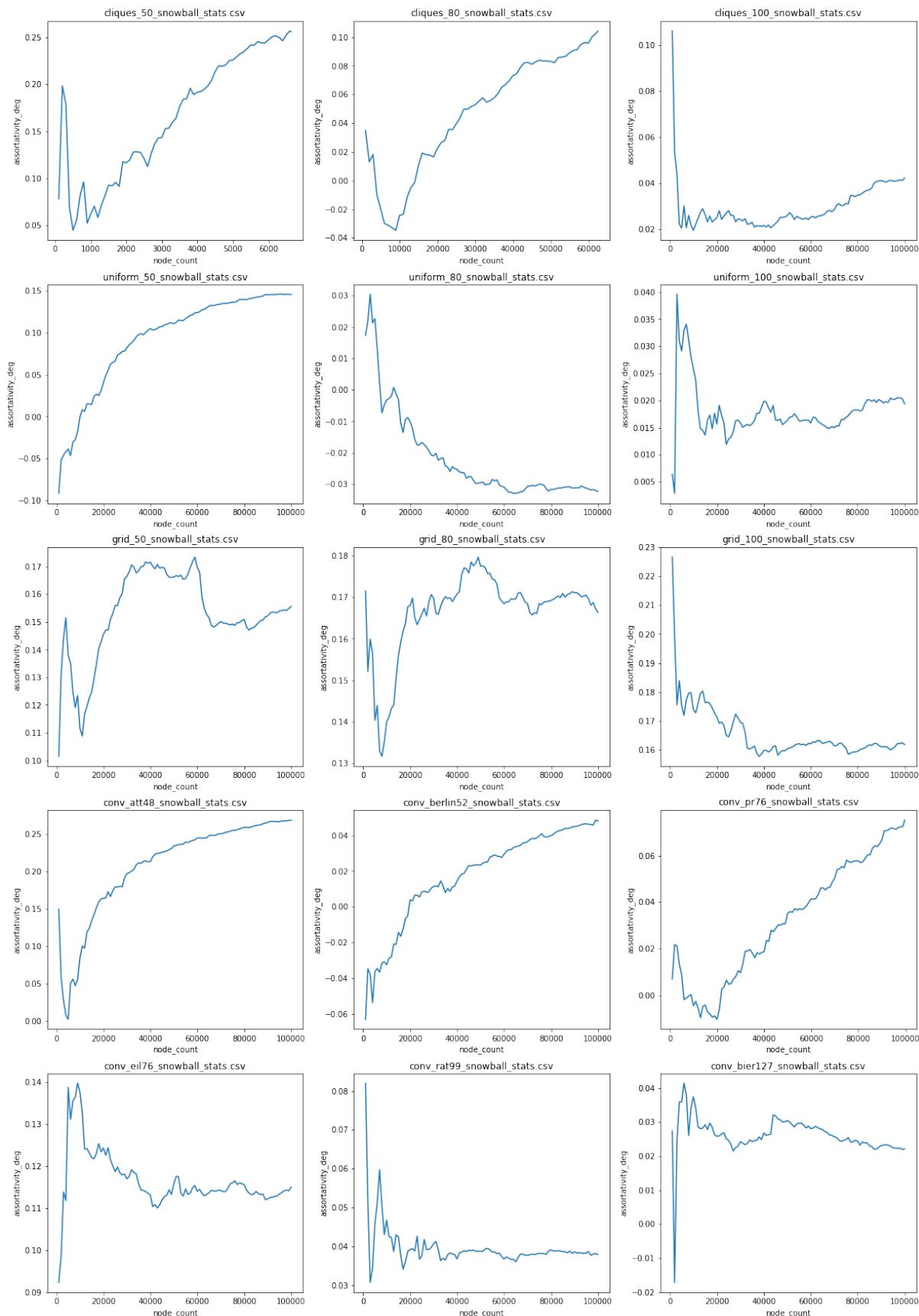
3.4.2 Badanie stabilności



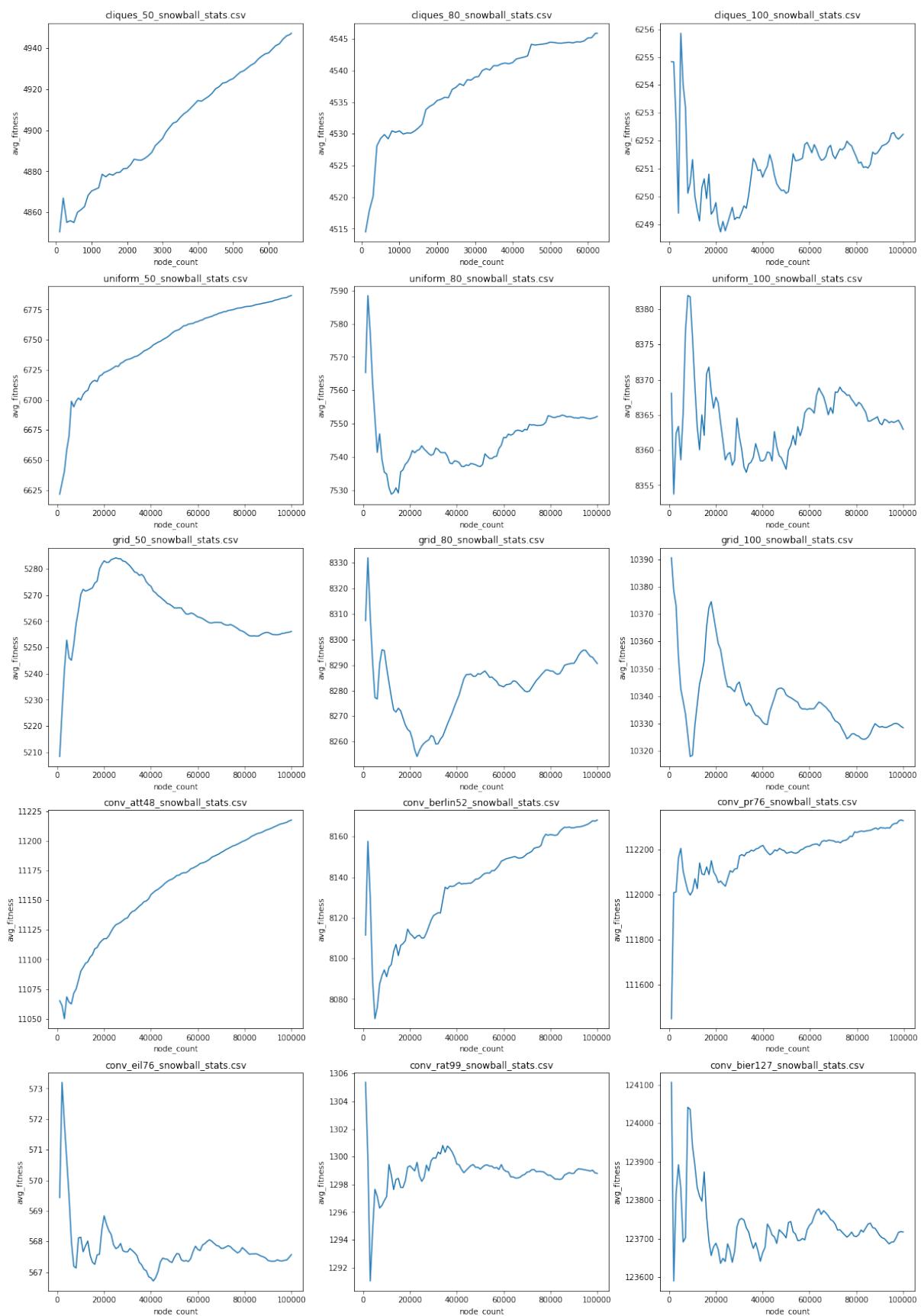
Rysunek 3.4 Liczba krawędzi zależności od liczby wierzchołków - próbkowanie snowball



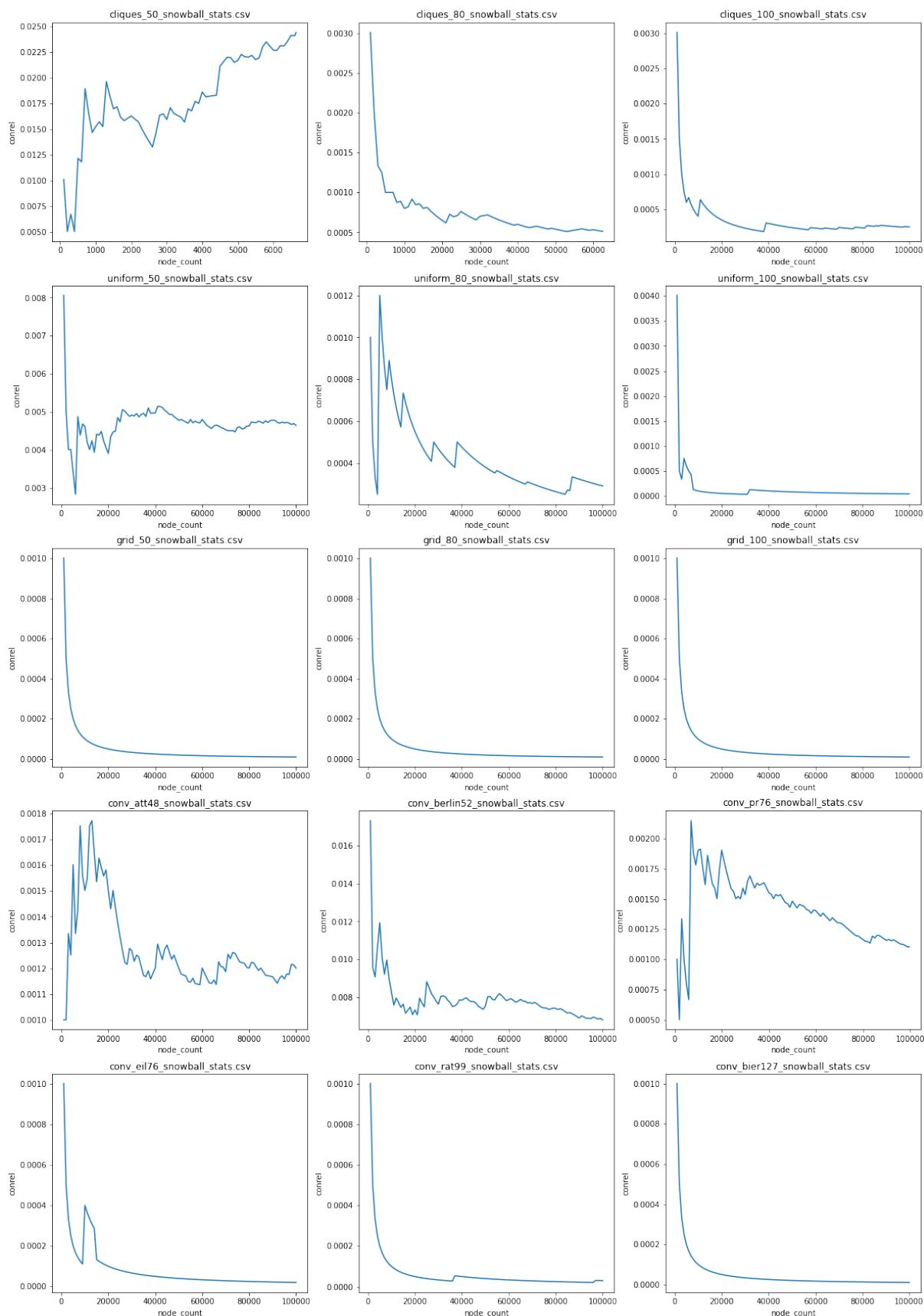
Rysunek 3.5 Stosunek liczby krawędzi do liczby wierzchołków zależności od liczby wierzchołków - próbkowanie snowball



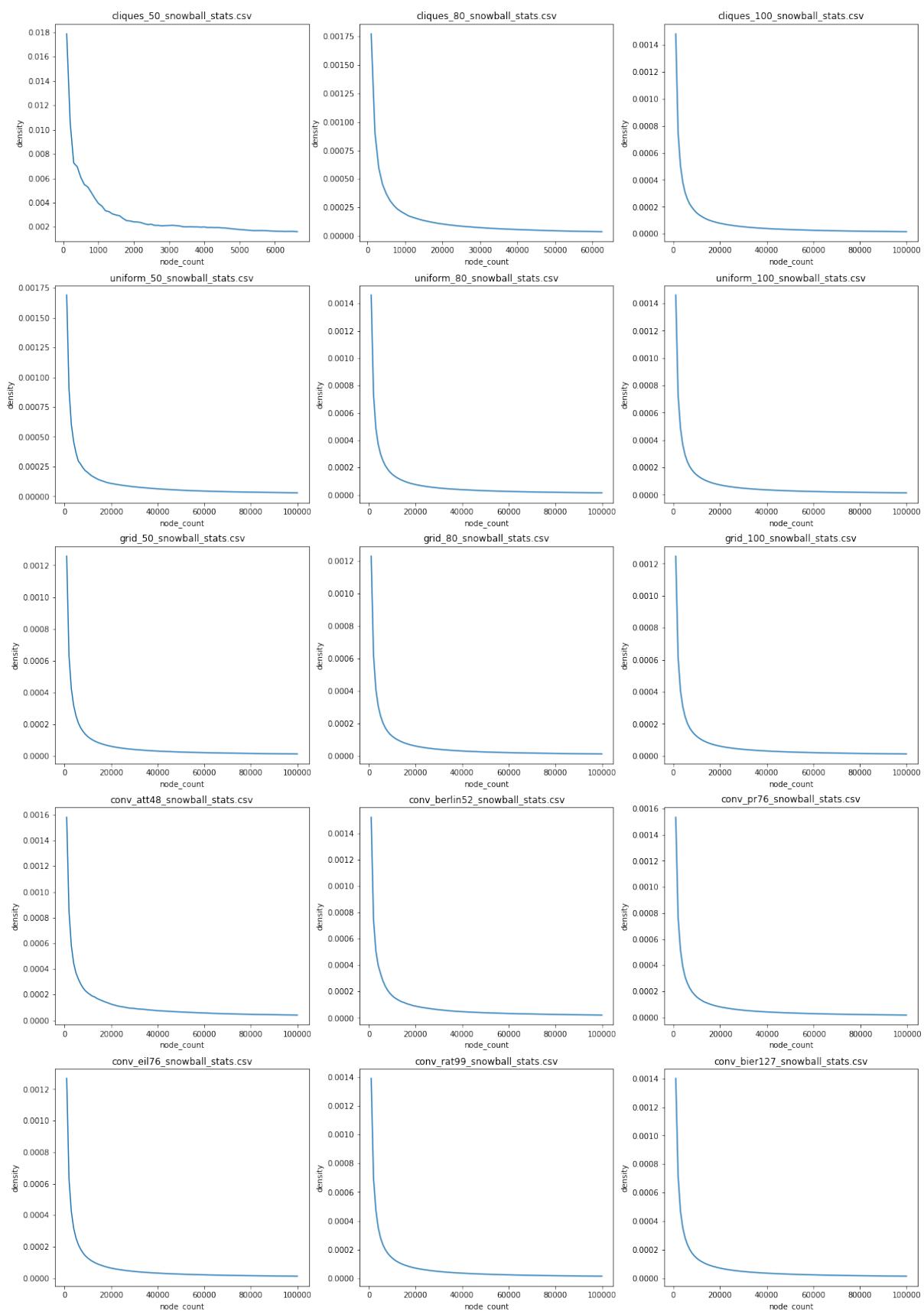
Rysunek 3.6 Współczynnik różnorodności grafów zależności od liczby wierzchołków - próbkowanie snowball



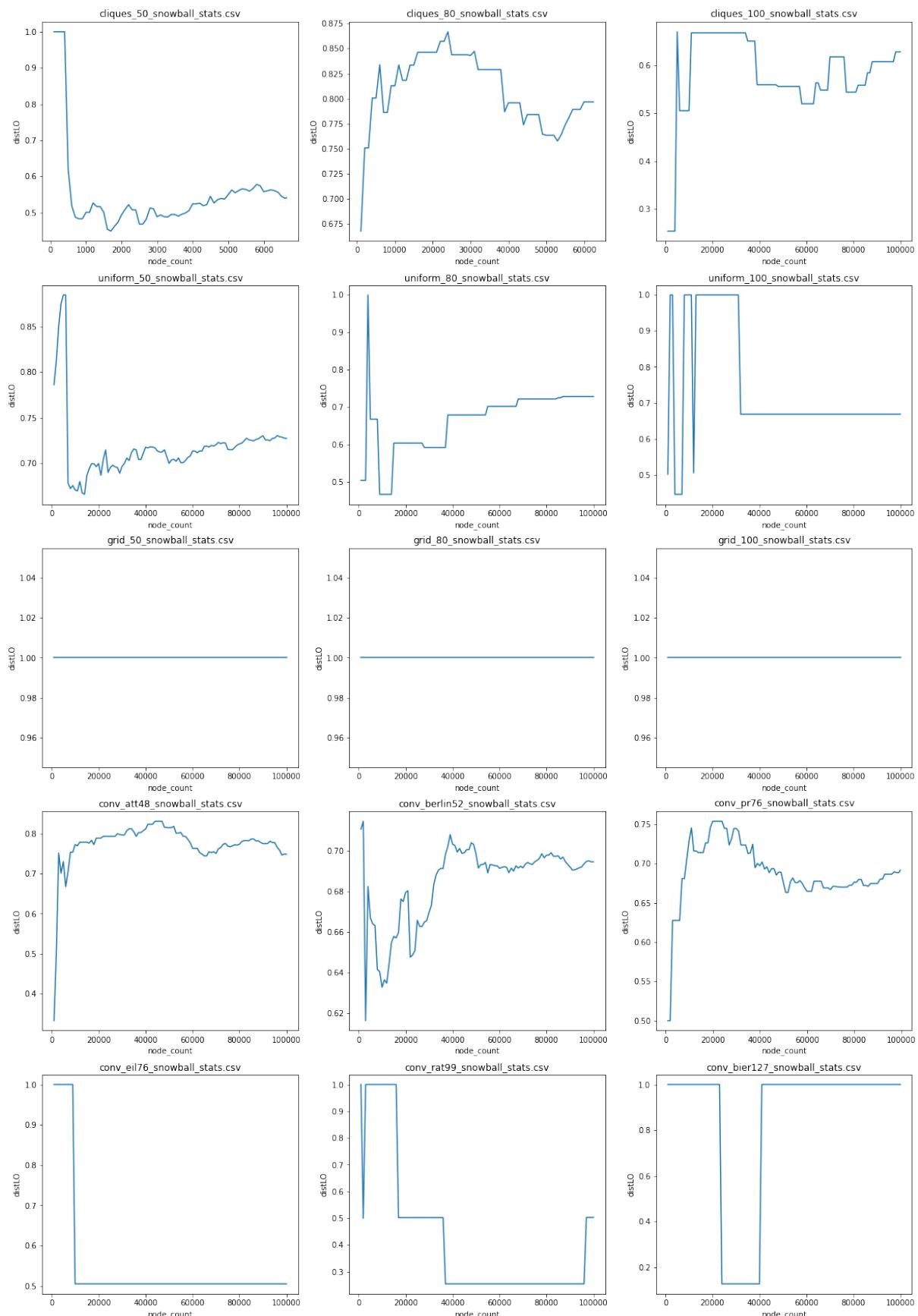
Rysunek 3.7 Średnia wartość funkcji celu w znalezionych optimach lokalnychw zależności od liczby wierzchołków - próbkowanie snowball



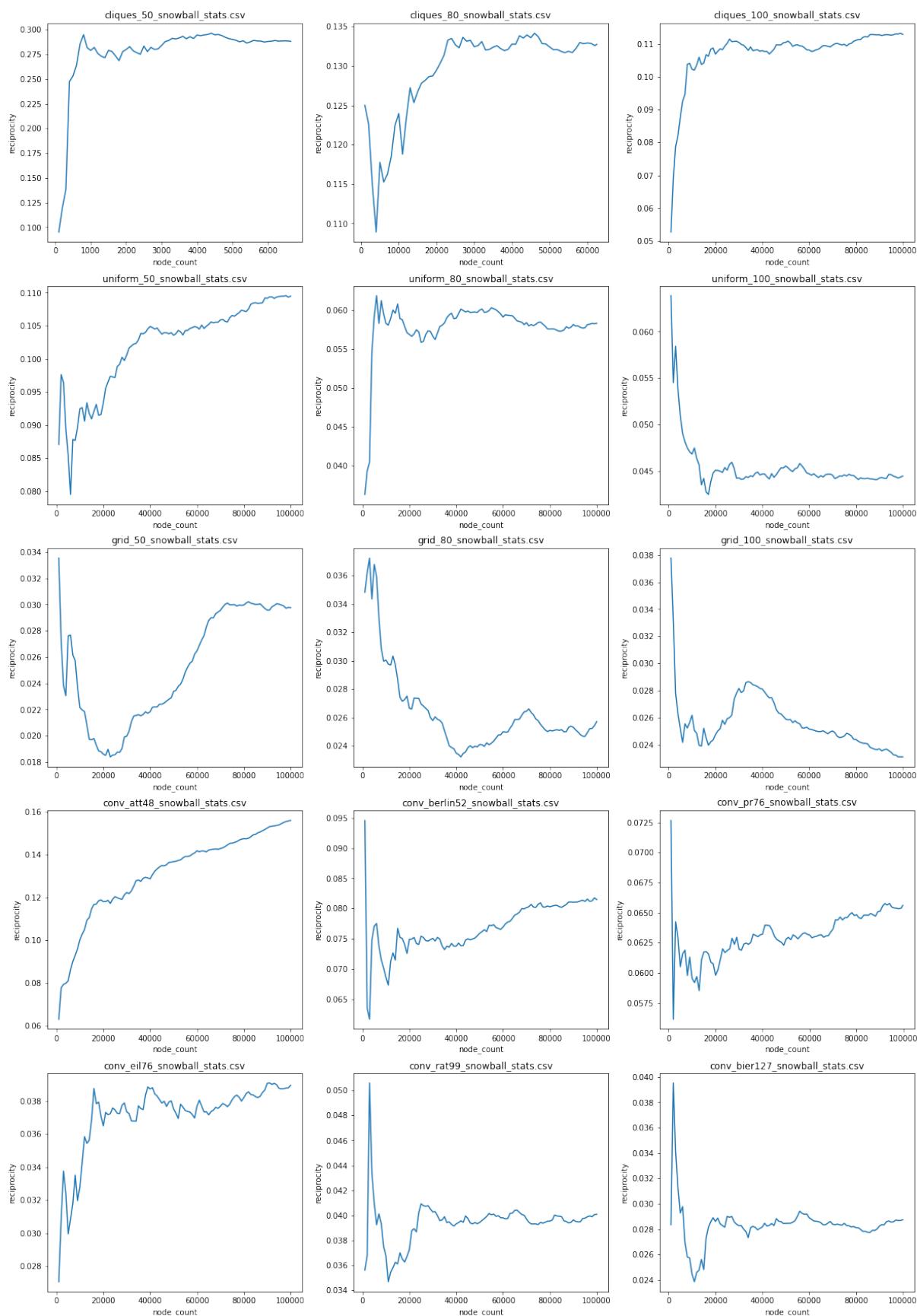
Rysunek 3.8 Współczynnik conrelw zależności od liczby wierzchołków - próbkowanie snowball



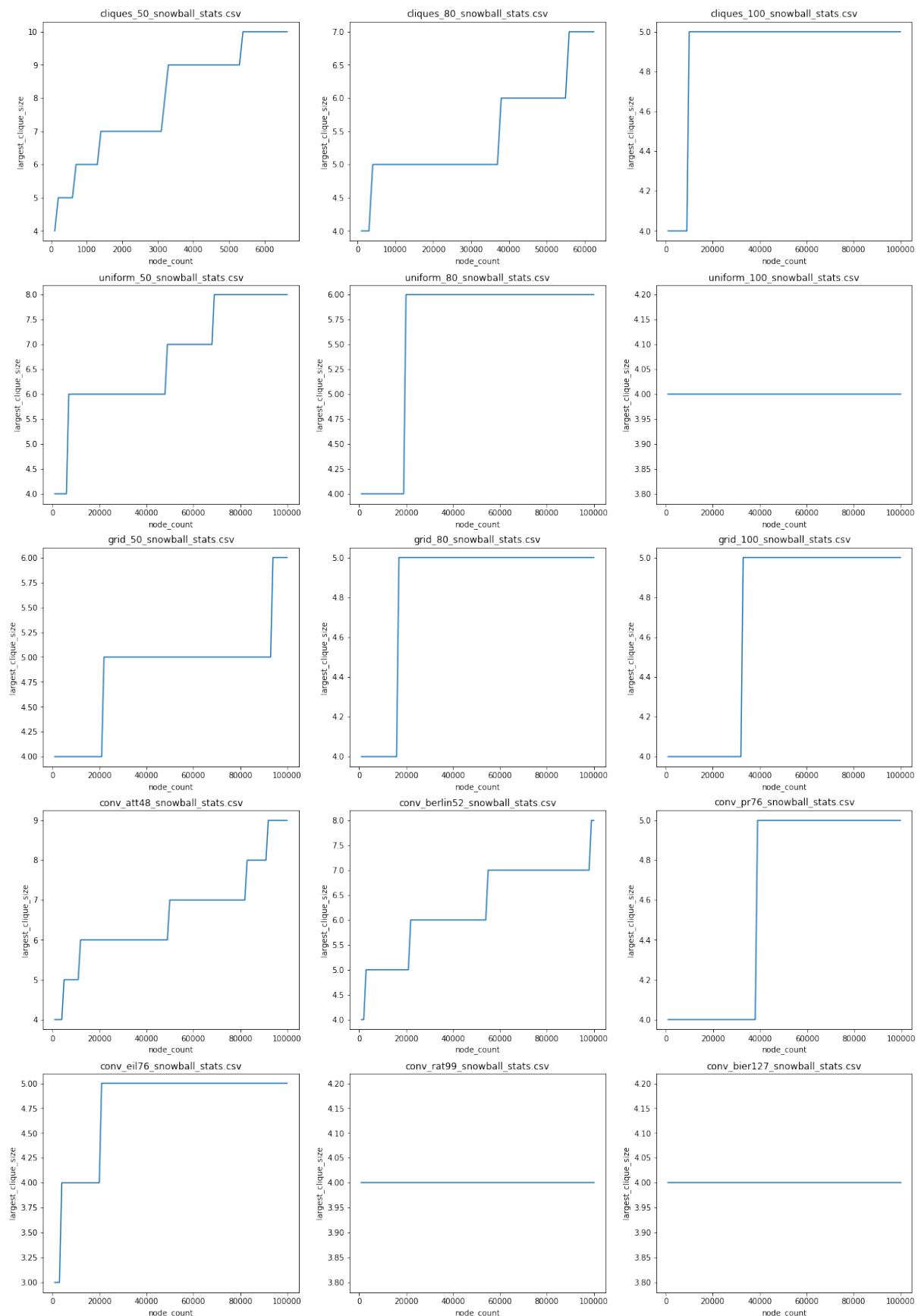
Rysunek 3.9 Gęstość grafów zależności od liczby wierzchołków - próbkowanie snowball



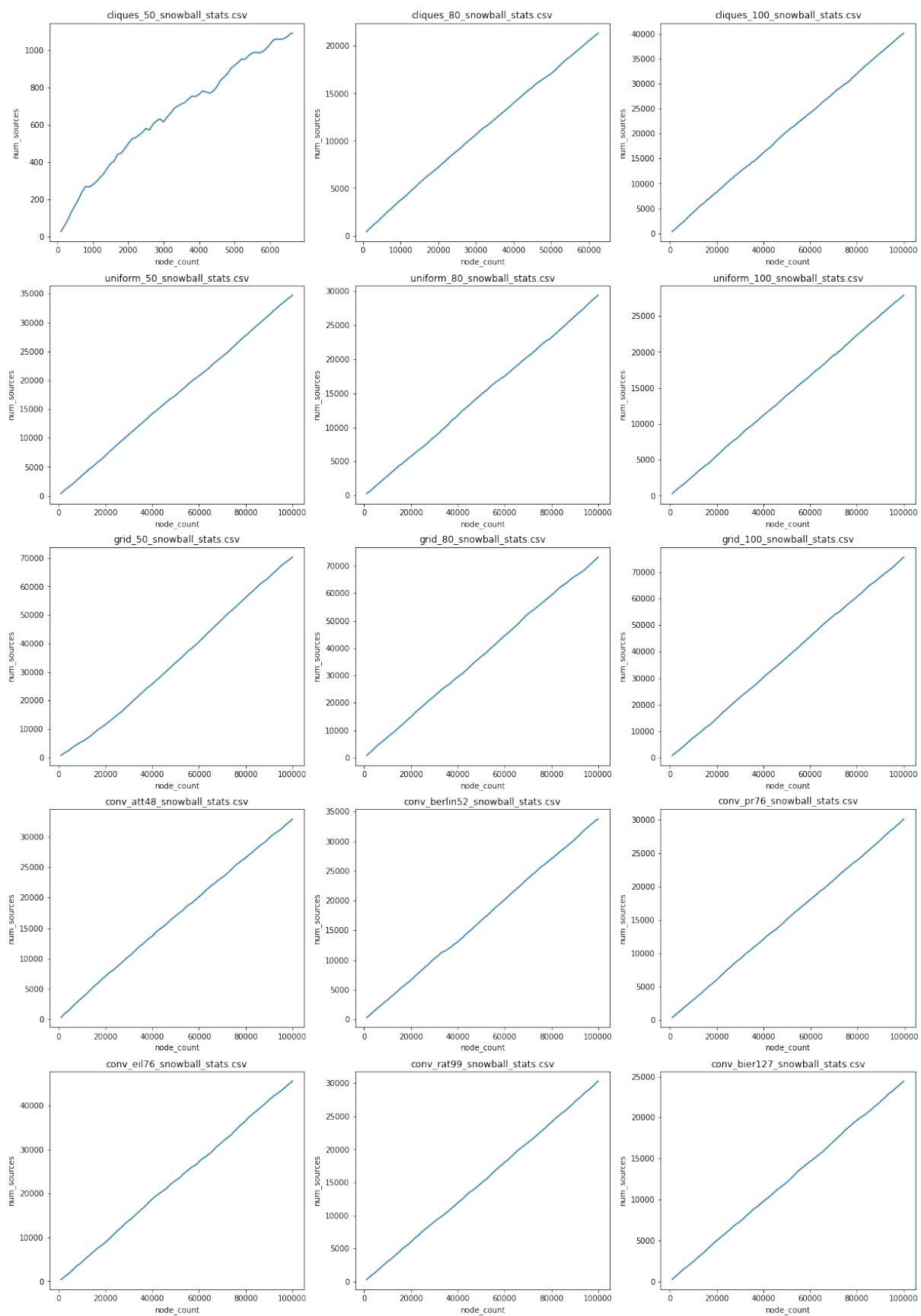
Rysunek 3.10 Współczynnik distLOw zależności od liczby wierzchołków - próbkowanie snowball



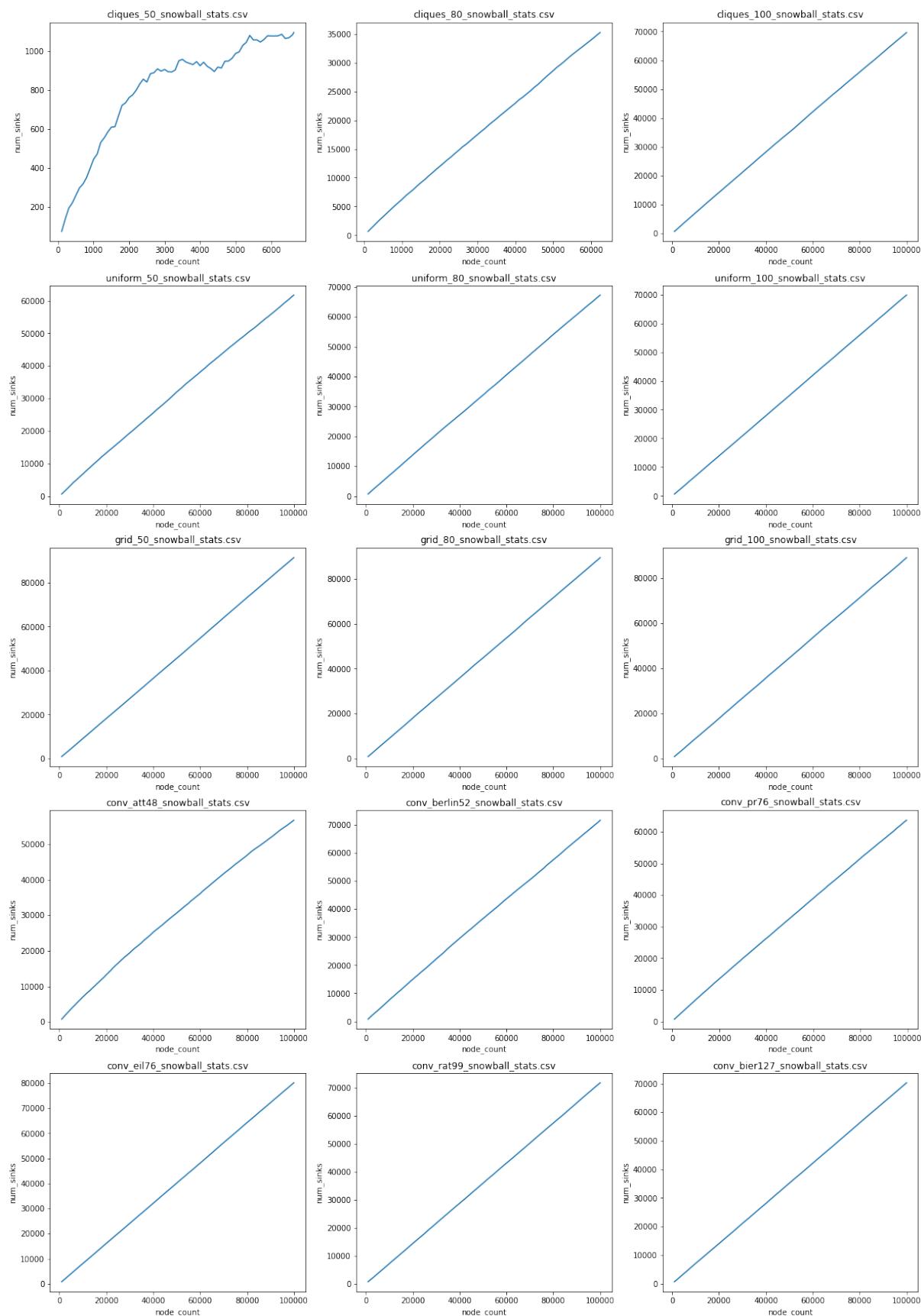
Rysunek 3.11 Współczynnik wzajemności grafów zależność od liczby wierzchołków - próbkowanie snowball



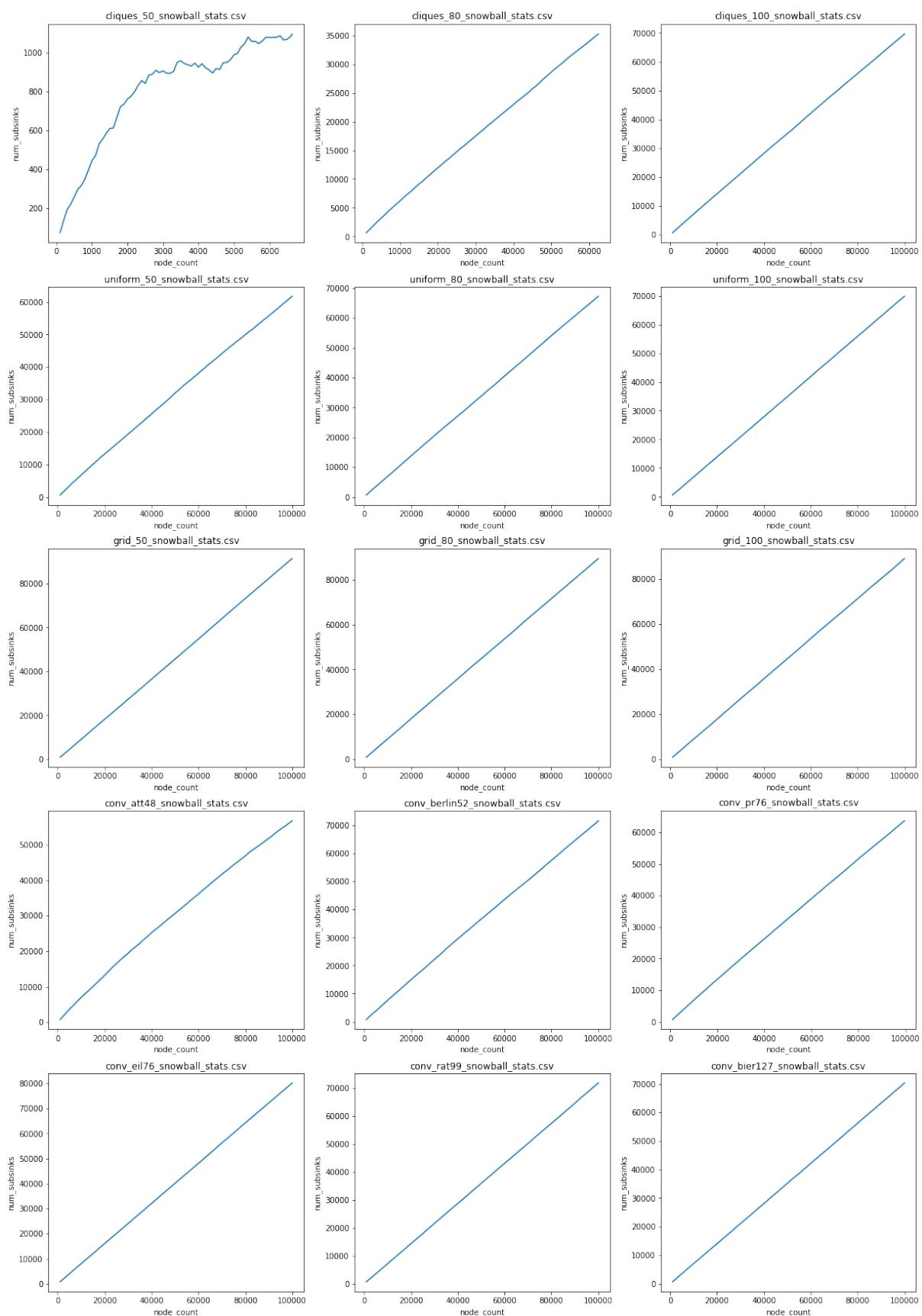
Rysunek 3.12 Rozmiar największej kliki w grafach zależności od liczby wierzchołków - próbkowanie snowball



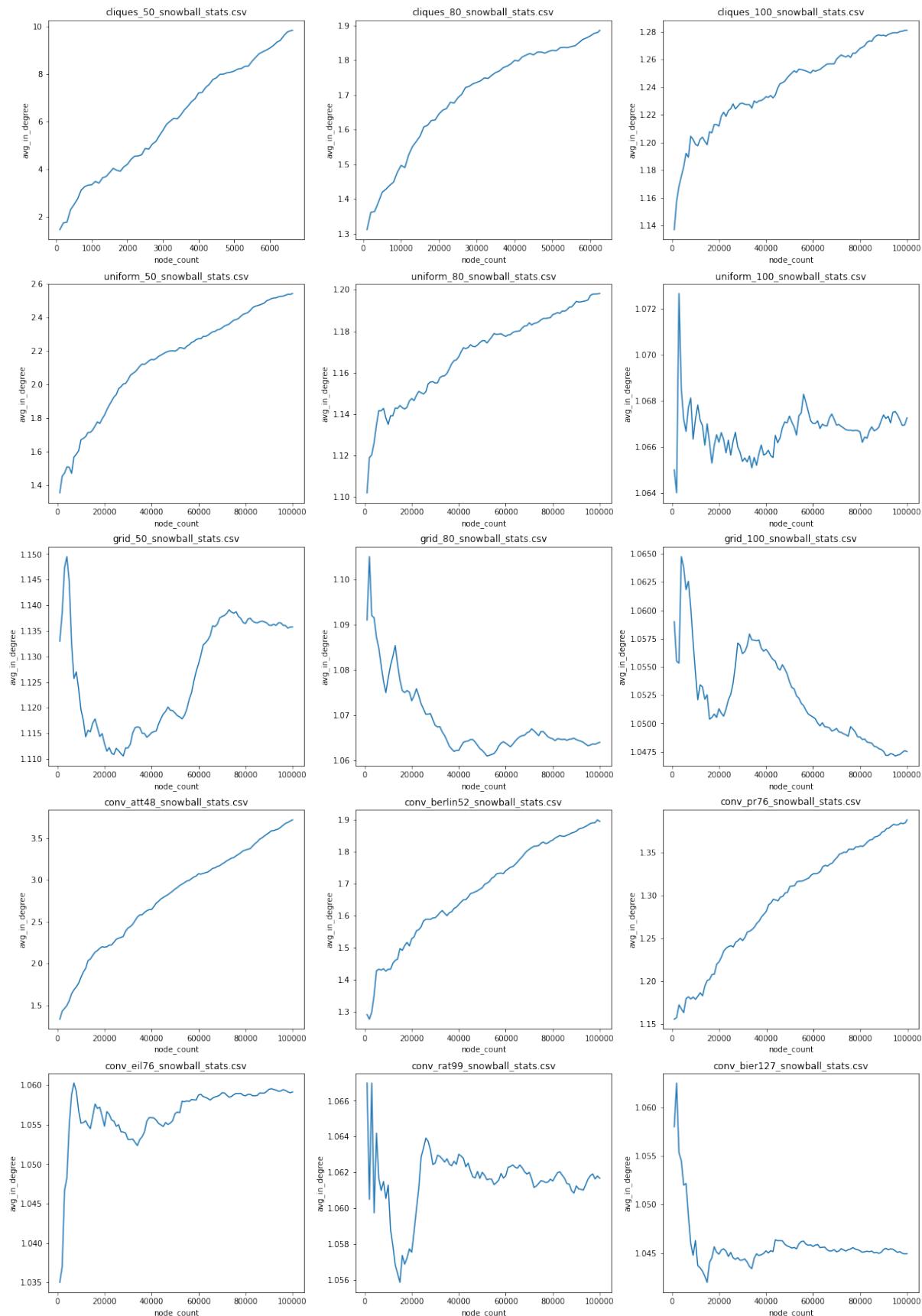
Rysunek 3.13 Liczba źródeł w grafach zależności od liczby wierzchołków - próbkowanie snowball



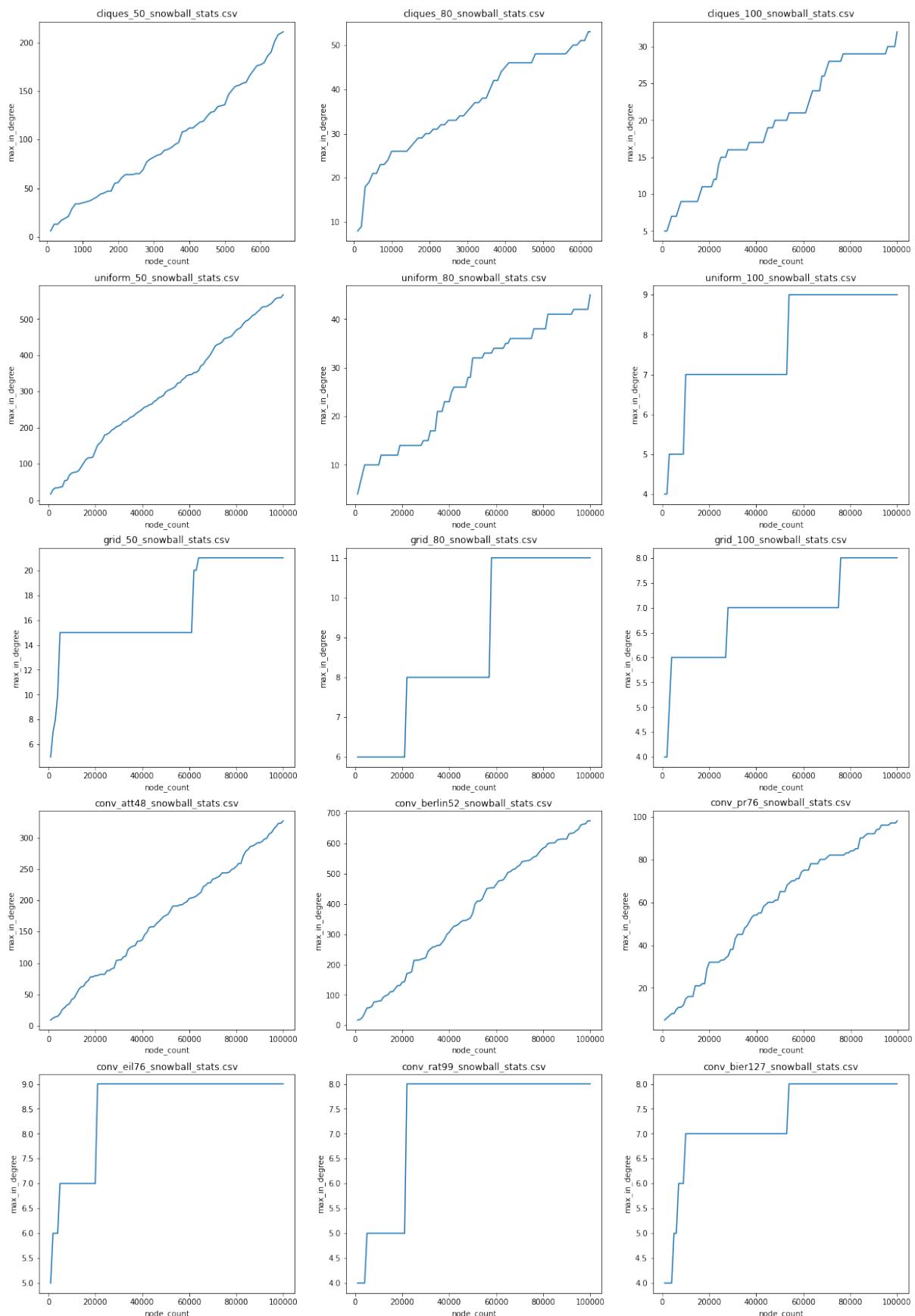
Rysunek 3.14 Liczba ścieków w grafach zależności od liczby wierzchołków - próbkowanie snowball



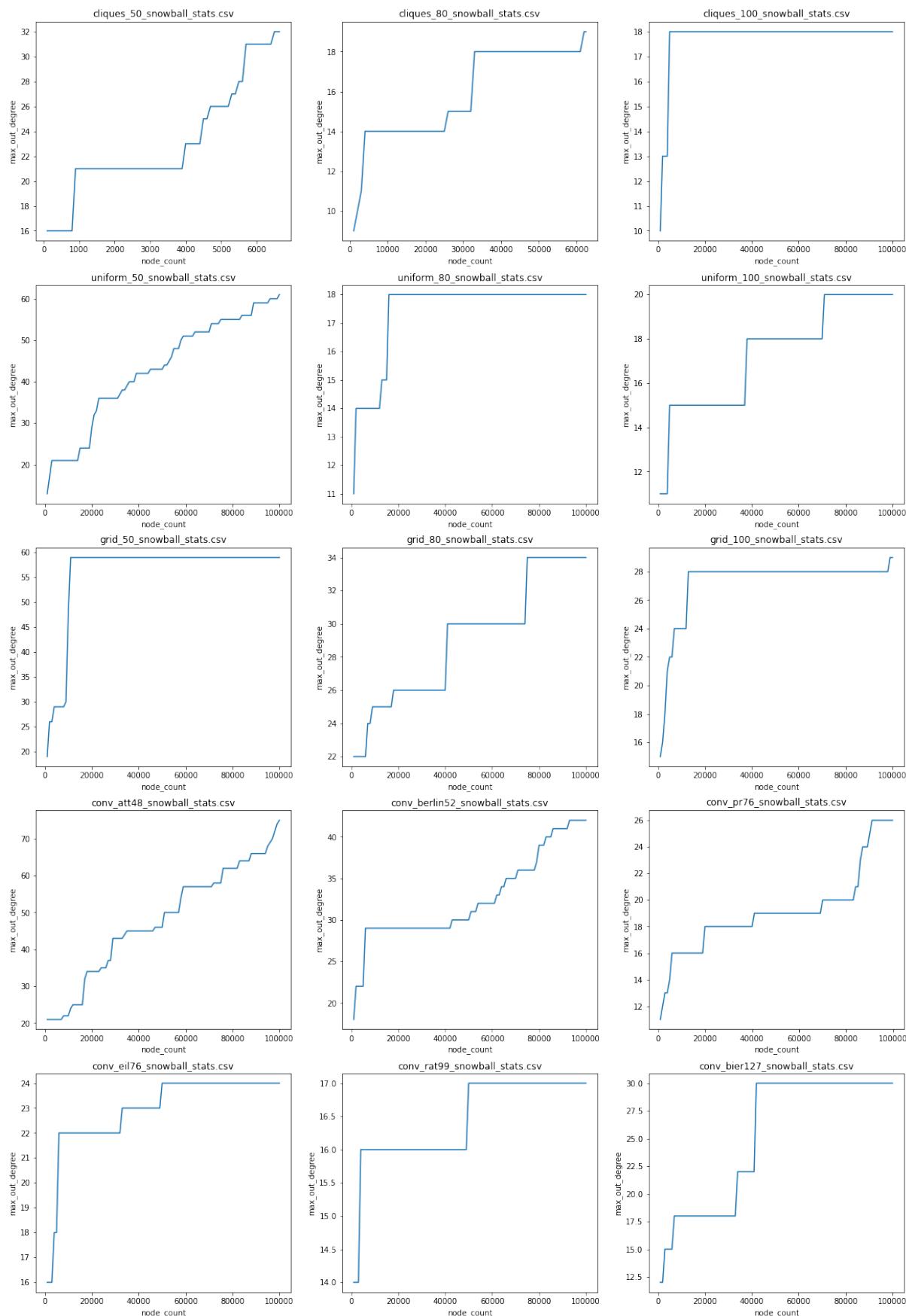
Rysunek 3.15 Liczba subsinks w grafach zależności od liczby wierzchołków - próbkowanie snowball



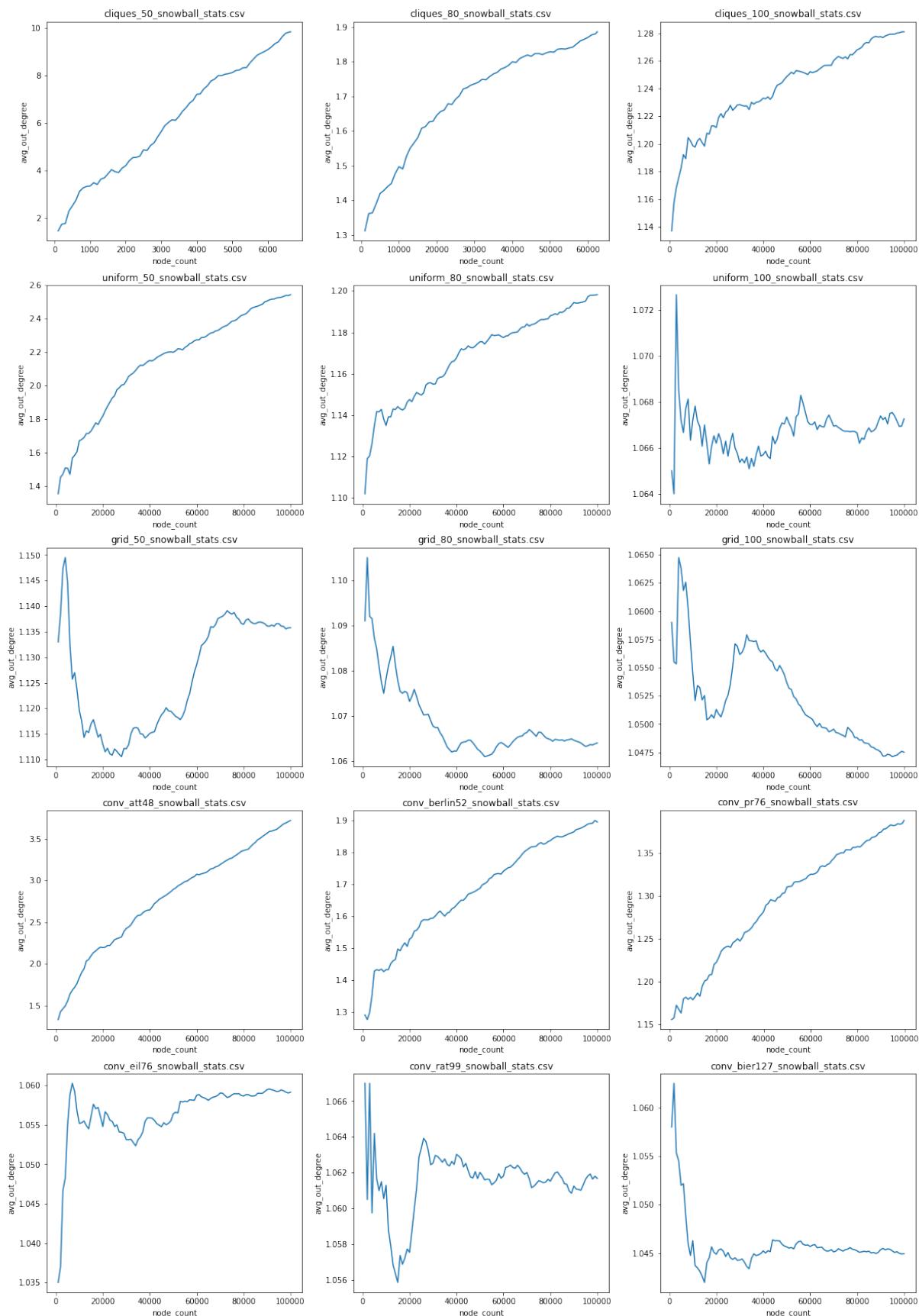
Rysunek 3.16 Średni stopień wchodzący wierzchołków w grafach zależności od liczby wierzchołków - próbkowanie snowball



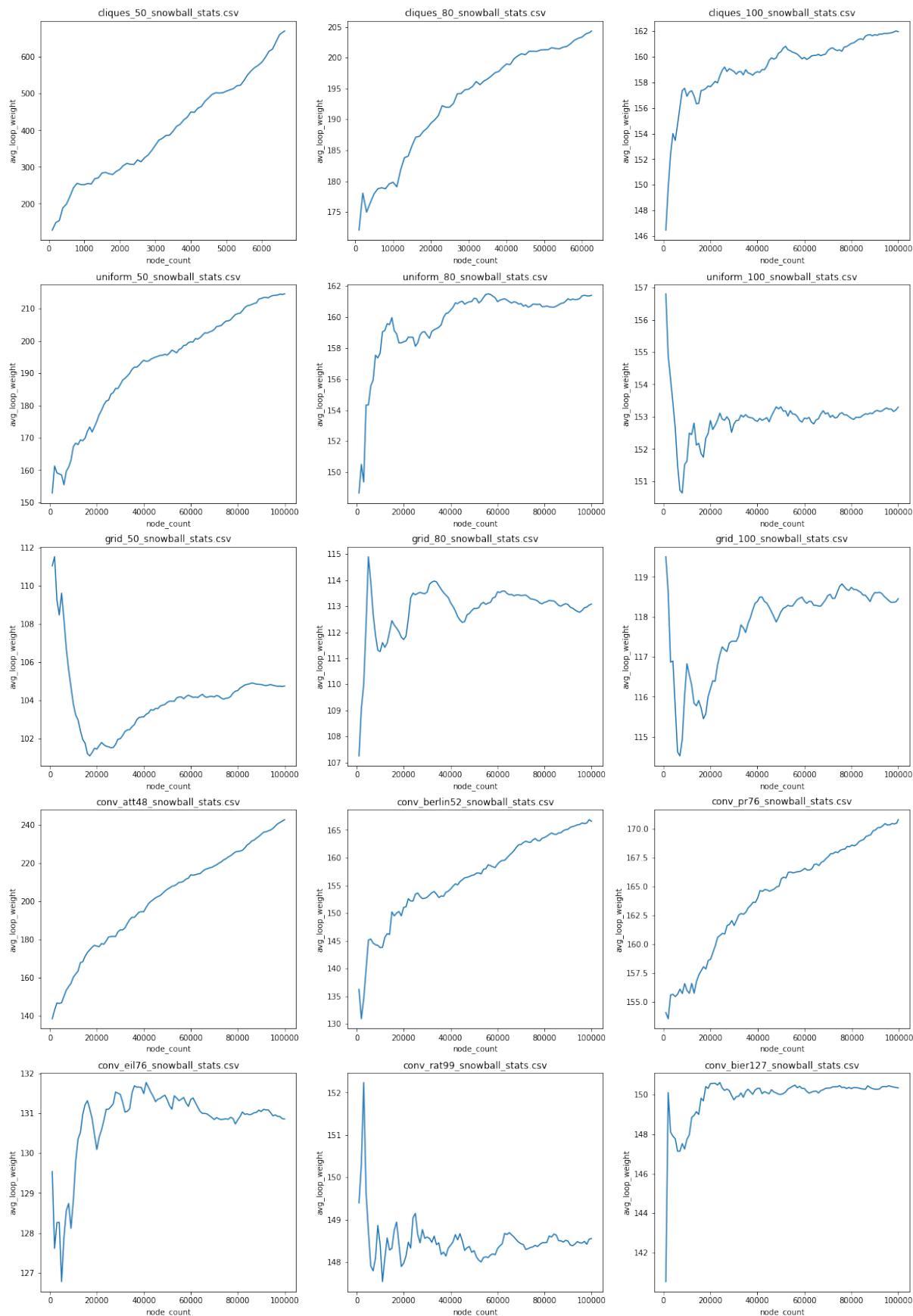
Rysunek 3.17 Maksymalny stopień wchodzący wśród wierzchołków w grafach zależności od liczby wierzchołków - próbkowanie snowball



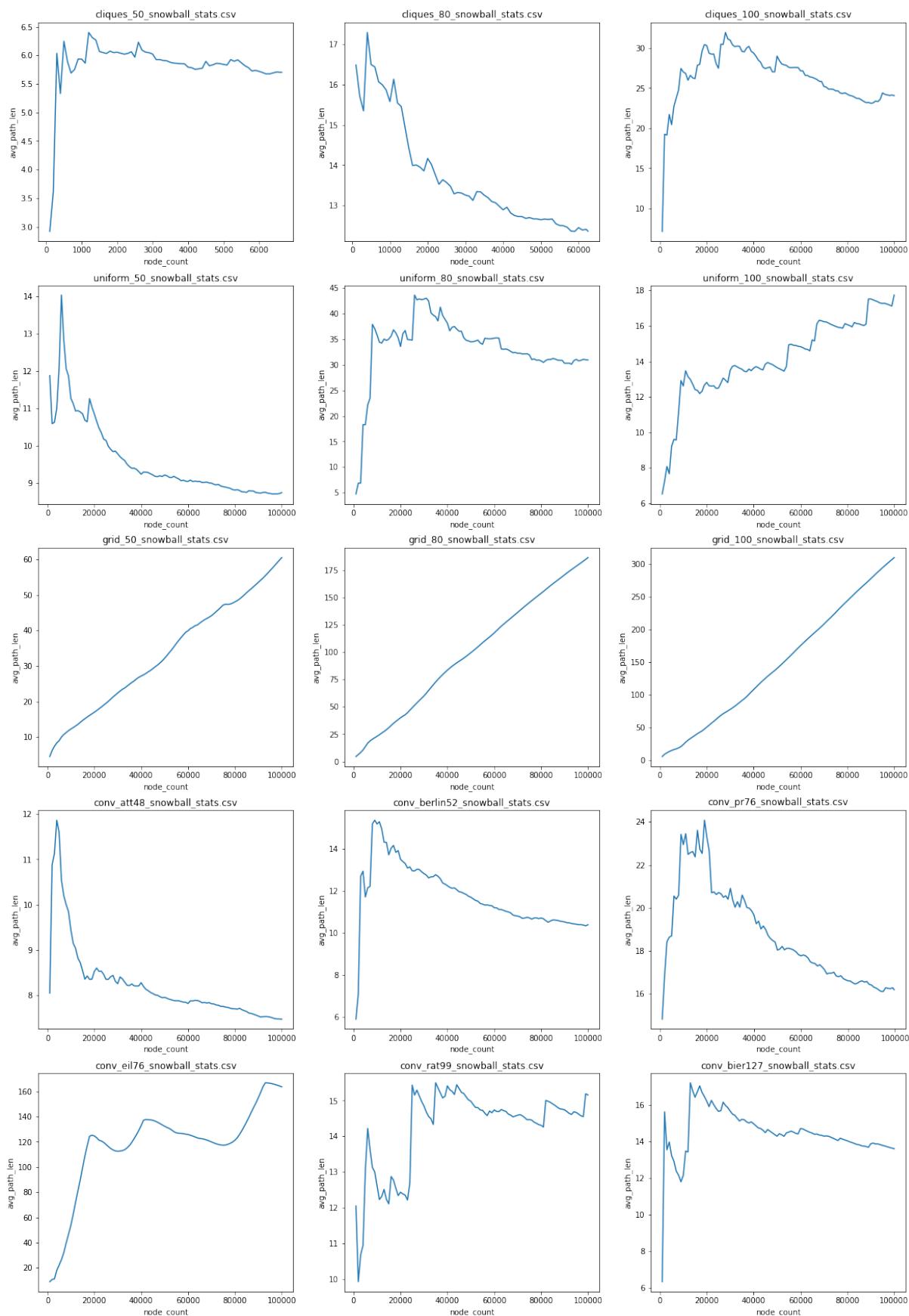
Rysunek 3.18 Średni stopień wychodzący wierzchołków w grafach zależności od liczby wierzchołków - próbkowanie snowball



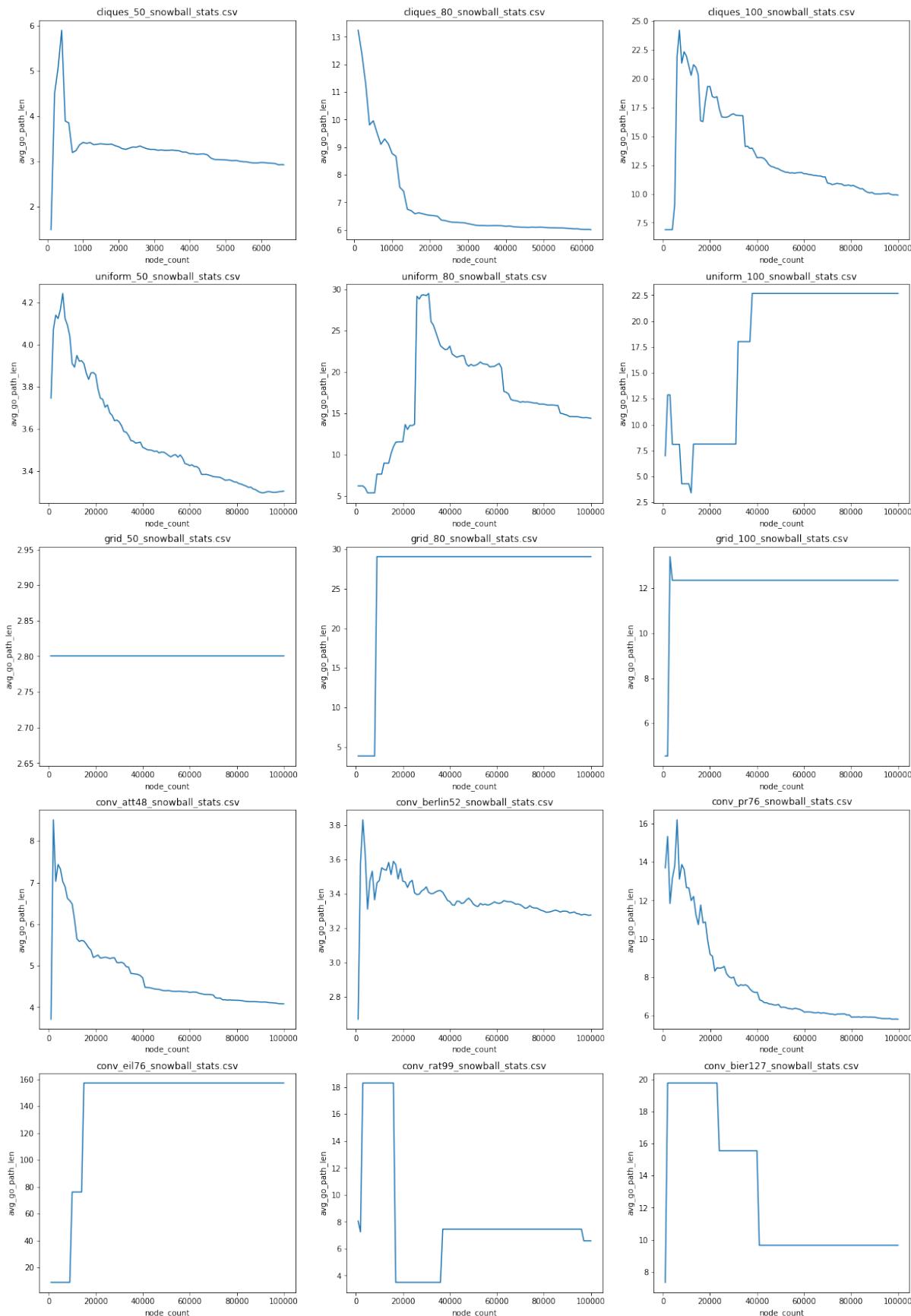
Rysunek 3.19 Maksymalny stopień wychodzący wśród wierzchołków w grafach zależności od liczby wierzchołków - próbkowanie snowball



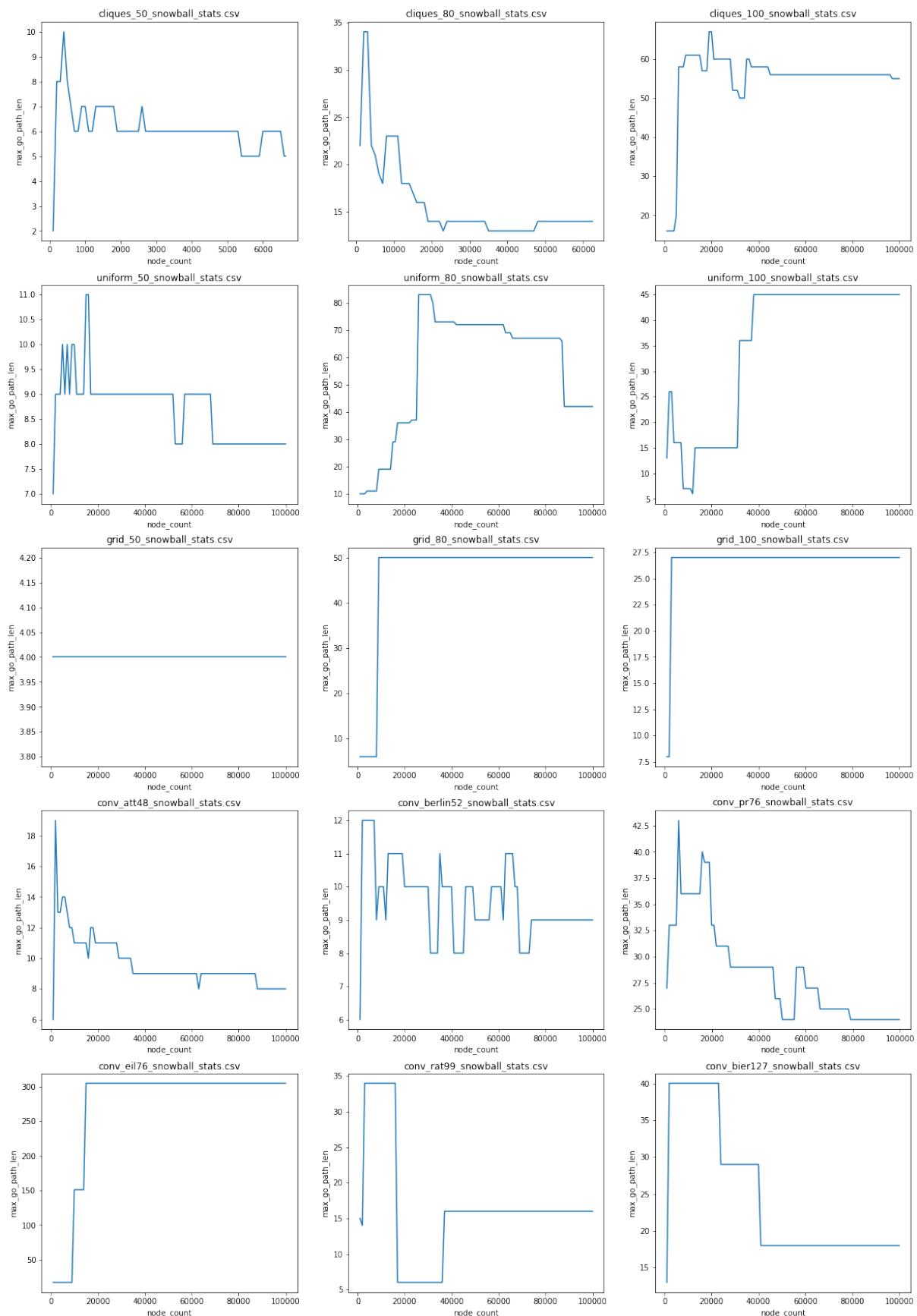
Rysunek 3.20 Średnia waga pętli w grafach zależności od liczby wierzchołków - próbkowanie snowball



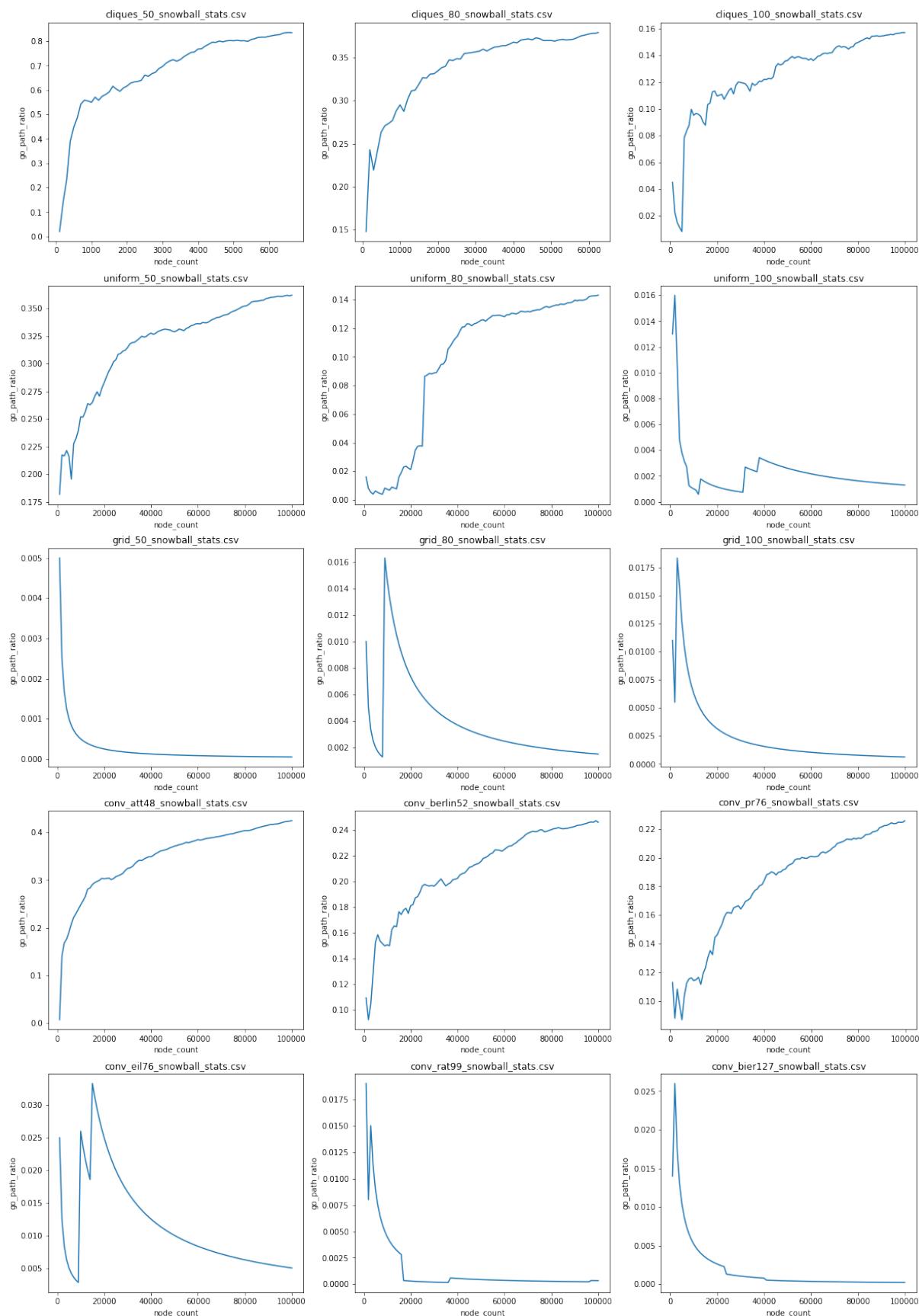
Rysunek 3.21 Średnia waga pętli w grafach zależności od liczby wierzchołków - próbkowanie snowball



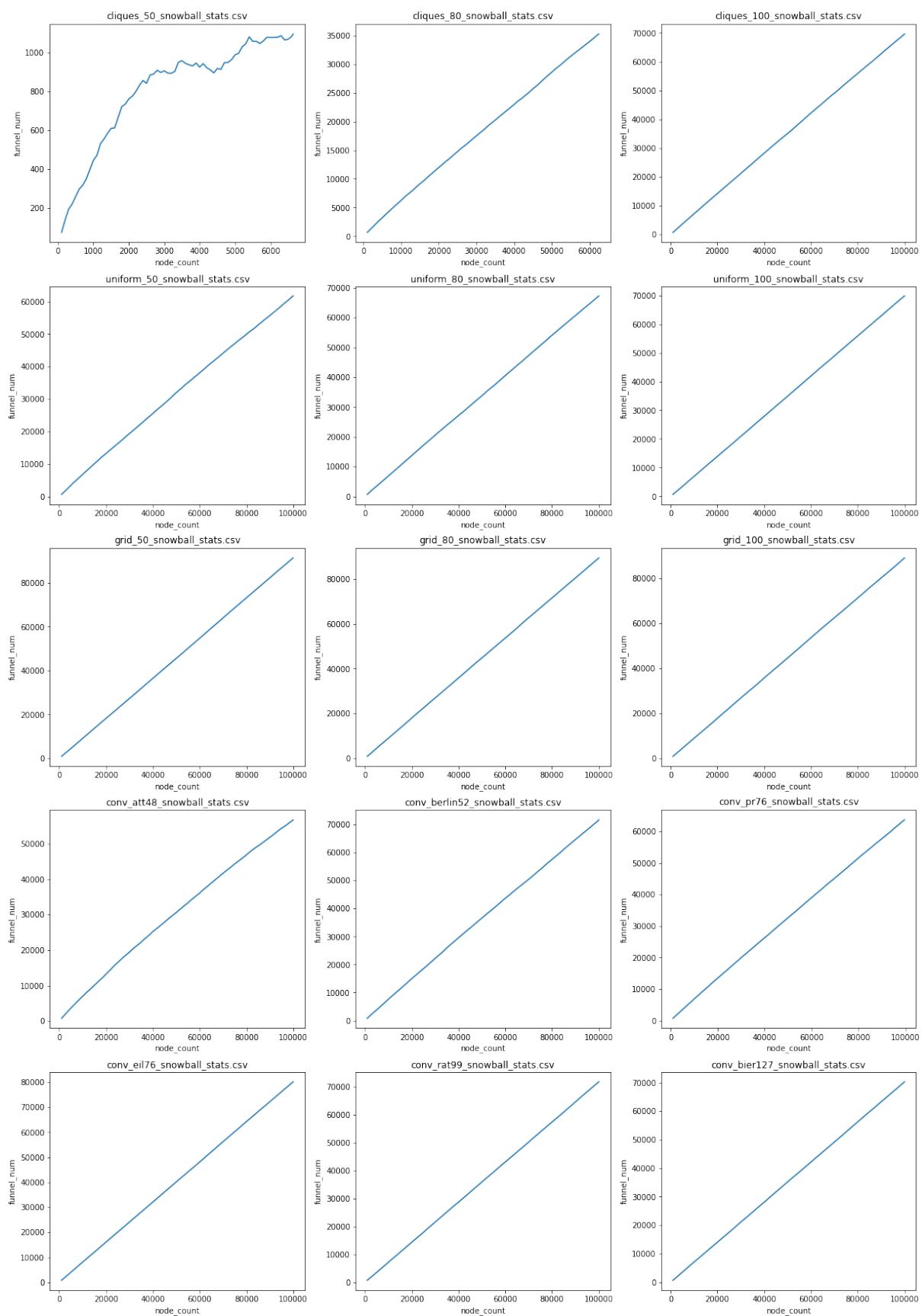
Rysunek 3.22 Średnia długość istniejących ścieżek do globalnego optimumw zależności od liczby wierzchołków - próbkowanie snowball



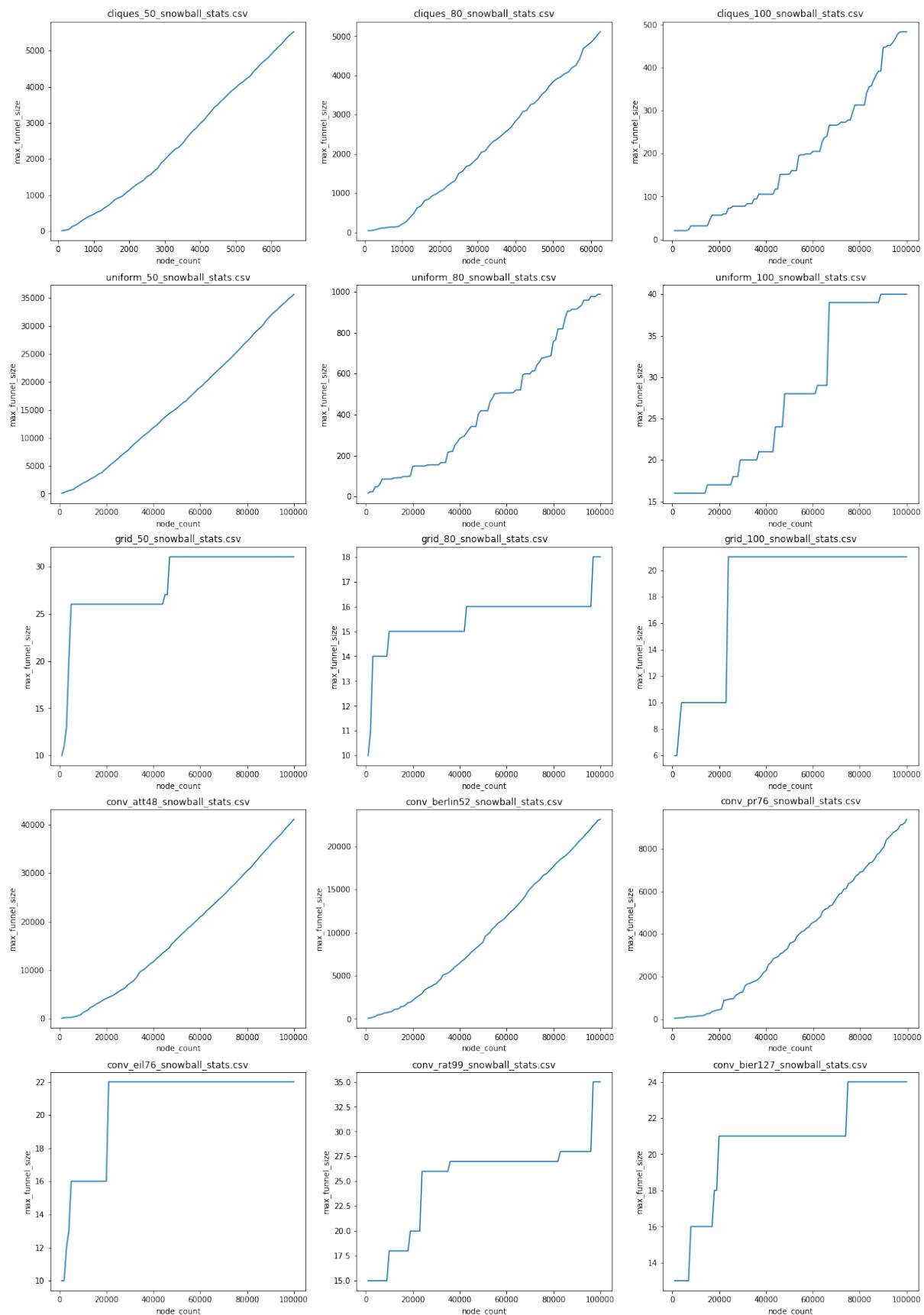
Rysunek 3.23 Długość najdłuższej istniejącej ścieżki do globalnego optimum w zależności od liczby wierzchołków - próbkowanie snowball



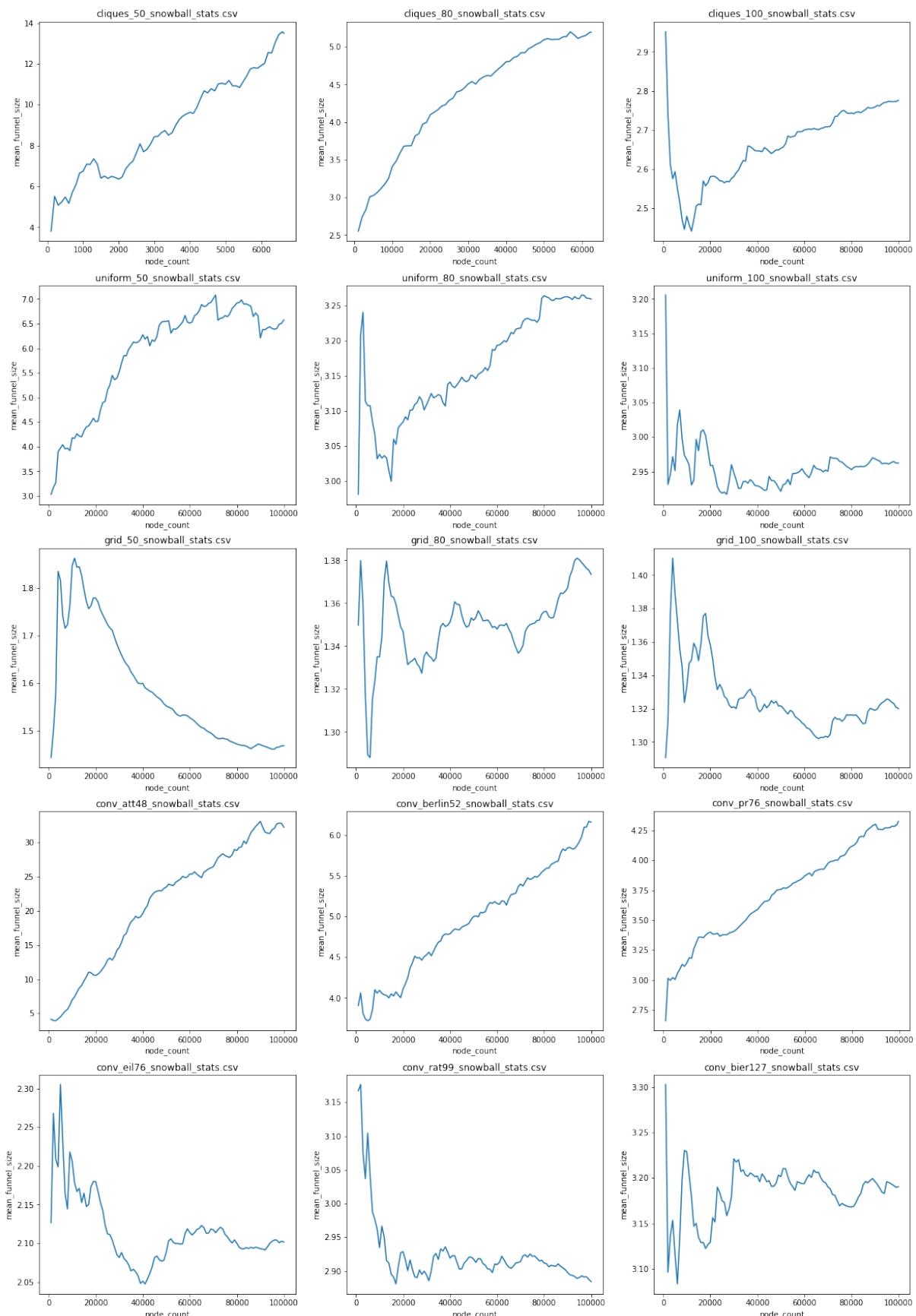
Rysunek 3.24 Stosunek liczby wierzchołków, z których istnieje ścieżka do globalnego optimum do liczby wszystkich wierzchołków w zależności od liczby wierzchołków - próbkowanie snowball



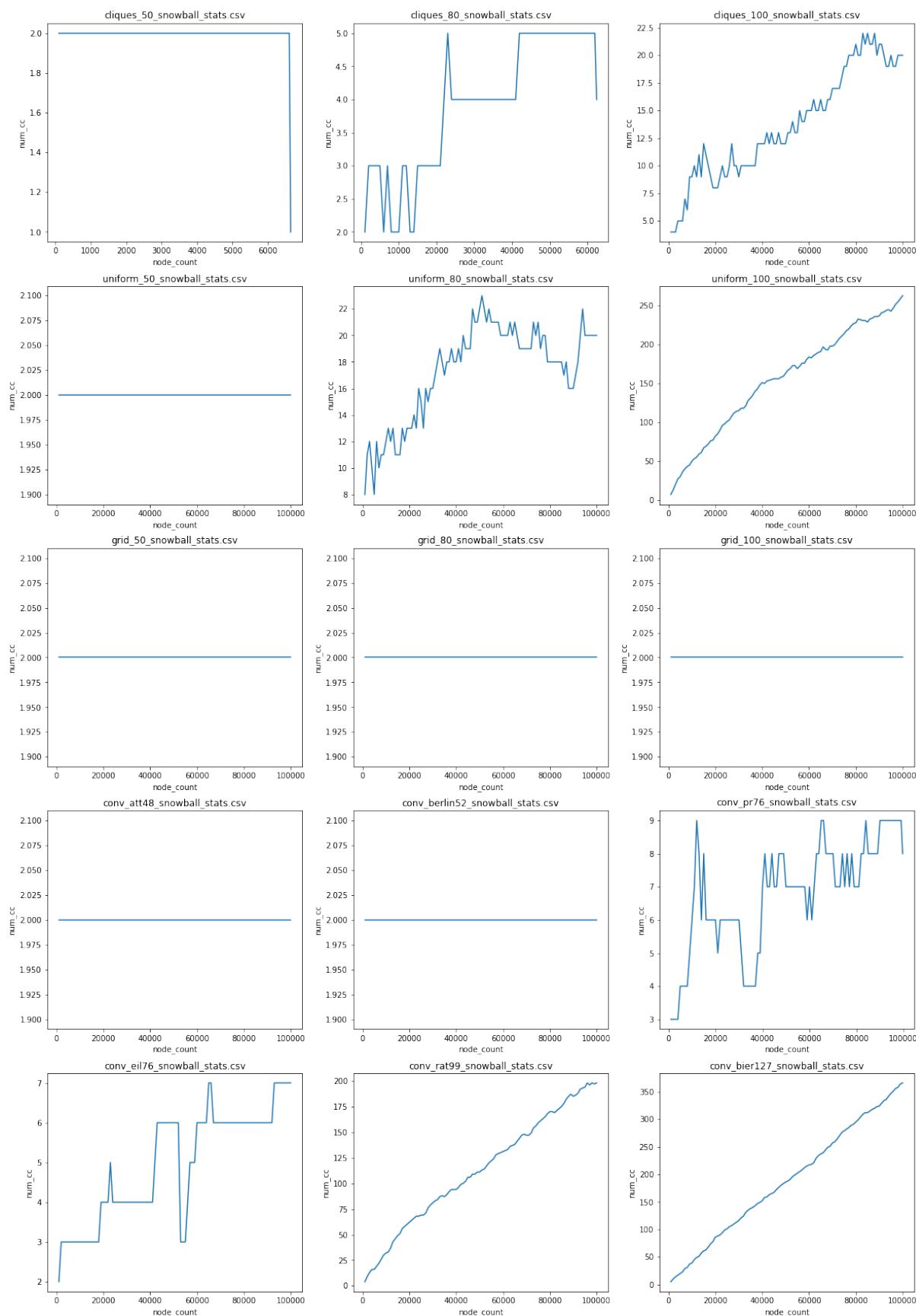
Rysunek 3.25 Liczba lejów w przestrzeni zależności od liczby wierzchołków - próbkowanie snowball



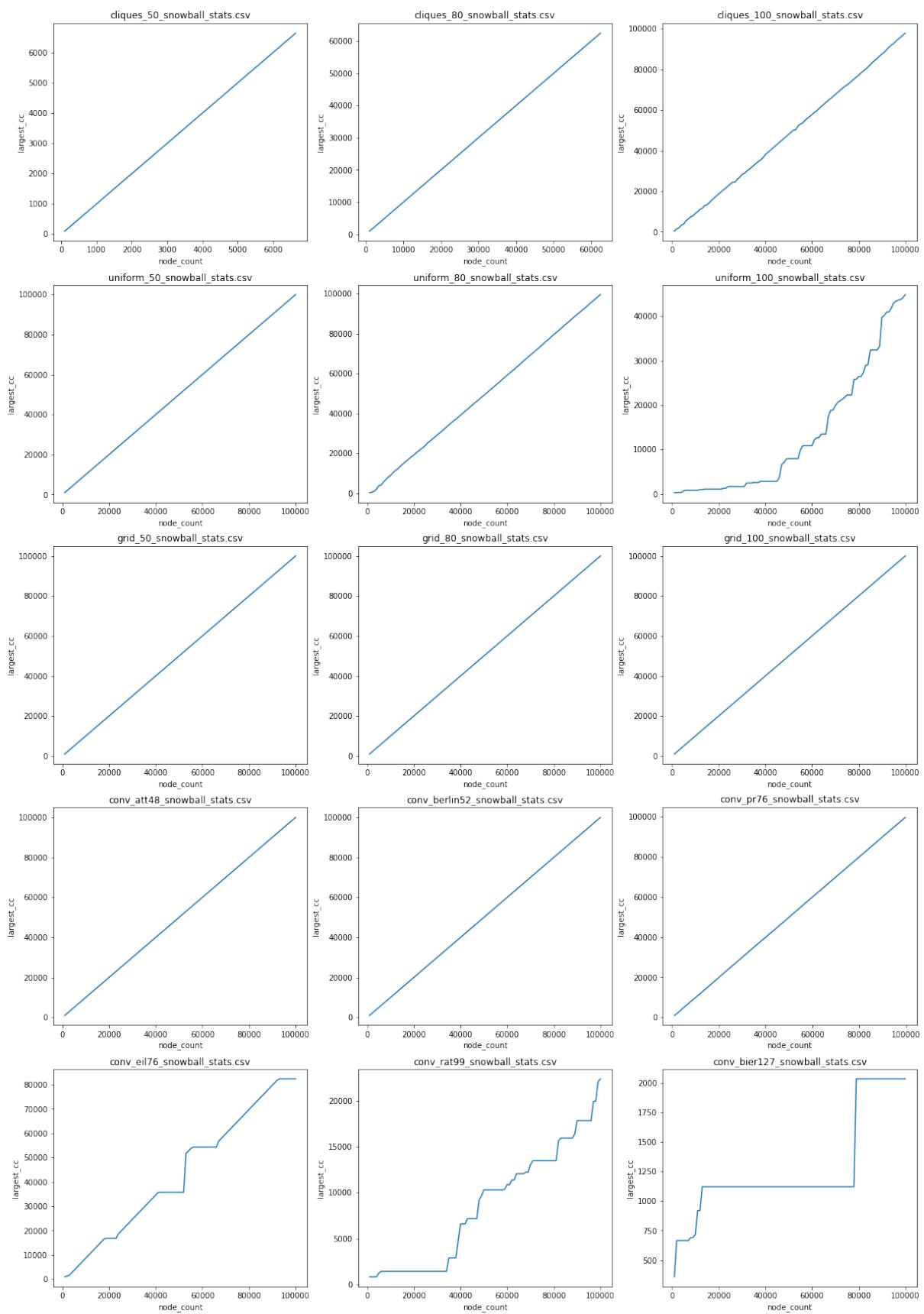
Rysunek 3.26 Rozmiar największego leja w przestrzeni zależności od liczby wierzchołków - próbkowanie snowball



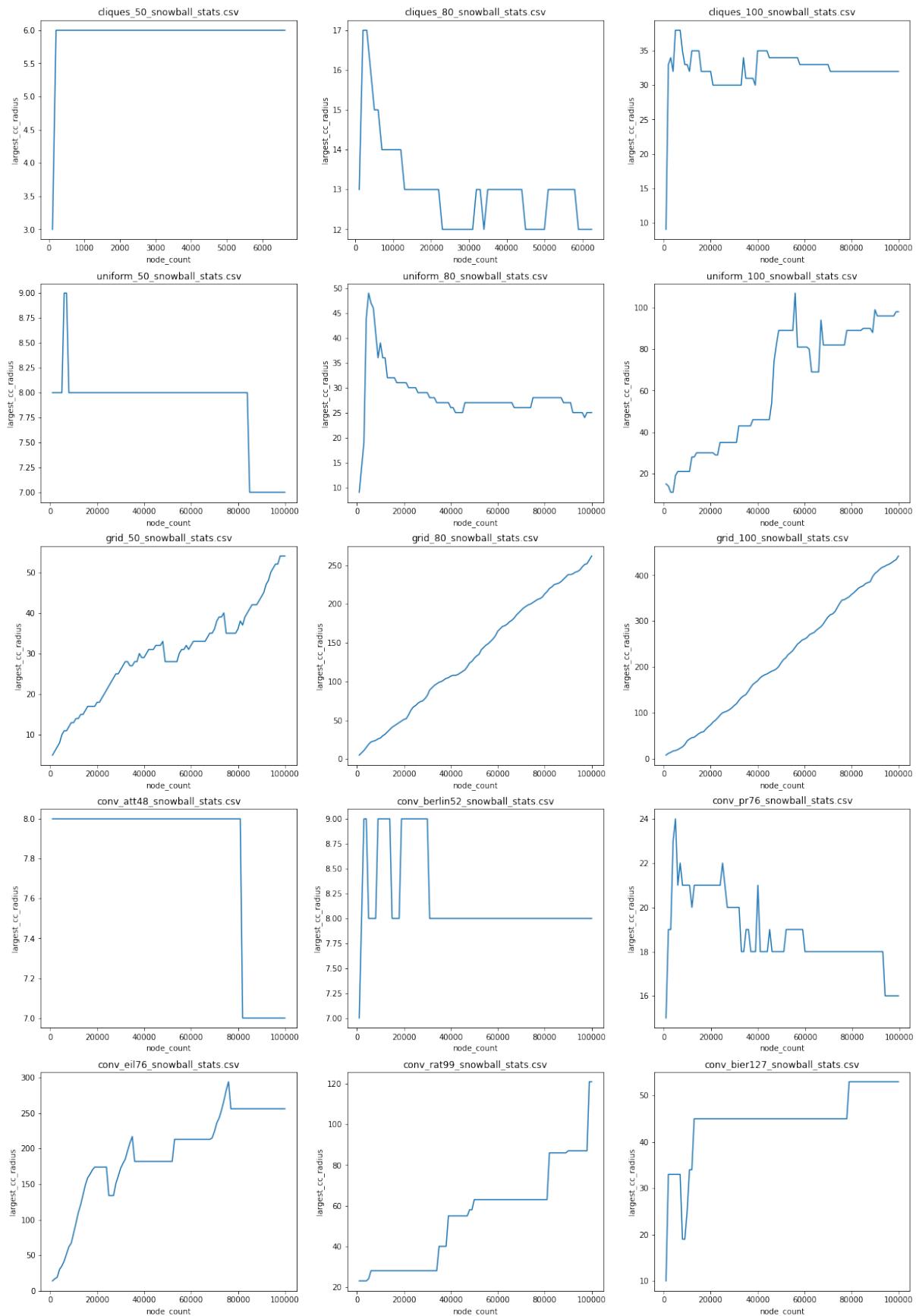
Rysunek 3.27 Średni rozmiar lejów w przestrzeni zależności od liczby wierzchołków - próbkowanie snowball



Rysunek 3.28 Liczba spójnych podgrafów zależności od liczby wierzchołków - próbkowanie snowball



Rysunek 3.29 Rozmiar największego spójnego podgrafu zależności od liczby wierzchołków - próbkowanie snowball



Rysunek 3.30 Promień największego spójnego podgrafu zależności od liczby wierzchołków - próbkowanie snowball

Rozdział 4

Opis implementacji

Do realizacji badania konieczne było stworzenie odpowiedniego oprogramowania. Kod źródłowy oprogramowania jest publicznie dostępny na platformie Github pod adresem https://github.com/gero0/FLA_research.

Opracowano następujące narzędzia:

- Program wykonujący próbkowanie przestrzeni rozwiązań "tsp_samplers",
- Program obliczający wartości miar na podstawie spróbkowanej przestrzeni "stat_calculator",
- Generatory instancji testowych,
- Skrypty pomocnicze.

4.1 Program próbujący

Ponieważ próbkowanie przestrzeni rozwiązań wymaga dużego nakładu obliczeniowego, implementację algorytmów wykonano w języku Rust. Rust jest językiem kompilowanym do kodu maszynowego, wspierającym programowanie wielowątkowe.

4.1.1 Uruchamianie i parametry

Program ma postać pliku wykonywalnego uruchamianego z linii poleceń. Program wymaga podania nazwy pliku wejściowego, nazwę algorytmu oraz wartości parametrów tego algorytmu. Opcjonalnie jako parametr możliwe jest podanie nazwy folderu, do którego mają zostać zapisane wyniki próbkowania, jeśli nie zostanie ona podana program zapisze wyniki do folderu o nazwie "<algorytm>_latest".

Format komendy wygląda więc następująco:

```
1  tsp_samplers <plik_wejsciowy> [plik_wyjsciowy] \
2  <algorytm> <parametry_algorytmu>
```

Akceptowane wartości parametru "algorytm" to:

- tp - próbkowanie dwufazowe
- snowball - próbkowanie snowball
- exhaustive - przegląd zupełny

Algorytm przeglądu zupełnego nie przyjmuje żadnych parametrów wejściowych.

Algorytm dwufazowy przyjmuje parametry w następującej kolejności:

1 <ITERS> <N_MAX> <N_ATT> <E_ATT> <MUT_D>

Gdzie:

- ITERS (n_{runs}) - liczba iteracji,
- N_MAX (n_{max}) - żądana liczba wierzchołków,
- N_ATT (n_{att}) - liczba prób generowania wierzchołków,
- E_ATT (e_{att}) - liczba prób generowania krawędzi,
- MUT_D (D) - parametr D

Algorytm został opisany szczegółowo w sekcji 3.1.1 i listingu 2.

Algorytm *snowball* przyjmuje parametry w następującej kolejności:

1 <WALK_LEN> <N_EDGES> <DEPTH> <MUT_D> <SAVE_THRESHOLD>

Gdzie:

- WALK_LEN (w_{len}) - długość losowego spaceru
- N_EDGES (m) - liczba prób przeszukania sąsiedztwa
- DEPTH> ($depth$) - głębokość przeszukiwania
- MUT_D (D) - parametr D
- SAVE_THRESHOLD (s_{thresh}) - interwał zapisu

Algorytm został opisany szczegółowo w sekcji 3.1.2 i listingu 3.

4.1.2 Pliki wejściowe i wyjściowe

Plik wejściowy musi być plikiem txt w następującym formacie:

- Linia 1 - nazwa instancji
- Linia 2 - liczba miast w instancji
- N kolejnych linii - Pełna macierz odległości pomiędzy miastami podanych w postaci liczb całkowitych. Kolejne wartości w rzędzie macierzy rozdzielone spacją, wiersze rozdzielone znakiem nowej linii

Przykładowy plik wejściowy przedstawiającyinstancję burma14 ze zbioru TSPLIB został przedstawiony na listingu 4.1.

Listing 4.1 Instancja burma14 w formacie akceptowanym przez program

```
1 burma14
2 14
3 0 153 510 706 966 581 455 70 160 372 157 567 342 398
4 153 0 422 664 997 598 507 197 311 479 310 581 417 376
5 510 422 0 289 744 390 437 491 645 880 618 374 455 211
6 706 664 289 0 491 265 410 664 804 1070 768 259 499 310
7 966 997 744 491 0 400 514 902 990 1261 947 418 635 636
```

```

8      581 598 390 265 400 0 168 522 634 910 593 19 284 239
9      455 507 437 410 514 168 0 389 482 757 439 163 124 232
10     70 197 491 664 902 522 389 0 154 406 133 508 273 355
11     160 311 645 804 990 634 482 154 0 276 43 623 358 498
12     372 479 880 1070 1261 910 757 406 276 0 318 898 633 761
13     157 310 618 768 947 593 439 133 43 318 0 582 315 464
14     567 581 374 259 418 19 163 508 623 898 582 0 275 221
15     342 417 455 499 635 284 124 273 358 633 315 275 0 247
16     398 376 211 310 636 239 232 355 498 761 464 221 247 0

```

Wyniki programu zapisuje okresowo do docelowego folderu w plikach "samples_<i>.json". Pliki zawierają następujące informacje:

- nodes - lista wierzchołków grafu. Każdy wierzchołek opisywany jest 3-elementową listą: [unikalny identyfikator, rozwiązanie, wartość funkcji celu]. Identyfikator oraz wartość funkcji celu są liczbami całkowitymi, rozwiązanie ma zaś postać listy identyfikatorów miast w kolejności, w jakiej przebiega przez nie trasa,
- edges - lista krawędzi skierowanych. Każda krawędź jest opisane 3-elementową listą - [id. wierzchołka początkowego, id. wierzchołka docelowego, waga],
- opt_count - liczba wywołań funkcji optymalizującej (2opt),
- oracle_count - liczba wykonanych obliczeń długości ścieżki,
- time_ms - czas działania programu podany w ms. Liczony jest jedynie czas spędzony na próbkowaniu (bez liczenia czasu spędzonego na zapisywanie wyników),
- comment - komentarz, zawiera parametry, z którymi uruchomiono algorytm,
- missed - tylko dla próbkowania dwufazowego, licznik prób dodania krawędzi, które zakończyły się niepowodzeniem z powodu braku wierzchołka należącego do krawędzi w zbiorze spróbkowanych wierzchołków.

Przykładowy plik wyjściowy programu został przedstawiony na listingu 4.2.

Listing 4.2 Przykład pliku wyjściowego po krótkim próbkowaniu małej instancji

```

1  {
2  "opt_count":101802,
3  "oracle_count":5114163,
4  "time_ms":199,
5  "comment":"n_max:100 n_att:100 e_att:100 iters:10 file:cliques_6",
6  "missed":0,
7  "nodes": [
8  [1,[0, 1, 4, 5, 3, 2],1622],
9  [0,[0, 2, 3, 5, 4, 1],1622]],
10 "edges": [
11 [0,0,335],
12 [1,0,93],
13 [0,1,665],
14 [1,1,907]]
15 }

```

4.2 Program obliczający wartości miar

Program obliczający wartości miar został napisany w języku Python. Większość miar grafu obliczone zostało przy pomocy biblioteki igraph. Aby zwiększyć wydajność, do obliczeń metryk **avg_loop_weight** i **num_sub sinks** wykorzystano moduł natywny napisany w języku Rust.

4.2.1 Uruchamianie i parametry

Program jest uruchamiany przy użyciu interpretera języka Python, jako argument przyjmuje ścieżkę do folderu zawierającego pliki z próbami. Opcjonalnymi parametrami są: nazwa pliku wyjściowego limit plików wejściowych (-l), oraz nazwy miar do policzenia (-s, -stats). Jeśli nazwa pliku wyjściowego nie zostanie podana, wyniki zapisywane są do pliku "results.csv". Jeśli limit nie zostanie podany, obliczenia są wykonywane dla wszystkich plików w folderze. Jeśli nie podane zostaną żadne nazwy miar do policzenia, domyślnym działaniem jest obliczenie ich wszystkich.

Format komendy wygląda następująco:

```
1 python stat_calculator/main.py <folder_wejsciowy> \
2 [plik_wyjsciowy] [-l <N>] [—stats ...]
```

4.2.2 Pliki wejściowe i wyjściowe

Program na wejście przyjmuje ścieżkę do folderu. W wybranym folderze wyszukiwane są pliki JSON wygenerowane przez program próbujący. Pliki są sortowane na podstawie numeracji (natsort) i dla każdego wykonywany jest proces wczytania danych, utworzenia grafu LON i obliczenia wartości miar. Wszystkie pliki w folderze są traktowane jako pliki z próbami, dlatego nie można umieszczać w nim innych plików.

Dane wyjściowe zapisywane są w postaci pliku csv zgodnego ze strukturami DataFrame biblioteki pandas. Dodatkowo generowany jest plik "results_corr.csv" zawierający tablice z wartościami współczynników korelacji między wartościami miar. Pliki jako separatora używają znaku średnika ';'. Przykładowy plik wyjściowy został przedstawiony na listingu 4.3.

Listing 4.3 Przykład pliku wyjściowego z obliczonymi miarami num_cc i largest_cc

```
1 ;time_ms;opt_count;oracle_count;node_count;edge_count;num_cc;largest_cc
2 0;0;1;140;1;0;1;1
3 1;0;15;1204;2;1;2;1
4 2;1520;1010200;85157744;2;4;1;2
```

4.3 Generatory instancji testowych

Generatory instancji testowych napisano w języku Python. Są to trzy proste skrypty, każdy generujący inny rodzaj instancji problemu. Skrypt uniform.py generuje instancję o miastach ułożonych równomiernie. Uruchomienie skryptu wygląda następująco:

```
1 python uniform.py <N> <area_size> [-o OUTPUT]
```

Parametr N określa liczbę miast w instancji. Parametr area_size określa rozmiar boku kwadratowej planszy, na której zostaną umieszczone miasta. Opcjonalny parametr output pozwala na ustawienie nazwy folderu wyjściowego (domyślnie uniform_output).

Skrypt grid.py generuje instancję o miastach ułożonych na siatce. Uruchomienie skryptu wygląda następująco:

```
1 python grid.py <N> <gap> [-o OUTPUT]
```

Parametr N określa liczbę miast w instancji. Parametr gap określa odległość pomiędzy sąsiadującymi miastami w siatce. Opcjonalny parametr output pozwala na ustawienie nazwy folderu wyjściowego (domyślnie grid_output).

Skrypt cliques.py generuje instancję o miastach ułożonych w klastrach (klikach). Uruchomienie skryptu wygląda następująco:

```
1 python cliques.py <N> <N_cliques> [-minld MINLD] [-maxld MAXLD] \
2 [-mincd MINCD] [-maxcd MAXCD] [-o OUTPUT]
```

Parametr N określa liczbę miast w instancji. Parametr N_cliques określa liczbę klastrów. W odróżnieniu od pozostałych generatorów generator cliques.py może przyjąć dużą liczbę parametrów opcjonalnych:

- minld - minimalna odległość między miastami w tym samym klastrze (domyślnie 5)
- maxld - maksymalna odległość między miastami w tym samym klastrze (domyślnie 30)
- mincd - minimalna odległość między środkami klastrów (domyślnie 100)
- maxcd - maksymalna odległość między środkami klastrów (domyślnie 300)
- output - nazwa folderu wyjściowego (domyślnie cliques_output)

Możliwe jest że programowi nie uda się wygenerować instancji o zadanych parametrach. W takim przypadku w konsoli wyświetlany jest stosowny komunikat.

4.3.1 Pliki wyjściowe

Generatory zapisują trzy pliki do foldery wyjściowej:

- points.txt - zawiera listę miast w formacie: identyfikator, koordynat x, koordynat y
- matrix.txt - macierz odległości w formacie obsługiwany przez program próbujący
- vis.png - wizualizacja instancji

4.4 Skrypty pomocnicze

Skrypty pomocnicze wykonano w języku Python. Należą do nich:

- plotting.py - Skrypt rysujący wykresy poszczególnych miar w zależności od liczby wierzchołków. Skrypt przyjmuje na wejście pliki z wartościami miar wygenerowane przez program stat_calculator
- convert_tsplib.py - Skrypt konwertujący pliki w formacie tsplib na uproszczony format przyjmowany przez program próbujący. Korzysta z biblioteki tsplib95.
- visualize.py - Prosty skrypt tworzący wizualizację grafu LON na podstawie pliku wejściowego json z próbami.

- run_exhaustive.py - Skrypt uruchamiający po kolej przegląd zupełny dla podanej listy instancji
- run_tp.py - Skrypt uruchamiający po kolej próbkowanie dwufazowe dla podanej listy instancji
- run_snowball.py - Skrypt uruchamiający po kolej próbkowanie *snowball* dla podanej listy instancji
- run_stat_calc.py - Skrypt uruchamiający program obliczający wartości miar dla podanej listy folderów w próbkami
- run_plotting.py - Skrypt uruchamiający po kolej skrypt plotting.py dla podanej listy plików z obliczonymi wartościami miar

4.5 Wykorzystane biblioteki

Najważniejsze zewnętrzne biblioteki wykorzystane w projekcie:

Rust:

- rand_chacha - szybki i deterministyczny generator liczb losowych,
- clap - parser argumentów konsoli,
- rustc-hash - szybka hashmapa,
- permutations_iter - generator permutacji metodą Steinhausa-Johnsona-Trottera,
- PyO3 - narzędzie do tworzenia natywnych modułów Pythona w języku Rust,
- tsptools - generator losowego rozwiązania problemu TSP oraz funkcja obliczająca długość ścieżki.

Python:

- igraph - obliczanie wartości miar grafu LON, generowanie wizualizacji,
- matplotlib - rysowanie wykresów,
- numpy, pandas - przetwarzanie danych,
- Pillow - generowanie obrazów,
- tsplib95 - odczytywanie plików w formacie tsplib.

Rozdział 5

Podsumowanie

Literatura

- [1] W. BOZEJKO, A. GNATOWSKI, T. NIZYNSKI, M. AFFENZELLER, AND A. BEHAM, *Local optima networks in solving algorithm selection problem for TSP*, in Contemporary Complex Systems and Their Dependability - Proceedings of the 13th International Conference on Dependability and Complex Systems DepCoS-RELCOMEX, July 2-6, 2018, Brunów, Poland, W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak, and J. Kacprzyk, eds., vol. 761 of Advances in Intelligent Systems and Computing, Springer, 2018, pp. 83–93.
- [2] C. W. CLEGHORN AND G. OCHOA, *Understanding parameter spaces using local optima networks: a case study on particle swarm optimization*, in GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021, K. Krawiec, ed., ACM, 2021, pp. 1657–1664.
- [3] I. FRAGATA, A. BLANCKAERT, M. A. DIAS LOURO, D. A. LIBERLES, AND C. BANK, *Evolution in the light of fitness landscape theory*, Trends in Ecology & Evolution, 34 (2019), pp. 69–82.
- [4] D. IC LANZAN, F. DAOLIO, AND M. TOMASSINI, *Data-driven local optima network characterization of qaplib instances*, in Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO '14, New York, NY, USA, 2014, Association for Computing Machinery, p. 453–460.
- [5] P. MC MENEMY, N. VEERAPEN, AND G. OCHOA, *How perturbation strength shapes the global structure of TSP fitness landscapes*, in Evolutionary Computation in Combinatorial Optimization - 18th European Conference, EvoCOP 2018, Parma, Italy, April 4-6, 2018, Proceedings, A. Liefooghe and M. López-Ibáñez, eds., vol. 10782 of Lecture Notes in Computer Science, Springer, 2018, pp. 34–49.
- [6] G. OCHOA AND S. HERRMANN, *Perturbation Strength and the Global Structure of QAP Fitness Landscapes: 15th International Conference, Coimbra, Portugal, September 8–12, 2018, Proceedings, Part II*, 01 2018, pp. 245–256.
- [7] G. OCHOA AND N. VEERAPEN, *Mapping the global structure of TSP fitness landscapes*, J. Heuristics, 24 (2018), pp. 265–294.
- [8] G. OCHOA, S. VÉREL, F. DAOLIO, AND M. TOMASSINI, *Local optima networks: A new model of combinatorial fitness landscapes*, CoRR, abs/1402.2959 (2014).
- [9] G. OCHOA, S. VÉREL, AND M. TOMASSINI, *First-improvement vs. best-improvement local optima networks of NK landscapes*, CoRR, abs/1207.4455 (2012).
- [10] M. C. TEIXEIRA AND G. L. PAPPA, *Understanding automl search spaces with local optima networks*, in GECCO '22: Genetic and Evolutionary Computation Conference,

Boston, Massachusetts, USA, July 9 - 13, 2022, J. E. Fieldsend and M. Wagner, eds., ACM, 2022, pp. 449–457.

- [11] S. L. THOMSON, G. OCHOA, AND S. VÉREL, *Clarifying the difference in local optima network sampling algorithms*, in Evolutionary Computation in Combinatorial Optimization - 19th European Conference, EvoCOP 2019, Held as Part of EvoStar 2019, Leipzig, Germany, April 24-26, 2019, Proceedings, A. Liefooghe and L. Paquete, eds., vol. 11452 of Lecture Notes in Computer Science, Springer, 2019, pp. 163–178.
- [12] S. L. THOMSON, G. OCHOA, S. VÉREL, AND N. VEERAPEN, *Inferring future landscapes: Sampling the local optima level*, Evol. Comput., 28 (2020), pp. 621–641.
- [13] M. TOMASSINI, S. VÉREL, AND G. OCHOA, *Complex-network analysis of combinatorial spaces: The nk landscape case*, Phys. Rev. E, 78 (2008), p. 066114.
- [14] S. VÉREL, F. DAOLIO, G. OCHOA, AND M. TOMASSINI, *Sampling local optima networks of large combinatorial search spaces: The QAP case*, in Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part II, A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and L. D. Whitley, eds., vol. 11102 of Lecture Notes in Computer Science, Springer, 2018, pp. 257–268.