Minimum Cut problem

Luca Zaninotto – 2057074

30 sept 2022

Contents

1 The problem **2**

2 Karger and Stein **2**
2.1 Another bound . . . 3
2.2 Results . . . 5

3 Stoer and Wagner **5**
3.1 On implementation . . . 6
3.2 Results . . . 7

4 Hybrid approach **8**
4.1 How many times run the iteration . . . 8
4.2 Results . . . 10

5 Conclusions **10**
5.1 On efficiency . . . 10
5.2 On discovery time . . . 11

A Appendix **13**
A.1 On running the script . . . 13
A.2 Structure . . . 13
A.3 Collecting results . . . 15
A.4 Given results . . . 15
A.4.1 Karger and Stein . . . 15
A.4.2 Stoer and Wagner . . . 17
A.4.3 Hybrid . . . 18

1 The problem

The problem we want to solve is to find a cut of a graph such that its weight is minimal. In other words, given a graph

$$G = (V, E)$$

where

$$V = \{1, 2, 3, \ldots, n\}$$

is a set of n vertices and

$$E \subseteq \{(u, v) \mid u, v \in V\}$$

a set of $|E| = m$ edges and a function

$$w : E \to \mathbb{R}$$

a *cost* function, that returns the weight of each edge, we want to find

$$p_1, p_2 \subseteq V \mid p_1 \cap p_2 = \emptyset\,,\; p_1 \cup p_2 = V$$

such that

$$\text{cost}(p_1, p_2) = \sum_{u \in p_1, v \in p_2} w((u, v))$$

is minimized.

2 Karger and Stein

Karger and stein works by joining subsets of nodes, until only two remain. Those two are a cut of the graph. More in detail, given a Graph $G = (V, E)$ where $V = \{1, 2, 3, 4, \ldots, n\}$ is the set of vertices, the algorithm starts with a partition of V where each node appears by itself

$$\{\{1\}, \{2\}, \{3\}, \ldots \{n\}\}$$

Then, let $\mathcal{P}_a(V)$ be the set of partitions of V, a `contraction` procedure joins two subsets and produces a new partition of V.

$$\text{contraction} : \mathcal{P}_a(V) \longrightarrow \mathcal{P}_a(V)$$

This keeps happening, until the partition of V is made of only two sets p_1, p_2. At this stage the sum of the weights that between the two nodes (the cut of the graph) is returned. This works because of the way in which the `contraction` procedure selects the nodes to join in the partitions. The two nodes are selected based on a random selection based on the weight of the nodes. Nodes with an higher weight have an higher probability of being selected.

Doing such procedure, allows to find a minimum cut with probability $\frac{1}{\log(n)}$, therefore by repeating this procedure $O(\log^2(n))$ times we can find a minimum cut with error probability less than $\frac{1}{n}$.

Karger and Stein implemented as such has a complexity of

$$O(n^2 \log^3(n))$$

2.1 Another bound

Another bound for Karger and Stein is possible to achieve, if we consider that early contractions in the `contraction` procedure are much less likely to contract on the minimum cut.

We call X_i the event that no edge of the minimum cut is contracted during the i-th iteration. We can define

$$X = \sum_{i=0}^{k} X_i$$

We know fron the analysis of the algorithm that by contracting to $t = \frac{n}{\sqrt{2}} + 1$ nodes the probability of contracting the minimum cut in this process is

$$P(n) \geq \frac{1}{2}$$

wich means the probability of success at the i=th iteration is at least $\frac{1}{2}$

$$P(X_i = 1) \leq \frac{1}{2}$$

We waht to find k such that our our random variable X (the sum of the *success* of each iteration) does not go "too far away" from the mean value μ

$$\mu = E[X] = \sum_{i=0}^{k} E[X_i] = \frac{k}{2}$$

since X_is are indipendent random indicator variables.
so we want to find

$$P\left(|X-\mu|<\epsilon\mu\right)=1-P\left(|X-\mu|>\epsilon\mu\right)$$

and we know by chernoff bound that

$$P\left(|X-\mu|>\epsilon\mu\right)\leq 2e^{-\frac{\mu\epsilon^2}{3}}\quad \text{for } 0<\epsilon\leq 1$$

by replacing $\mu=\frac{k}{2}$

$$P\left(\left|X-\frac{k}{2}\right|>\epsilon\frac{k}{2}\right)\leq 2e^{-\frac{\frac{k}{2}\epsilon^2}{3}}\quad \text{for } 0<\epsilon\leq 1$$

and therefore by choosing $k=\frac{6}{\epsilon^2}\log(n)=O(\log(n))$ we have that

$$P\left(|X-\mu|>\epsilon\mu\right)\leq\frac{2}{n}=O\left(\frac{1}{n}\right)$$

and so

$$P\left(|X-\mu|<\epsilon\mu\right)=1-O\left(\frac{1}{n}\right)$$

therefore, for $k=O\left(\log(n)\right)$ the algorithm gives a correct result with high probability with a complexity of

$$O\left(n^2\log^2(n)\right)$$

2.2 Results

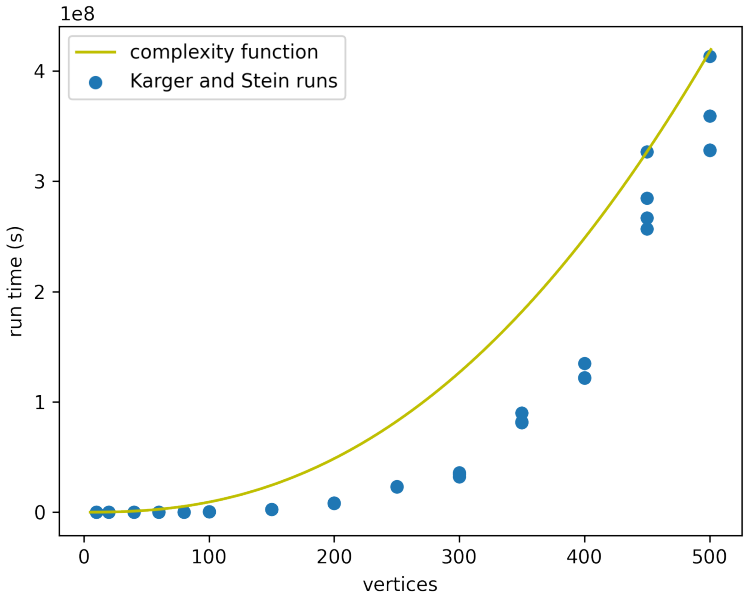

Figure 1: Karger and Stein run times against input size (number of vertices)

The function plotted is the complexity of the algorithm. The coefficient for the surface (under which all results should fall) is calculated as

$$\max\left(\frac{t}{n^2\log^2(n)}\right)$$

since the time of execution of the algorithm is maximized by this function and the different coefficient depend also on external factors (CPU throttle, memory allocations, etc.) and we're interested the behavior of the algorithm against the input nodes. As we can see in Figure 1 the run times follow the surface of the complexity function.

3 Stoer and Wagner

Stoer and Wagner works in another way. Instead of relying on random cuts relies on the fact that given a couple of nodes, They either are separated

by the minimum cut or are on the same side of the cut. So given a graph $G = (V, E)$, $s, t \in V$, an s, t minimum cut is a cut (S, T) of G s.t.

$$s \in S \vee t \in T$$

$$w(S, T) \text{ is minimum among all } s, t \text{ cuts}$$

So, let (S, T) be a global min-cut for G. For every pair $s, t \in V$ either $s \in S$ and $t \in T$ or s and t are on the same side of the cut.

Stoer and wagner leverages this fact and searches for $s, t \in V$ s.t. (S, T) is a global min cut for G. If that's also a global min-cut its weight is returned, otherwise the global min-cut for $G \backslash \{s, t\}$ (The same graph as G, where the nodes s and t are merged together) is also a global min-cut for G.

3.1 On implementation

In `stoer_wagner.py`, in the function `stMinCut`, around line 21 we can find

```
if v in q:
    keys[v] = keys[v] + graph.weight(u, v)
    q.update_elem(v, (keys[v], v))
```

`q` is an object of type `PriorityQueue` (q = PriorityQueue()). The definition for this object can be found in the file `custom_queue.py`. The reason for using a custom version of a priority queue over the default `heapq` implementation of python are essentially 2:

1. Allows to be syntactically closer to the algorithm definition we gave in class. The `PriorityQueue` object implements in fact a max-Heap structure where objects inside are tuples of type `(key, value)`, therefore in the implementation I don't have to rely on tricks such as decrementing keys instead of incrementing them to make a min-Heap (the default `heapq` in python) work like a max-Heap.

2. Allows to update elements inside. the line displayed above shows `q.update_elem(v, (keys[v], v))` which is an internal function to update an element in the queue without removing ti ad adding it again, therefore resulting in a simpler and more concise definition of the `stMinCut` function.

The `custom_queue` implementation was actually found online on GitHub[1], and was originally used by the developer to implement the Dijkstra algo-

[1] github.com/denizetkar/priority-queue

rithm. I added the function `__contains__` in `custom_queue.py` in order to be able to write `if v in q:` in `stoer_wagner.py` at line 19.

3.2 Results

The complexity of this algorithm is

$$O(mn\log(n))$$

where $n = |V|$ and $m = |E|$. so the coefficient for the plot is calculated again as

$$\max\left(\frac{t}{mn\log(n)}\right)$$

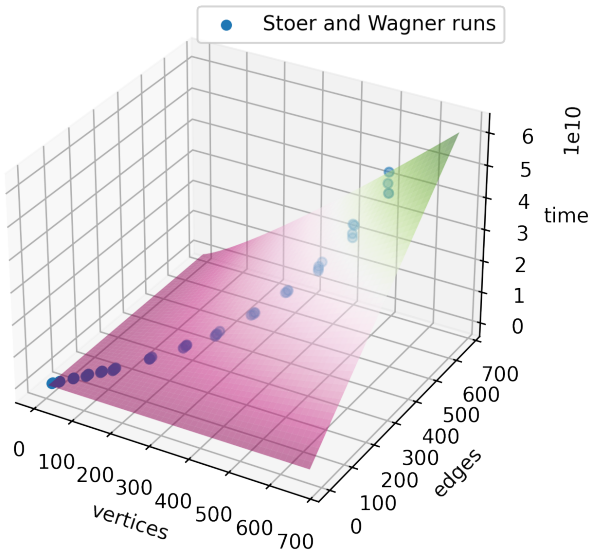

Figure 2: Stoer and Wagner run times against input size (nodes and edges)

By plotting the run times against the input sizes (nodes and vertices) we can see that the plot follows the surface of the complexity function.

4 Hybrid approach

An Hybrid approach consist into merging the two approaches (Karger and Stein and Stoer and Wagner): the algorithm contracts the graph until there are $t = \frac{n}{\sqrt{2}} + 1$ nodes, from there, Stoer and Wagner is run on the contracted graph.

To study the complexity we can take look at the hybrid procedure:

```
def hybrid(graph):
    n = graph.n_vertices
    # t = O(long(n))
    t = int(np.ceil((n / (n-1)) * np.log(n))
    amin = np.Inf
    d_time = 0
    for i in range(t):
        cut, d_time = hybrid_iteration(graph)
        if cut < amin:
            amin = cut
            d_time = perf_counter_ns()
    return amin, d_time
```

where the hybrid iteration is described as

```
def hybrid_iteration(graph):
    t = np.ceil(graph.n_vertices / np.sqrt(2) + 1)
    g = contract(graph, t)
    return stoer_wagner(g)
```

one `hybrid_iteration` consists in a contraction of the given graph, then from that contraction we run the Stoer and Wagner algorithm. Since the contraction is just a single loop running $t = \frac{n}{\sqrt{2}} + 1$ times, `contract(graph,t)` is $O(n)$. Stoer and Wagner on the resulting graph is $O(mn \log(n))$ by hypothesis.

4.1 How many times run the iteration

If we consider

$$X_i = \text{the } i\text{-th iteration has contracted a minimum cuts'edge}$$

we can have

$$X = \sum_{i=0}^{k} X_i$$

the sum of indipendent random variables that states how many times the random contraction have contracted the minimum cut of the graph. We know from hypothesis of Karger Stein algorithm, that

$$P(X_i = 1) \leq \frac{1}{2}$$

and by taking $P(X_i = 1) = \frac{1}{2}$ as assumption we can find an upper bound on the number of iterations needed in order to find w.h.p. the minimum cut. We know that, on k iterations

$$\mu = E[X] = \sum_{i=0}^{k} E[X_i] = \sum_{i=0}^{k} \frac{1}{2} = \frac{k}{2}$$

And we want to find k s.t.

$$P\left(\frac{|\beta - \alpha|}{\alpha} > \epsilon\right) \leq O\left(\frac{1}{n}\right)$$

We can use the chernoff bound

$$P\left(|X - \mu| > \mu\delta\right) \leq 2e^{-\frac{\mu\delta^2}{3}}$$

that works for $\delta \in (0, 1]$ with $\mu = \frac{k}{2}$. We conclude that, with $k = \frac{4}{\epsilon^2}\log(n)$ we can say that

$$2e^{-\frac{k\epsilon^2}{4}} = \frac{2}{n} = O\left(\frac{1}{n}\right)$$

So the same bound as for Karger and Stein applies, and the algorithm has complexity

$$O\left(mn\log^2(n)\right)$$

4.2 Results

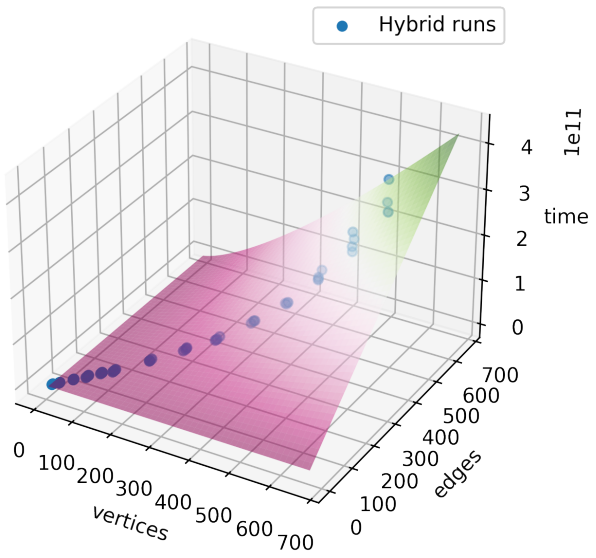

Figure 3: Hybrid run times against input size (nodes and edges)

By plotting the run times against the input size (nodes and edges) we can see that they follow the surface induced by the complexity function of the algorithm.

5 Conclusions

5.1 On efficiency

By purely relying on the complexity analysis of the algorithms we should be able to see how the Karger and stein algorithm should perform worse than Stoer and Wagner and the Hybrid approach, which have instead comparable complexities. By plotting the run time of each algorithm:

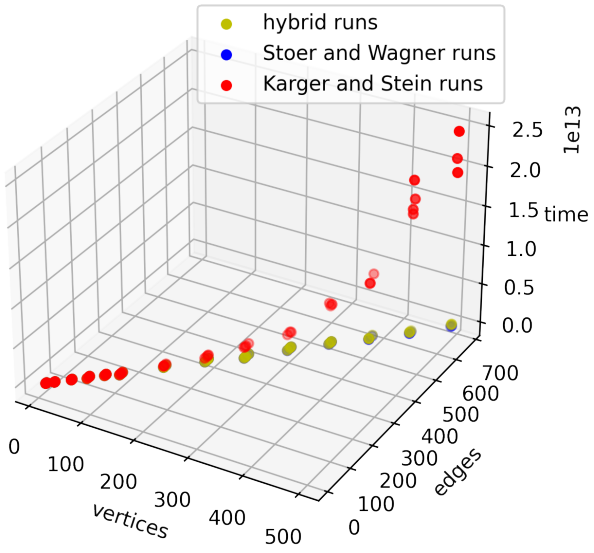

Figure 4: Runtime comparison between the three algorithms

By looking closely to the graph we can see how the results meet our expectations: Karger and Stein perform generally worse than Stoer and Wagner, that performs in a similar way to our approach.

5.2 On discovery time

Discovery time tells us another story. Even though Karger and Stein performs badly compare to the other two algorithms, the discovery time for the graphs in the dataset is not that worse compared to the discovery time of the hybrid algorithm. Stoer and Wagner outperforms the two even from this point of view, showing how is a generally faster algorithm.

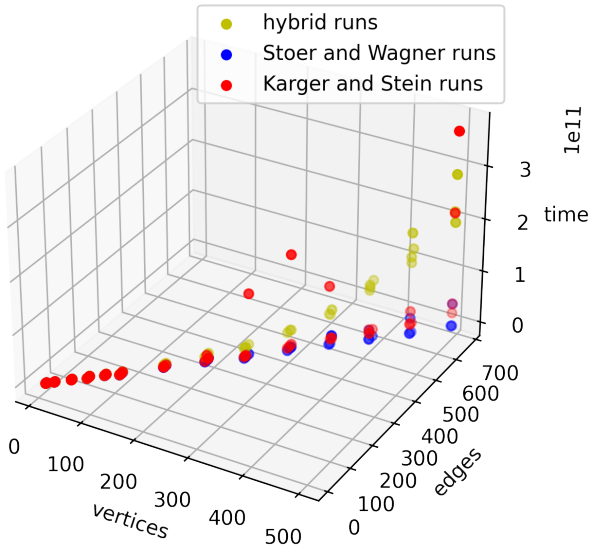

Figure 5: Discovery time comparison between the three algorithms

A Appendix

A.1 On running the script

Every command from now on is intended to be run from the `proj` folder as `PWD`.

One of the included file of the project is `requirements.txt`, it contains the dependencies for the project, and is intended to be give to pip in order to download and install them. Is also good practice to create a virtual environment to run the project, in order to avoid dependency conflicts with packages installed in the main system. To do so we can run in a shell

```
python3 -m venv venv
```

A `venv` folder will be created, with scripts to activate the environment depending on the system shell. E.g. on bash on Linux

```
. venv/bin/activate
```

or on windows shell

```
venv\Scripts\activate.bat
```

or Power-Shell

```
venv\Scripts\Activate.ps1
```

With the environment active is possible to install all the required libraries (listed in `requirements.txt`)

```
pip install -r requirements.txt
```

From this point on a variety of scripts are available for the project. The main script is in the `main.py` file. can be run with

```
python src/main.py --help
```

to show all the available options. Similarly `plot_d_times.py` and `plot_run_times.py` can be run in order to plot discovery times and run times of precedent runs.

A.2 Structure

Files in the `proj/src` folder are organized as follows:

`graph.py` contains the implementation of a class `Grap`. It represents a graph, and internally consists on the adjacency matrix of the graph built starting by a list of nodes and edges. it implements methods useful for the algorithms (such as `merge_vertices()` and `cut_weight()`);

`stoer_wagner.py` contains the implementation of the Stoer and Wagner algorithm, along with all the functions seeded by the algorithm as subroutines (e.g. `stMinCut`);

`karger_stein.py` contains the implementation of the Karger and Stein algorithm, along with all the functions seeded by the algorithm as subroutines (e.g. `rec_contract`, `contract`);

`hybrid.py` contains the hybrid approach implementation and all the auxiliary functions needed (e.g. `hybrid_iteration`);

`test.py` contains the functions to test each single algorithm and a general `mesure_run_time` function to collect results of tests run in parallel on a given collection of inputs, in order to run multiple test in parallel and waste as little time as possible (each test runs on a dedicated processor, and does not exchange data with anything outside itself, therefore can return reliable time measurements). Each test is run only one time, since the computation of the min cut of each graph is long enough to have reliable data with just one run;

`rapresentation.py` contains functions to plot, print, read ad write csv files of data, relies on the common tuple rapresentation of a single run: `(name, time, discovery time, #vertices, #edges, solution)`;

`custom_queue.py` contains the custom max heap implementation as described in 3.1;

`files.py` contains and exports a simple list with all the files in the `dataset` folder written manually, ideally it could read the directory and return all the files in it but was out of the scope of this project;

`file_parser.py` contains a parser function that reads a file in the dataset and returns a graph based on its contents;

`plot_*.py` are scripts to plot the collected results.

A.3 Collecting results

Each run of a test function (`test_sw`, `test_ks`, `test_hy`) writes the partial results in a file, respectively `stoer_wagner`, `karger_stein` and `hybrid`, in order to use the data also later. The `plot_d_time.py` `plot_run_times.py` scripts read these files and produced the graphs found in this document for the discovery times and run times analysis (5.2 and 5.1).

A.4 Given results

Inside of the `project` folder, three files (`stoer_wagner`, `karger_stein` and `hybrid`) represent a run carried on the 15 oct 2022, and contains the results collected to make this document. The following are the raw results, to show that each of the algorithms returns the same solution for each graph. Another script, `check-results.py` is also left in the repository in order to verify that each run of each algorithm returned the same min cut solution.

A.4.1 Karger and Stein

Name	Run time	Discovery time	vertices	edges	solution
input_random_01_10.txt	21955203	1174838	10	14	3056.0
input_random_02_10.txt	20413053	753182	10	10	223.0
input_random_03_10.txt	20776475	747968	10	12	2302.0
input_random_04_10.txt	20472007	1516644	10	11	4974.0
input_random_05_20.txt	274192186	2803016	20	24	1526.0
input_random_06_20.txt	270699951	2618570	20	24	1684.0
input_random_07_20.txt	264952076	3521828	20	27	522.0
input_random_08_20.txt	265902617	3433319	20	25	2866.0
input_random_09_40.txt	1476586536	16951262	40	52	2137.0
input_random_10_40.txt	1475485029	12473544	40	54	1446.0
input_random_11_40.txt	1454066644	101075577	40	51	648.0
input_random_12_40.txt	1478923010	16581984	40	50	2486.0
input_random_13_60.txt	8637836227	31530458	60	82	1282.0
input_random_14_60.txt	9342201751	31317170	60	72	299.0
input_random_15_60.txt	9631183852	35754003	60	83	2113.0
input_random_16_60.txt	9673709351	35012807	60	79	159.0
input_random_17_80.txt	13897099761	74847998	80	101	969.0
input_random_18_80.txt	14021950241	75411182	80	105	1756.0
input_random_19_80.txt	12779063559	68113521	80	108	714.0

Continued on next page

Continued from previous page

Name	Run time	Discovery time	vertices	edges	solution
input_random_20_80.txt	14102664257	75610855	80	108	2610.0
input_random_21_100.txt	33997418357	136236155	100	128	341.0
input_random_22_100.txt	34435586028	136011572	100	120	890.0
input_random_23_100.txt	34251055196	147592742	100	125	772.0
input_random_24_100.txt	34410125153	969611258	100	133	1561.0
input_random_25_150.txt	151399865942	745934464	150	197	951.0
input_random_26_150.txt	156699196372	539111706	150	206	424.0
input_random_27_150.txt	156060167624	504156601	150	195	1153.0
input_random_28_150.txt	156187861829	495752154	150	198	707.0
input_random_29_200.txt	496395640857	1237153250	200	276	484.0
input_random_30_200.txt	497940654946	1360465386	200	260	850.0
input_random_31_200.txt	483762197450	9704456269	200	269	1382.0
input_random_32_200.txt	491288826973	1219510888	200	274	1102.0
input_random_33_250.txt	1408696415850	2614073674	250	317	346.0
input_random_34_250.txt	1368501989564	2443071985	250	322	381.0
input_random_35_250.txt	1374163922731	116600640150	250	338	129.0
input_random_36_250.txt	1375792730008	2486468842	250	326	670.0
input_random_37_300.txt	2160360481508	4729169208	300	403	1137.0
input_random_38_300.txt	2067086037812	4753854152	300	393	869.0
input_random_39_300.txt	2072898800415	179186798959	300	408	868.0
input_random_40_300.txt	1923396947986	5522981742	300	411	1148.0
input_random_41_350.txt	4870485051012	7870926943	350	468	676.0
input_random_42_350.txt	4898918023563	7745123834	350	475	290.0
input_random_43_350.txt	5389664961634	113102192992	350	462	818.0
input_random_44_350.txt	4930420561984	8317874494	350	474	175.0
input_random_45_400.txt	8086065229811	13289047299	400	543	508.0
input_random_46_400.txt	7298649248233	11756584059	400	527	904.0
input_random_47_400.txt	7315726405427	11872670003	400	526	362.0
input_random_48_400.txt	7316690453540	19451067426	400	525	509.0
input_random_49_450.txt	19609649101979	43061356586	450	595	400.0
input_random_50_450.txt	17094362531762	18633946835	450	602	364.0
input_random_51_450.txt	15998863588135	19416630027	450	593	336.0
input_random_52_450.txt	15400159582561	17399072176	450	594	639.0
input_random_53_500.txt	19696179196609	26037970924	500	670	43.0
input_random_54_500.txt	19689858290136	371243413033	500	671	805.0
input_random_55_500.txt	24796428460856	44263417192	500	670	363.0
input_random_56_500.txt	21542543097327	220545587133	500	666	584.0

A.4.2 Stoer and Wagner

Name	Run time	Discovery time	vertices	edges	solution
input_random_01_10.txt	1868183	444901	10	14	3056.0
input_random_02_10.txt	1380138	1377095	10	10	223.0
input_random_03_10.txt	1408000	1405271	10	12	2302.0
input_random_04_10.txt	1413036	197632	10	11	4974.0
input_random_05_20.txt	6274105	461092	20	24	1526.0
input_random_06_20.txt	6017664	2003795	20	24	1684.0
input_random_07_20.txt	6424315	5577288	20	27	522.0
input_random_08_20.txt	5960108	5376774	20	25	2866.0
input_random_09_40.txt	30056332	13217218	40	52	2137.0
input_random_10_40.txt	28955114	3882604	40	54	1446.0
input_random_11_40.txt	28514608	1110675	40	51	648.0
input_random_12_40.txt	29238009	3817260	40	50	2486.0
input_random_13_60.txt	78069225	78033753	60	82	1282.0
input_random_14_60.txt	74788053	74716488	60	72	299.0
input_random_15_60.txt	77171997	21217012	60	83	2113.0
input_random_16_60.txt	76838371	11816505	60	79	159.0
input_random_17_80.txt	160722075	2753487	80	101	969.0
input_random_18_80.txt	160732488	6865519	80	105	1756.0
input_random_19_80.txt	162146962	162086843	80	108	714.0
input_random_20_80.txt	156576268	21843238	80	108	2610.0
input_random_21_100.txt	291154524	4052142	100	128	341.0
input_random_22_100.txt	281328196	3872485	100	120	890.0
input_random_23_100.txt	283306668	42489851	100	125	772.0
input_random_24_100.txt	288681502	15789615	100	133	1561.0
input_random_25_150.txt	888806863	66819211	150	197	951.0
input_random_26_150.txt	933381691	19517209	150	206	424.0
input_random_27_150.txt	928410676	20067148	150	195	1153.0
input_random_28_150.txt	934776976	20166095	150	198	707.0
input_random_29_200.txt	2128443446	2127167507	200	276	484.0
input_random_30_200.txt	2118771350	33580391	200	260	850.0
input_random_31_200.txt	2089197558	12115377	200	269	1382.0
input_random_32_200.txt	2113652689	97629097	200	274	1102.0
input_random_33_250.txt	4149600610	17868924	250	317	346.0
input_random_34_250.txt	4349838016	17913841	250	322	381.0
input_random_35_250.txt	4389176835	18926186	250	338	129.0
input_random_36_250.txt	4371370387	224112624	250	326	670.0

Continued on next page

Continued from previous page

Name	Run time	Discovery time	vertices	edges	solution
input_random_37_300.txt	7536406300	357329473	300	403	1137.0
input_random_38_300.txt	7902264998	24722137	300	393	869.0
input_random_39_300.txt	7842301919	130103355	300	408	868.0
input_random_40_300.txt	7522399111	172481124	300	411	1148.0
input_random_41_350.txt	12300730169	101696946	350	468	676.0
input_random_42_350.txt	12353832981	12346669225	350	475	290.0
input_random_43_350.txt	12601069776	96861288	350	462	818.0
input_random_44_350.txt	12690842484	12685652056	350	474	175.0
input_random_45_400.txt	19248320867	39073104	400	543	508.0
input_random_46_400.txt	18822976449	135980810	400	527	904.0
input_random_47_400.txt	18086670915	18080326232	400	526	362.0
input_random_48_400.txt	17492311052	573160494	400	525	509.0
input_random_49_450.txt	29907533730	29905283808	450	595	400.0
input_random_50_450.txt	29053648353	48436245	450	602	364.0
input_random_51_450.txt	27014829338	522779458	450	593	336.0
input_random_52_450.txt	25655230862	272333112	450	594	639.0
input_random_53_500.txt	36942166036	346434119	500	670	43.0
input_random_54_500.txt	36908249499	785632091	500	671	805.0
input_random_55_500.txt	43592181837	43583246488	500	670	363.0
input_random_56_500.txt	40277008215	1967361752	500	666	584.0

A.4.3 Hybrid

Name	Run time	Discovery time	vertices	edges	solution
input_random_01_10.txt	4088655	3351086	10	14	3056.0
input_random_02_10.txt	3294920	3291836	10	10	223.0
input_random_03_10.txt	3462209	2713269	10	12	2302.0
input_random_04_10.txt	3372256	2643410	10	11	4974.0
input_random_05_20.txt	19791194	16138175	20	24	1526.0
input_random_06_20.txt	19137502	16760852	20	24	1684.0
input_random_07_20.txt	19238629	18836466	20	27	522.0
input_random_08_20.txt	19061688	18647480	20	25	2866.0
input_random_09_40.txt	87344273	78245010	40	52	2137.0
input_random_10_40.txt	86141798	71389457	40	54	1446.0
input_random_11_40.txt	84159723	69195570	40	51	648.0
input_random_12_40.txt	85501253	71220215	40	50	2486.0
input_random_13_60.txt	276781947	276761089	60	82	1282.0

Continued on next page

Continued from previous page

Name	Run time	Discovery time	vertices	edges	solution
input_random_14_60.txt	266818369	266764735	60	72	299.0
input_random_15_60.txt	280281449	250188309	60	83	2113.0
input_random_16_60.txt	281927233	246319929	60	79	159.0
input_random_17_80.txt	583545683	499974956	80	101	969.0
input_random_18_80.txt	588730382	507793920	80	105	1756.0
input_random_19_80.txt	576544114	576503083	80	108	714.0
input_random_20_80.txt	574857019	499417765	80	108	2610.0
input_random_21_100.txt	1056989502	907746217	100	128	341.0
input_random_22_100.txt	1034288335	888690879	100	120	890.0
input_random_23_100.txt	1033052050	896796835	100	125	772.0
input_random_24_100.txt	1052319379	911892872	100	133	1561.0
input_random_25_150.txt	4065041217	3642696289	150	197	951.0
input_random_26_150.txt	4297474170	3801951454	150	206	424.0
input_random_27_150.txt	4284086138	3801383003	150	195	1153.0
input_random_28_150.txt	4361882227	3839540132	150	198	707.0
input_random_29_200.txt	10953520005	10951529912	200	276	484.0
input_random_30_200.txt	10841043139	9664725165	200	260	850.0
input_random_31_200.txt	9783532421	8684263208	200	269	1382.0
input_random_32_200.txt	10332277585	9185811855	200	274	1102.0
input_random_33_250.txt	21811819289	19565854091	250	317	346.0
input_random_34_250.txt	20804383930	18544152279	250	322	381.0
input_random_35_250.txt	21078613696	18828802520	250	338	129.0
input_random_36_250.txt	21032298627	18832366721	250	326	670.0
input_random_37_300.txt	39790831942	35683660284	300	403	1137.0
input_random_38_300.txt	39258689960	35152327934	300	393	869.0
input_random_39_300.txt	38485959091	34710415537	300	408	868.0
input_random_40_300.txt	37128197126	33283938006	300	411	1148.0
input_random_41_350.txt	62104527340	55815830795	350	468	676.0
input_random_42_350.txt	62594969768	62587378081	350	475	290.0
input_random_43_350.txt	66709504245	59824164725	350	462	818.0
input_random_44_350.txt	63536744935	63528928898	350	474	175.0
input_random_45_400.txt	118797461358	108884681048	400	543	508.0
input_random_46_400.txt	109811983801	100672401710	400	527	904.0
input_random_47_400.txt	105986835038	105977818024	400	526	362.0
input_random_48_400.txt	104418502238	96070943601	400	525	509.0
input_random_49_450.txt	194304130909	194289584026	450	595	400.0
input_random_50_450.txt	175572221446	161672580462	450	602	364.0

Continued on next page

Continued from previous page

Name	Run time	Discovery time	vertices	edges	solution
input_random_51_450.txt	162423749089	148967096585	450	593	336.0
input_random_52_450.txt	150508799752	137813743111	450	594	639.0
input_random_53_500.txt	219140835732	200789502784	500	670	43.0
input_random_54_500.txt	219056925829	201044112385	500	671	805.0
input_random_55_500.txt	291439211170	291420484239	500	670	363.0
input_random_56_500.txt	242646544130	223077731877	500	666	584.0