Project Gegevensstructuren & Algoritmen
Hamiltonian Completion Problem

Academiejaar 2023 – 2024

Seppe Degryse

Inleiding

In dit verslag wordt een analyse gemaakt van een algoritme voor het oplossen van het Hamiltonian Completion Problem (HCP). Het verslag begint met een beschrijving van het algoritme. Vervolgens wordt toegelicht welke datastructuren zijn gebruikt om een graaf voor te stellen en welke nodig waren om het probleem verder op te lossen. Daarnaast zijn er enkele tests uitgevoerd om enkele ontwerpbeslissingen te ondersteunen. Tot slot wordt een kort besluit gevormd en wordt een kleine vergelijking gemaakt met de algoritmen waarop dit algoritme is gebaseerd. In de bijlage is een tabel opgenomen met de resultaten voor elke grafiek in de dataset.

Inhoudsopgave

1 Algoritme

Het algoritme volgt over het algemeen het stappenplan zoals beschreven in het artikel *"A multi-start local search algorithm for the Hamiltonian completion problem on undirected graphs"* [Jooken et al., 2020]. De implementatie is volledig zelf ontwikkeld, waarbij bepaalde stappen, zoals rotation moves, zijn weggelaten omdat uit testresultaten bleek dat ze weinig toegevoegde waarde hadden.

1.1 Initial spanning tree

De initiële stap van het algoritme omvat de constructie van een opspannende boom, waarbij verschillende methoden mogelijk zijn. In dit geval is de keuze gemaakt voor een diepte-eerst benadering. Deze keuze is gemotiveerd door het streven naar een opspannende boom met een grotere hoogte in vergelijking met een breedte-eerst methode. Dit is gunstig, aangezien het algoritme tracht langere paden te construeren. Het idee hierachter is dat door te starten met langere paden, het algoritme zich al dichter bij een oplossing bevindt.

Een tweede essentieel aspect in deze fase betreft de willekeurige selectie van de wortel van de boom. Deze benadering is bewust gekozen vanwege de herhaalde uitvoering van deze stap, waardoor wordt voorkomen dat elke keer dezelfde opspannende boom wordt gegenereerd. Hierdoor levert het algoritme consistente resultaten op, ongeacht de volgorde waarin knooppunten aan de graaf zijn toegevoegd. Hoewel het waar is dat bij het oplossen van het HCP voor een boom het algoritme mogelijk een voordeel zou hebben als de wortel als eerste wordt toegevoegd en vervolgens als wortel wordt gekozen voor het construeren van de opspannende boom, is het belangrijk op te merken dat het algoritme is ontworpen om consistent te werken voor elk type graaf.

1.2 Path cover

Vervolgens wordt uit de opspannende boom de path cover geconstrueerd, wat efficiënt kan gebeuren in lineaire tijd, zoals beschreven in het artikel *"Optimal Hamiltonian completions and path covers for trees, and a reduction to maximum flow"* [Franzblau and Raychaudhuri, 2002].

Bij het berekenen van de path cover wordt elke knoop met een graad van drie of meer als volgt behandeld: verwijder elke boog naar een buur, behalve de eerste twee. Op deze manier blijft een correcte path cover over.

Het is essentieel dat de boom in de juiste volgorde wordt doorlopen. Hoewel het genoemde artikel een achterwaartse breedte-eerst methode voorstelt, bleek deze moeilijk te implementeren. De voorwaarde vereist namelijk dat voor elke knoop eerst de kinderen moeten worden behandeld voordat de knoop zelf wordt behandeld, wat ook kan worden bereikt met een diepte-eerst methode.

Aanvankelijk werd de diepte-eerst methode geïmplementeerd met recursie, waarbij de recursieve oproepen voor de initiële oproep werden behandeld, startende vanuit de wortel. Echter, voor grotere grafen stuitte het algoritme op een `StackOverflowError` vanwege het grote aantal recursieve oproepen. Een mogelijke oplossing was het vergroten van de stack size van de JVM, maar dit wordt als "bad practice"beschouwd. De alternatieve oplossing was om te werken met een stack, wat weliswaar extra geheugenallocatie met zich meebrengt. Hierbij wordt een stack gebruikt om de boom te doorlopen, een set om gecontroleerde knopen bij te houden, en een extra stack om uiteindelijk de gewenste volgorde op te slaan. Deze aanpassing stelt het algoritme echter in staat om een path cover te berekenen voor een boom van willekeurige grootte.

1.3 Connect paths

De derde stap van het algoritme is de eerste fase die daadwerkelijk op zoek gaat naar de oplossing. In deze stap wordt ervan uitgegaan dat de boom bestaat uit zijn path cover.

De eerste handeling in deze stap is het opnieuw genereren van de union-find datastructuur. Tijdens de constructie van de path cover zijn bogen uit de boom verwijderd, maar aangezien deze datastructuur daar geen ondersteuning voor biedt, moet deze opnieuw worden gegenereerd. Hoewel dit ongeveer 57% van de uitvoeringstijd van deze stap in beslag neemt, is het noodzakelijk om te voorkomen dat er lussen worden gecreëerd.

Daarna wordt de lijst met knopen gepermuteerd. Dit is belangrijk om het deterministische gedrag van het algoritme te doorbreken en de algemene prestaties te verbeteren (zie sectie 3.2). Deze aanpak stelt het algoritme ook in staat om naar elke mogelijke opspannende boom over te gaan.

Ten slotte wordt de feitelijke stap uitgevoerd. Het algoritme doorloopt elke knoop en voegt, als de knoop een graad van één of lager heeft, de eerste boog toe naar een andere knoop met graad één die kan worden toegevoegd. Op deze manier kan mogelijk een kleinere path cover worden verkregen.

1.4 Restoring the tree

De laatste stap is het terug omzetten van onze samengevoegde path cover naar een boom, waardoor het proces opnieuw kan worden gestart. In eerste instantie werd ook een permutatie toegepast op de verzameling knopen, maar dit is later achterwege gelaten omdat de testen aantoonden dat dit overbodig was (zie sectie 3.2).

Om opnieuw een boom te verkrijgen, blijft het algoritme bogen toevoegen die geen lussen creëren, totdat de boom $n-1$ bogen bevat. Een graaf met $n-1$ bogen die geen lus bevat, is per definitie een boom.

1.5 Simulated annealing

Er bestaan twee varianten van het algoritme: een local search variant en een metaheuristic search variant. In de local search variant blijft het algoritme zoeken totdat het resultaat verslechtert, waarna het wordt gestopt en opnieuw wordt gestart met een nieuwe opspannende boom. In het geval van de metaheuristic search kan, bij een slechter resultaat, met een bepaalde kans worden besloten om toch verder te rekenen.

De evaluatiefunctie bestaat uit een combinatie van drie factoren. De eerste factor is de dichtheid van de path cover, berekend als het aantal bogen in de boom na het construeren van de path cover gedeeld door het aantal bogen in de originele graaf. De tweede factor is het path partitie nummer, dat aangeeft uit hoeveel paden de path cover bestaat. De laatste factor is de isolatie van de knopen, wat aangeeft hoeveel knopen er zijn met graad nul. De uiteindelijke evaluatie wordt als volgt geformuleerd:

$$E = density - ppn - isolation$$

De move-stap begint met het construeren van de path cover. Vervolgens wordt de connect paths operatie toegepast, gevolgd door het opnieuw construeren van de boom. Dit proces wordt herhaald totdat de temperatuur lager is dan de minimumtemperatuur. De temperatuur wordt alleen verlaagd als de keuze wordt gemaakt om door te gaan met een toestand die een lagere evaluatiewaarde heeft.

1.6 Volledig algoritme

Nu alle stappen van het algoritme beschreven zijn, kan een overzicht van het volledige algoritme worden geschetst. Allereerst wordt bepaald hoe vaak het algoritme opnieuw mag starten. Deze waarde is het minimum van het aantal knopen en de gespecificeerde *maxIterations*. Dit is essentieel, aangezien het aantal herstarts beperkt moet zijn tot het aantal knopen om te voorkomen dat het algoritme een opspannende boom maakt die reeds eerder is doorzocht.

Nadat de opspannende boom is geconstrueerd, wordt het simulated annealing-proces toegepast op de boom. Als dit proces een padpartitienummer van één oplevert, kan het zoekproces worden gestopt, omdat dit overeenkomt met het globale minimum.

2 Datastructuren

2.1 Adjacency list

Voor de representatie van de graaf is gekozen voor een adjacency list. Deze representatie sluit goed aan bij de benodigde operaties, zoals `getVertices()` en `getNeighborsOf()`. De implementatie maakt gebruik van een hashmap, waarbij de knoop fungeert als sleutel en de bijbehorende buurknopen als waarden in een set zijn opgeslagen. Hierdoor kan in O(n) tijd de lijst met buren worden opgevraagd, wat een veelvoorkomende operatie is.

Er is ook een aanvullende lijst gecreëerd die alle knopen bevat. Dit was noodzakelijk om de knopen te kunnen permuteren, aangezien deze operatie niet wordt ondersteund op een set. Desondanks is er nog steeds een methode, `getVerticesSet()`, beschikbaar om in constante tijd een set van alle knopen op te vragen en te controleren of een graaf een specifieke knoop bevat.

2.2 Disjoint Set

Om efficiënt te kunnen controleren of een boog een lus vormt, wordt gebruikgemaakt van een disjoint set. Deze disjoint set bestaat uit twee hashmaps: de parent-hashmap en de size-hashmap. Beide hashmaps gebruiken de knoop als sleutel. De parent-hashmap heeft als waarde de ouderknoop, terwijl de size-hashmap de grootte van de set als waarde heeft.

De implementatie is gebaseerd op de pseudocode van Wikipedia [Wikipedia contributors, 2023]. Om te controleren of een boog geen lus vormt, hoeft alleen te worden gecontroleerd of beide knopen niet dezelfde ouder hebben. Dit kan worden gedaan met de methode `find()`.

In de `find()`-methode wordt gebruikgemaakt van padcompressie. Dit zorgt ervoor dat elke knoop tussen de queryknoop en de ouderknoop wijst naar de ouderknoop, waardoor volgende oproepen versnellen.

2.3 Counters

Om het aantal knopen in de graaf te verkrijgen, wordt eenvoudigweg de grootte van de lijst met knopen opgevraagd, wat in constante tijd kan worden gedaan.

Voor het opvragen van het aantal bogen in de graaf werd aanvankelijk de waarden van de adjacency list overlopen, wat een hashset met de buren is, en werd de grootte van die sets opgeteld. Dit proces gebeurde in lineaire tijd, wat suboptimaal is. Aangezien deze functie in elke iteratie van het restore tree-proces wordt opgeroepen, is er een aparte teller toegevoegd. Deze teller wordt verhoogd wanneer een boog wordt toegevoegd en verlaagd wanneer een boog wordt verwijderd. Hierdoor kan het aantal bogen in de graaf nu in constante tijd worden opgevraagd.

3 Resultaten en experimenten

De experimenten werden uitgevoerd op een dataset bestaande uit 264 grafen, die uitsluitend samenhangende ongerichte grafen bevatte. Deze set is opgedeeld in twee delen: een testset bestaande uit 160 grafen en een tuningset bestaande uit 104 grafen. Voor elke test werden de uitvoeringstijd en de oplossing opgeslagen in een CSV-bestand. Alle tests werden uitgevoerd op een Apple MacBook Pro 13 met een Apple M2-chip.

Bij elke test werden verschillende waarden geanalyseerd, waaronder:

1. **Gemiddelde uitvoeringstijd**:
   Dit omvat de gemiddelde tijd die nodig is voor de uitvoering van zowel het local search- als het metaheuristic search-algoritme.

2. **Gemiddeld aantal bogen afwijking van het globale optimum**:
   Deze meting geeft aan in hoeverre de oplossingen van de algoritmes afwijken van het globale optimum. Het gemiddelde aantal bogen dat het algoritme mist of te veel heeft ten opzichte van het optimale aantal wordt geëvalueerd.

3. **Significantie tussen het gemiddelde van local search en metaheuristic search**:
   Hierbij wordt gekeken naar de statistische significantie tussen het gemiddelde van de resultaten verkregen door het local search-algoritme en het metaheuristic search-algoritme. Dit wordt geëvalueerd met behulp van een t-test voor gepaarde groepen.

4. **Het aantal keren dat het algoritme het optimale resultaat bereikt**:
   Dit geeft inzicht in de robuustheid en effectiviteit van elk algoritme. Het aantal keren dat elk algoritme het optimale resultaat bereikt, wordt vastgesteld.

Deze waarden bieden een omvattend beeld van de prestaties van de algoritmen, met aandacht voor zowel efficiëntie als nauwkeurigheid. Het gebruik van statistische tests voegt een objectief element toe aan de vergelijking tussen local search en metaheuristic search, terwijl het aantal keren dat het optimale resultaat wordt bereikt, inzicht geeft in de betrouwbaarheid van elk algoritme.

3.1 Meerdere startpunten

De eerste onderzoeksvraag die wordt gesteld, betreft het ideale aantal initiële opspannende bomen. Het spreekt voor zich dat hoe groter het aantal is, des te meer van de zoekruimte het algoritme zal hebben verkend, en dus zou het een betere oplossing moeten kunnen vinden. Echter, dit gaat gepaard met aanzienlijke tijdsinvesteringen. Daarom werden tests uitgevoerd met verschillende waarden voor n, variërend van 1 tot 40, met stappen van 10. Deze tests werden uitgevoerd op de volledige dataset, inclusief zowel de test- als de tuningset. Deze testen zijn uitgevoerd op het algoritme met twee knooppermutaties, $T_{max} = 100$, $T_{min} = 0,1$ en $\alpha = 0.93$.

In Tabel 1a zijn de gemiddelde afwijkingen voor elke n weergegeven. Er is een duidelijke sprong van 1 naar 10, zowel in het local search-algoritme als in het metaheuristic search-algoritme. Dit komt doordat de kans dat het algoritme moet starten vanuit een worst-case scenario, namelijk een suboptimale opspannende boom, zeer klein is geworden. Het local search-algoritme krijgt nu in plaats van 1 maar liefst 10 kansen om een oplossing te vinden. Voor de grotere waarden van n zien we steeds een marginale daling van de gemiddelde afwijking, terwijl de uitvoeringstijd steeds verdubbelt (Tabel 1b). Het aantal keren dat het optimale resultaat wordt bereikt, volgt dezelfde trend als de gemiddelde afwijking (Tabel 1c). Opvallend is dat voor $n = 40$ minder vaak het optimale resultaat wordt bereikt dan voor $n = 30$. Dit lijkt op het eerste gezicht vreemd, maar het onthult meteen een van de nadelen van dit algoritme. Omdat verschillende stappen willekeurig worden uitgevoerd, kan niet worden gegarandeerd dat het algoritme hetzelfde resultaat zal behalen. Er bestaat ook de kans dat het algoritme één of meerdere keren vanuit dezelfde opspannende boom is gestart.

Er is besloten om de waarde $n = 30$ te kiezen, omdat we van 20 naar 30 nog steeds dezelfde significante sprong zien in de gemiddelde afwijking, vergelijkbaar met die van 10 naar 20 voor het metaheuristic algoritme. Dit patroon is niet langer zichtbaar van 30 naar 40. Bovendien was er nog wat ruimte beschikbaar binnen de aanvaardbare uitvoeringstijd, waardoor het algoritme nog steeds redelijk snel kon blijven werken.

Merk op dat de uitvoeringstijd van $n = 30$ naar $n = 40$ slechts met $1,5$ seconden stijgt, wat niet de voorgaande trend volgt. Dit kan mogelijk te wijten zijn aan het feit dat er slechts een bepaald aantal goede initiële opspannende bomen bestaat voor elke graaf. Hierdoor zal het algoritme bij de slechtere opspannende bomen mogelijk vroegtijdig stoppen omdat het betere resultaten kan behalen.

n	Local search	Metaheurstic search
1	136,3	90,1
10	109,0	81,9
20	102,3	79,9
30	102,8	77,9
40	99, 6	77,3

(a) Gemiddelde afwijking (bogen)

n	Local search	Metaheurstic search
1	0,04	0,59
10	0,26	5,16
20	0,54	11,05
30	0,92	20,18
40	1,02	21,67

(b) Gemiddelde tijd (seconden)

n	Local search	Metaheurstic search
1	16	28
10	22	36
20	26	42
30	31	43
40	29	44

(c) Aantal keer dat het globale optima wordt bereikt

Tabel 1: Resultaten voor de testen met meerdere startpunten

3.2 Knooppermutaties

Om het deterministische gedrag van het algoritme te doorbreken en om vanuit een bepaalde opspannende boom naar eender welke opspannende boom te kunnen gaan, zijn knooppermutaties geïntroduceerd. Veel stappen van het algoritme itereren over de verzameling knopen, maar als deze steeds dezelfde volgorde hebben, zal het algoritme snel vastlopen omdat het steeds hetzelfde resultaat zal behalen.

In deze test wordt vergeleken tussen drie scenario's: het algoritme zonder knooppermutaties, het algoritme met een eenmalige permutatie voor de move-stap, en het algoritme met een permutatie van de knopen voor zowel de connect paths-stap als de restore tree-stap. Ook deze test is uitgevoerd op de volledige testset waarin het algoritme $n = 30$, $T_{max} = 100$, $T_{min} = 0,1$ en $\alpha = 0.93$ als waarden heeft.

In Tabel 2a is duidelijk te zien dat de knooppermutaties hun werk doen; er wordt een mooie sprong gemaakt van ongeveer 20% betere afwijking van het globale optimum. Ook blijkt dat de tweede knooppermutatie overbodig is, omdat het verschil tussen één en twee permutaties niet significant is. De aandachtige lezer zal opmerken in Tabel 2c dat het local search-algoritme minder vaak het globale optimum bereikt. Dit komt door de cirkelvormige grafen, waarbij de manier waarop deze worden geconstrueerd en opgeslagen zeer gunstig is voor het algoritme indien de knopen in volgorde worden overlopen. Het algoritme krijgt hier dus een voordeel ten opzichte van andere grafen. Dit is ook de reden waarom de gemiddelde afwijking stijgt. Bij het metaheuristic-algoritme zien we dat het aantal permutaties niet veel verandert aan het aantal keren dat het globale optimum wordt bereikt. Zonder knooppermutaties is er ook geen significant verschil tussen het local search-algoritme en het metaheuristic search-algoritme, zoals te zien is in Tabel 2d.

# perm	Local search	Metaheurstic search
0	94,2	94,5
1	99,7	77,0
2	102,8	77,9

(a) Gemiddelde afwijking (bogen)

# perm	Local search	Metaheurstic search
0	0,45	8,18
1	0,74	15,92
2	0,92	20,18

(b) Gemiddelde tijd (seconden)

# perm	Local search	Metaheurstic search
0	40	43
1	28	42
2	31	43

(c) Aantal keer dat het globale optima wordt bereikt

# perm	Significantie
0	$2,2 \times 10^{-1}$
1	$7,8 \times 10^{-12}$
2	$4,0 \times 10^{-8}$

(d) Significantie tussen beide algoritmen (LS en MHS)

Tabel 2: Resultaten voor de testen met knooppermutaties

3.3 Evaluatiefunctie

Om de metaheuristic search goed te laten verlopen, moet er een goede evaluatiefunctie zijn. Daarom rijst de vraag naar de invloed van het gebruik van een evaluatiefunctie in vergelijking met het gebruik van het pad partitie nummer als evaluatie. De besproken evaluatiefunctie is te vinden in sectie 1.5. Voor deze testen werd een oudere variant van het algoritme gebruikt die nog niet verder geoptimaliseerd was, waardoor de uitvoeringstijd zal verschillen met andere testen. De andere parameters van het algoritme waren: $n = 30$, $T_{\max} = 100$, $T_{\min} = 0,1$ en $\alpha = 0,93$. Het algoritme maakte ook gebruik van dubbele knooppermutaties. Deze testen zijn uitgevoerd op de volledige dataset.

In Tabel 3a is duidelijk te zien dat de evaluatiefunctie het metaheuristic search-algoritme verder laat zoeken dan voorheen. Dit wordt ook weerspiegeld in Tabel 3c, waar het metaheuristic search-algoritme vaker het globale optimum bereikt. Over het algemeen zal dit algoritme beter presteren met de evaluatiefunctie. Dit gaat echter ten koste van een verdubbeling van de uitvoeringstijd, omdat het algoritme nu langer kan zoeken. De significantie tussen beide algoritmen neemt af, maar blijft nog steeds zeer significant. Dit is waarschijnlijk toe te schrijven aan de willekeur van het algoritme, wat verschillende spreidingen genereert.

Evaluation	Local search	Metaheurstic search
PPN	100,2	88,0
E	100,3	77,9

(a) Gemiddelde afwijking (bogen)

Evalutation	Local search	Metaheurstic search
PPN	12,1	13,4
E	13,1	25,5

(b) Gemiddelde tijd (seconden)

Evaluation	Local search	Metaheurstic search
PPN	28	36
E	29	43

(c) Aantal keer dat het globale optima wordt bereikt

Evaluation	Significantie
PPN	$2,3 \times 10^{-13}$
E	$4,6 \times 10^{-11}$

(d) Significantie tussen beide algoritmen (LS en MHS)

Tabel 3: Resultaten voor de testen met en zonder evaluatiefunctie

4 Besluit

Na een kleine 20 uur aan testen is dit een optimale variant van het algoritme geworden. Mogelijk kunnen er nog enkele optimalisaties worden doorgevoerd, zoals het gebruik van een geavanceerdere evaluatiefunctie, ondersteuning voor niet-samenhangende grafen en de implementatie van parallellisme in het algoritme.

In vergelijking met de algoritmen uit het artikel [Jooken et al., 2020] lijkt dit algoritme op het eerste gezicht beter, vooral op de testset. Echter, dit kan mogelijk worden beïnvloed door de ongeveer 20 grafen die niet in die dataset zaten en die de gemiddelde afwijking omhoog duwen. Ook bereikt dit algoritme veel minder vaak het globale optimum. Het is belangrijk om in gedachten te houden dat prestatiebeoordeling afhankelijk is van verschillende factoren en criteria, en daarom is een uitgebreide vergelijking noodzakelijk voor een volledig begrip van de algoritmen.

Het gebruik van Generatieve Artificiële Intelligentie

Bij het opstellen van deze tekst is gebruikgemaakt van generatieve artificiële intelligentie. In eerste instantie werd een tekst of paragraaf geschreven, waarna artificiële intelligentie werd ingezet om de tekst te herschrijven en eventuele fouten te corrigeren. Dit proces is toegepast om het niveau, de leesbaarheid en de duidelijkheid van de tekst te verbeteren.

Het is belangrijk om op te merken dat bij het schrijven van de code en het ontwerpen van het algoritme geen gebruik is gemaakt van generatieve artificiële intelligentie.

Referenties

[Franzblau and Raychaudhuri, 2002] Franzblau, D. S. and Raychaudhuri, A. (2002). Optimal hamiltonian completions and path covers for trees, and a reduction to maximum flow. *The ANZIAM Journal*, 44(2):193–204.

[Jooken et al., 2020] Jooken, J., Leyman, P., and De Causmaecker, P. (2020). A multi-start local search algorithm for the hamiltonian completion problem on undirected graphs. *Journal of Heuristics*, 26(5):743–769.

[Wikipedia contributors, 2023] Wikipedia contributors (2023). Disjoint-set data structure — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Disjoint-set_data_structure&oldid=1184811860`. [Online; accessed 21-November-2023].

Graph	Global optima	Local search	metaheuristic search	LS duration	MHS duration	LS deviation	MHS deviation
adjnoun.in	7	19	17	155	803	12	10
celegans_metabolic.in	30	82	71	166	3831	52	41
celegansneural.in	15	36	29	80	3082	21	14
chesapeake.in	0	2	3	2	114	2	3
circle_like_100_2.in	0	0	0	2	24	0	0
circle_like_100_3.in	0	1	0	7	1	1	0
circle_like_100_5.in	0	0	0	6	0	0	0
circle_like_100_7.in	0	0	0	3	1	0	0
circle_like_100_10.in	0	0	0	0	0	0	0
circle_like_200_2.in	0	1	2	17	472	1	2
circle_like_200_3.in	0	6	0	19	105	6	0
circle_like_200_5.in	0	1	0	20	80	1	0
circle_like_200_7.in	0	1	0	17	34	1	0
circle_like_200_10.in	0	1	0	25	20	1	0
circle_like_500_2.in	0	7	4	58	1332	7	4
circle_like_500_3.in	0	13	2	63	2068	13	2
circle_like_500_5.in	0	9	4	63	1903	9	4
circle_like_500_7.in	0	9	2	58	1630	9	2
circle_like_500_10.in	0	3	1	56	1553	3	1
circle_like_1000_2.in	0	15	12	156	3059	15	12
circle_like_1000_3.in	0	32	5	150	4706	32	5
circle_like_1000_5.in	0	23	10	168	4688	23	10
circle_like_1000_7.in	0	16	4	139	4236	16	4
circle_like_1000_10.in	0	12	1	148	3832	12	1
circle_like_2000_2.in	0	29	32	368	5951	29	32
circle_like_2000_3.in	0	53	12	361	10764	53	12
circle_like_2000_5.in	0	47	21	361	10582	47	21
circle_like_2000_7.in	0	36	13	303	10152	36	13
circle_like_2000_10.in	0	36	6	325	9323	36	6
circle_like_5000_2.in	0	91	87	1067	16505	91	87
circle_like_5000_3.in	0	48	27	929	29132	48	27
circle_like_5000_5.in	0	130	72	1293	30429	130	72
circle_like_5000_7.in	0	114	34	1144	30134	114	34
circle_like_5000_10.in	0	94	25	997	28343	94	25
circle_like_10000_2.in	0	182	181	2511	32604	182	181
circle_like_10000_3.in	0	91	66	2095	63379	91	66
circle_like_10000_5.in	0	249	154	2936	67525	249	154
circle_like_10000_7.in	0	222	79	2847	69571	222	79
circle_like_10000_10.in	0	204	56	2152	64036	204	56
circle_like_20000_2.in	0	391	375	6069	75387	391	375
circle_like_20000_3.in	0	187	129	4701	138170	187	129
circle_like_20000_5.in	0	455	337	7855	153931	455	337
circle_like_20000_7.in	0	428	179	7316	151859	428	179
circle_like_20000_10.in	0	426	131	5244	156701	426	131
circle_like_30000_2.in	0	590	554	13005	149164	590	554
circle_like_30000_3.in	0	332	211	8812	236645	332	211
circle_like_30000_5.in	0	744	516	16674	258339	744	516
circle_like_30000_7.in	0	659	258	13328	262769	659	258
circle_like_30000_10.in	0	607	189	9849	266975	607	189
delaunay_n10.in	0	76	63	177	5913	76	63
delaunay_n11.in	1	151	122	352	12650	150	121
delaunay_n12.in	2	333	267	926	28161	331	265
dolphins.in	5	9	8	5	270	4	3
email.in	105	265	256	263	12726	160	151
er_8_3.in	1	13	5	31	1232	12	4
er_8_4.in	0	2	0	26	190	2	0
er_8_5.in	0	0	0	12	1	0	0
er_8_6.in	0	0	0	11	1	0	0
er_8_7.in	0	0	0	2	2	0	0
er_8_8.in	0	0	0	2	2	0	0
er_9_3.in	1	30	22	82	3235	29	21
er_9_4.in	0	10	4	66	1968	10	4
er_9_5.in	0	3	0	63	338	3	0
er_9_6.in	0	1	0	77	3	1	0
er_9_7.in	0	0	0	4	4	0	0
er_9_8.in	0	0	0	7	7	0	0
er_9_9.in	0	0	0	5	15	0	0
er_10_4.in	0	22	12	153	4965	22	12
er_10_5.in	0	5	3	167	3554	5	3
er_10_6.in	0	3	0	174	413	3	0
er_10_7.in	0	0	0	10	35	0	0
er_10_8.in	0	0	0	44	21	0	0
er_10_9.in	0	0	0	21	19	0	0
er_10_10.in	0	0	0	45	31	0	0
er_11_3.in	3	148	129	413	15429	145	126
er_11_4.in	0	56	31	361	11758	56	31
er_11_5.in	0	21	9	334	8764	21	9
er_11_6.in	0	6	3	406	7065	6	3
er_11_7.in	0	3	0	561	828	3	0
er_11_8.in	0	0	0	466	46	0	0
er_11_9.in	0	0	0	44	73	0	0
er_11_10.in	0	0	0	82	99	0	0
er_11_11.in	0	0	0	125	122	0	0

er_12_4.in	0	112	82	826	27610	112	82
er_12_5.in	0	43	24	809	21120	43	24
er_12_6.in	0	19	9	1139	17283	19	9
er_12_7.in	0	8	3	1174	15289	8	3
er_12_8.in	0	2	0	2440	528	2	0
er_12_9.in	0	0	0	3883	357	0	0
er_12_10.in	0	0	0	164	260	0	0
er_12_11.in	0	0	0	294	262	0	0
er_12_12.in	0	0	0	444	402	0	0
er_13_4.in	0	254	186	2946	67479	254	186
er_13_5.in	0	107	68	2310	53221	107	68
er_13_6.in	0	45	25	3134	39062	45	25
er_13_7.in	0	20	8	2369	32378	20	8
er_13_8.in	0	8	3	3579	30831	8	3
er_13_9.in	0	1	0	6501	3350	1	0
er_13_10.in	0	0	0	3812	530	0	0
er_13_11.in	0	0	0	2741	662	0	0
grid_graph_2_3000.in	0	394	375	833	29577	394	375
grid_graph_2_5000.in	0	644	639	1457	52091	644	639
grid_graph_10_10.in	0	2	1	6	226	2	1
grid_graph_10_20.in	0	8	5	14	660	8	5
grid_graph_10_30.in	0	14	9	27	1107	14	9
grid_graph_10_40.in	0	15	11	36	1444	15	11
grid_graph_10_50.in	0	27	20	49	2069	27	20
grid_graph_10_60.in	0	30	25	62	2611	30	25
grid_graph_10_70.in	0	38	20	60	2912	38	20
grid_graph_10_80.in	0	26	12	98	2717	26	12
grid_graph_10_200.in	0	118	103	280	9032	118	103
grid_graph_10_300.in	0	189	179	287	15452	189	179
grid_graph_20_20.in	0	19	15	33	1539	19	15
grid_graph_20_30.in	0	41	26	67	2652	41	26
grid_graph_20_40.in	0	39	32	84	3255	39	32
grid_graph_20_50.in	0	62	42	117	4792	62	42
grid_graph_20_60.in	0	74	65	115	5661	74	65
grid_graph_20_70.in	0	82	37	125	6557	82	37
grid_graph_20_80.in	0	34	23	187	5975	34	23
grid_graph_30_30.in	0	50	38	114	4143	50	38
grid_graph_30_40.in	0	61	53	143	5155	61	53
grid_graph_30_50.in	0	101	68	237	7406	101	68
grid_graph_30_60.in	0	107	101	182	9070	107	101
grid_graph_30_70.in	0	135	60	215	10546	135	60
grid_graph_30_80.in	0	54	32	306	9407	54	32
grid_graph_40_40.in	0	85	73	200	7258	85	73
grid_graph_40_60.in	0	144	145	236	12424	144	145
grid_graph_40_70.in	0	176	79	318	14740	176	79
grid_graph_40_80.in	0	71	45	438	13212	71	45
grid_graph_50_50.in	0	174	117	438	13242	174	117
grid_graph_50_60.in	0	178	157	352	15868	178	157
grid_graph_50_70.in	0	232	100	396	18706	232	100
grid_graph_50_80.in	0	90	63	566	17045	90	63
grid_graph_60_60.in	0	219	209	392	19315	219	209
grid_graph_60_70.in	0	281	111	528	22587	281	111
grid_graph_60_80.in	0	110	73	711	19927	110	73
grid_graph_70_70.in	0	348	136	586	27922	348	136
grid_graph_70_80.in	0	137	85	738	24138	137	85
grid_graph_80_80.in	0	140	97	958	28014	140	97
jazz.in	3	5	4	44	2189	2	1
karate.in	10	11	12	2	126	1	2
lesmis.in	19	22	20	9	410	3	1
polbooks.in	0	7	3	10	389	7	3
preferential_attachment_100_3	4	13	10	9	429	9	6
preferential_attachment_100_4	2	12	10	10	470	10	8
preferential_attachment_100_5	0	7	5	10	411	7	5
preferential_attachment_100_6	0	6	2	10	371	6	2
preferential_attachment_100_7	0	4	2	10	375	4	2
preferential_attachment_100_8	0	3	0	9	60	3	0
preferential_attachment_100_9	0	1	0	8	27	1	0
preferential_attachment_200_3	19	42	37	22	1151	23	18
preferential_attachment_200_4	9	33	30	25	1186	24	21
preferential_attachment_200_5	0	14	12	24	1064	14	12
preferential_attachment_200_6	0	12	7	24	968	12	7
preferential_attachment_200_7	0	9	5	26	990	9	5
preferential_attachment_200_8	0	7	2	27	813	7	2
preferential_attachment_200_9	0	4	0	23	509	4	0
preferential_attachment_300_3	31	63	54	41	1878	32	23
preferential_attachment_300_4	0	44	38	40	1864	44	38
preferential_attachment_300_5	0	29	24	50	1827	29	24
preferential_attachment_300_6	0	20	12	41	1591	20	12
preferential_attachment_300_7	0	14	10	41	1584	14	10
preferential_attachment_300_8	0	12	6	41	1444	12	6
preferential_attachment_300_9	0	10	4	40	1307	10	4
preferential_attachment_1000_3	105	215	202	158	7268	110	97
preferential_attachment_1000_4	0	154	144	206	7696	154	144
preferential_attachment_1000_5	0	106	83	191	7567	106	83

preferential_attachment_1000_6	0	89	62	194	7459	89	62
preferential_attachment_1000_7	0	68	48	217	7359	68	48
preferential_attachment_1000_8	0	54	34	184	6932	54	34
preferential_attachment_1000_9	0	51	28	184	6715	51	28
preferential_attachment_2000_3	215	450	439	385	15789	235	224
preferential_attachment_2000_4	0	334	311	428	16611	334	311
preferential_attachment_2000_5	0	244	219	473	16719	244	219
preferential_attachment_2000_6	0	179	147	506	16639	179	147
preferential_attachment_2000_7	0	160	119	478	16360	160	119
preferential_attachment_2000_8	0	123	84	470	15954	123	84
preferential_attachment_2000_9	0	99	63	449	15015	99	63
preferential_attachment_3000_3	289	666	659	624	24315	377	370
preferential_attachment_3000_4	0	498	449	695	25937	498	449
preferential_attachment_3000_5	0	367	323	771	26382	367	323
preferential_attachment_3000_6	0	272	225	731	28345	272	225
preferential_attachment_3000_7	0	251	194	695	26617	251	194
preferential_attachment_3000_8	0	195	143	865	26156	195	143
preferential_attachment_3000_9	0	144	109	739	24452	144	109
star_random_leaves_connected_100.in	35	40	37	7	430	5	2
star_random_leaves_connected_200.in	63	77	72	18	995	14	9
star_random_leaves_connected_300.in	97	115	110	32	1532	18	13
star_random_leaves_connected_400.in	131	153	149	38	2155	22	18
star_random_leaves_connected_500.in	158	192	186	53	2640	34	28
star_random_leaves_connected_600.in	192	237	227	61	3213	45	35
star_random_leaves_connected_700.in	211	264	260	68	3809	53	49
star_random_leaves_connected_800.in	258	319	312	88	4492	61	54
star_random_leaves_connected_900.in	283	355	342	88	4958	72	59
star_random_leaves_connected_1000.in	326	404	400	115	5632	78	74
star_random_leaves_connected_1100.in	347	447	433	120	6134	100	86
star_random_leaves_connected_1200.in	392	488	476	141	6716	96	84
star_random_leaves_connected_1300.in	431	537	521	134	7249	106	90
star_random_leaves_connected_1400.in	456	577	560	158	7864	121	104
star_random_leaves_connected_1500.in	486	621	602	167	8395	135	116
star_random_leaves_connected_1600.in	515	634	625	191	9044	119	110
star_random_leaves_connected_1700.in	527	675	662	206	9598	148	135
star_random_leaves_connected_1800.in	572	736	707	182	9863	164	135
star_random_leaves_connected_1900.in	617	781	758	209	10654	164	141
star_random_leaves_connected_2000.in	620	792	784	251	11428	172	164
star_random_leaves_connected_2100.in	679	845	828	237	11941	166	149
star_random_leaves_connected_2200.in	709	895	871	266	12554	186	162
star_random_leaves_connected_2300.in	746	941	928	262	13028	195	182
star_random_leaves_connected_2400.in	787	1000	987	273	13616	213	200
star_random_leaves_connected_2500.in	806	1023	997	316	14392	217	191
star_random_leaves_connected_2600.in	825	1057	1038	309	14708	232	213
star_random_leaves_connected_2700.in	866	1083	1073	353	15511	217	207
star_random_leaves_connected_2800.in	894	1143	1127	329	16321	249	233
star_random_leaves_connected_2900.in	934	1177	1152	328	16643	243	218
star_random_leaves_connected_3000.in	953	1219	1200	390	17145	266	247
star_random_leaves_connected_3100.in	985	1259	1233	359	17729	274	248
star_random_leaves_connected_3200.in	1026	1308	1281	363	18509	282	255
star_random_leaves_connected_3300.in	1043	1319	1310	475	19095	276	267
star_random_leaves_connected_3400.in	1076	1375	1346	391	19463	299	270
star_random_leaves_connected_3500.in	1129	1460	1416	383	20067	331	287
star_random_leaves_connected_3600.in	1151	1471	1443	471	20862	320	292
star_random_leaves_connected_3700.in	1178	1503	1475	427	20900	325	297
star_random_leaves_connected_3800.in	1241	1570	1544	457	21954	329	303
star_random_leaves_connected_3900.in	1226	1585	1546	486	22385	359	320
star_random_leaves_connected_4000.in	1293	1643	1614	476	22814	350	321
structured_tree_2_2	3	2	2	0	2	-1	-1
structured_tree_2_3	7	6	6	0	9	-1	-1
structured_tree_2_4	13	13	13	0	26	0	0
structured_tree_2_5	21	22	22	1	56	1	1
structured_tree_2_6	31	33	33	1	75	2	2
structured_tree_2_7	43	46	46	1	102	3	3
structured_tree_2_8	57	61	61	2	128	4	4
structured_tree_2_9	73	78	78	3	163	5	5
structured_tree_2_10	91	97	97	3	199	6	6
structured_tree_2_11	111	118	118	4	242	7	7
structured_tree_2_12	133	141	141	5	294	8	8
structured_tree_2_13	157	166	166	6	341	9	9
structured_tree_3_2	5	5	5	0	13	0	0
structured_tree_3_3	20	22	22	1	75	2	2
structured_tree_3_4	51	58	57	3	170	7	6
structured_tree_3_5	104	112	112	7	320	8	8
structured_tree_3_6	185	199	198	12	539	14	13
structured_tree_3_7	300	313	313	28	852	13	13
structured_tree_3_8	455	480	480	38	1234	25	25
structured_tree_3_9	656	669	669	50	1743	13	13
structured_tree_4_2	11	12	12	1	57	1	1
structured_tree_4_3	61	67	66	4	247	6	5
structured_tree_4_4	205	222	222	15	740	17	17
structured_tree_4_5	521	541	540	36	1772	20	19
structured_tree_4_6	1111	1158	1157	117	3725	47	46
structured_tree_4_7	2101	2166	2166	377	6899	65	65

structured_tree_5_2	21	21	22	2	121	0	1
structured_tree_5_3	182	201	201	19	817	19	19
structured_tree_5_4	819	887	886	103	3380	68	67
structured_tree_5_5	2604	2735	2735	470	9806	131	131
structured_tree_6_2	43	48	48	4	263	5	5
structured_tree_6_3	547	603	603	99	2708	56	56
structured_tree_6_4	3277	3545	3544	764	14302	268	267
structured_tree_7_2	85	96	96	10	557	11	11
structured_tree_7_3	1640	1810	1811	377	8564	170	171
structured_tree_8_2	171	196	196	20	1193	25	25
structured_tree_8_3	4921	5435	5435	1534	26923	514	514