Comparazione algoritmi risolutivi per il problema del minimo taglio in grafo

Progetto corso AMOD

Andrea Di Iorio
Università degli studi Roma Torvergata

ABSTRACT

In questo progetto mi sono occupato di effettuare uno studio comparativo tra due algoritmi risolutivi per il problema del minimo taglio in un grafo non orientato e connesso.

KEYWORDS
minCut, linear programming, gurobi, stoer_wagner, random algoritm, kargerAlgoritm,

1 Introduzione

Gl'algoritmi che sono andato a confrontare ed implementare sono l'algoritmo randomizato di Karger e un algoritmo iterativo basato sulla risoluzione di un modello di programmazione lineare ottenuto dal duale del problema del massimo flusso.

Nella comparazione dei risultati ottenuti da questi algoritmi ho considerato anche un altro algoritmo deterministico per la risoluzione del problema in oggetto ovvero l'algoritmo stoer_wagner [5]

Nel seguito, per descrivere un generico grafo *G*, mi riferirò al suo numero di nodi con *n* e al suo numero di archi con *e*

2 Implementazione

2.1 Algoritmo iterativo basato sulla programmazione lineare

Inizialmente viene convertito il grafo non orientato in un equivalente grafo orientato in cui ogni arco non orientato viene sostituito con una coppia di archi orientati.

successivamente per tutte le possibili $\frac{n(n-1)}{2}$ coppie non ordinate di nodi (s,t) risolvo un modello di programmazione linare ottenuto dal duale di una formulazione del problema del massimo flusso ottenendo dei tagli nel grafo.

Per la risoluzione del modello ho utilizzato il wrap di gurobi per python3.

Infine, ritorno il taglio minimo tra tutti quelli trovati.

Di seguito il modello di programmazione lineare [1]

$$\min \sum_{(i,j)\in A} k_{ij} y_{ij} \tag{7}$$

$$u_i - u_j + y_{ij} \geq 0, (i,j) \in A \tag{8}$$

$$-u_s + u_t = 1 \tag{9}$$

$$y_{ij} \geq 0 \tag{10}$$

le variabili y_{ij} sono relative a gli archi nel grafo, k_{ij} sono relative agl'eventuali pesi associati agl'archi del grafo e le variabili u_i sono relative ai nodi del grafo.

2.2 Algoritmo randomizzato di Karger D.

L'algoritmo è basato sul concetto di contrazione di un arco.
Contrarre un generico arco (n1,n2) di un grafo non orientato comporta la sostituzione dei vertici n1,n2 con un metavertice che ha come vertici incidenti l'unione dei vertici incidenti ad n1,n2.
Tramite n-2 contrazioni di archi scelti casualmente nel grafo si ottengono 2 (meta)vertici residui, in cui gli archi che li connettono identificano un taglio che può essere minimo

2.2.1 Scelta uniformemente casuale degli archi da contrarre.

Ho realizzato due modi distinti per realizzare un meccanismo di estrazione casuale di un arco in un grafo.
- Uno basato su una banale estrazione casuale all'interno di una lista di archi, che deve essere costruita su necessità nel caso il grafo sia rappresentato mediante liste di adiacenza

- Uno basato su un semplice algoritmo che associa ad ogni vertice un intervallo di numeri, la cui lunghezza è proporzionale al grado del vertice
A supporto di questa associazione sfrutto una Hashmap dinamicamente aggiornata ad ogni contrazione, che mappa per ogni nodo, il suo grado.
Tramite un'estrazione casuale uniforme all'interno di un macro intervallo che include tutti gl'intervalli allocati per i nodi, identifico un nodo n1 che sarà un estremo dell'arco casuale.
l'altro estremo dell'arco casuale n2 è scelto uniformante tra i nodi adiacenti ad n1.

2.3 Generazione di grafi casuali

Al fine di testare e confrontare gli algoritmi descritti, ho realizzato un meccanismo per la generazioni di grafi casuali, basato sulla generazione di alberi casuali, che poi vengono progressivamente "infoltiti" tramite l'aggiunta di archi casuali.
Per creare un albero casuale di n nodi parto da un prufer code casuale, generando n-2 numeri casuali compresi in (0,n) a cui viene associato univocamente un albero tramite l'algoritmo descritto in [2]
Successivamente aggiungo all'albero degli archi casuali tramite una selezione casuale di coppie di vertici nel grafo che identificano un nuovo arco valido.

3 CONFRONTO TRA GL'ALGORITMI

Ho confrontato questi due algoritmi mediante la loro esecuzione su alcuni grafi generati casualmente in cui ho fatto variare sia il numero di vertici che il numero di archi.
In particolare, partendo da un albero casuale generato come descritto in 2.3 ho progressivamente aggiunto nuovi insiemi di archi casuali, e per ognuno di questi grafi ho eseguito gli algoritmi descritti.
Tutti i dati relativi all'algoritmo randomizzato sono ottenute in seguito a statistiche calcolate su 100 ripetizioni.

3.1 CONFRONTO TEMPI DI ESECUZIONE

Seguono alcuni grafici che confrontano i tempi d'esecuzione di esecuzione dei vari algoritmi

Ho inoltre confrontato le performance dell'algoritmo randomizzato anche con l'algoritmo stoer wagner [5] per la risoluzione del medesimo problema

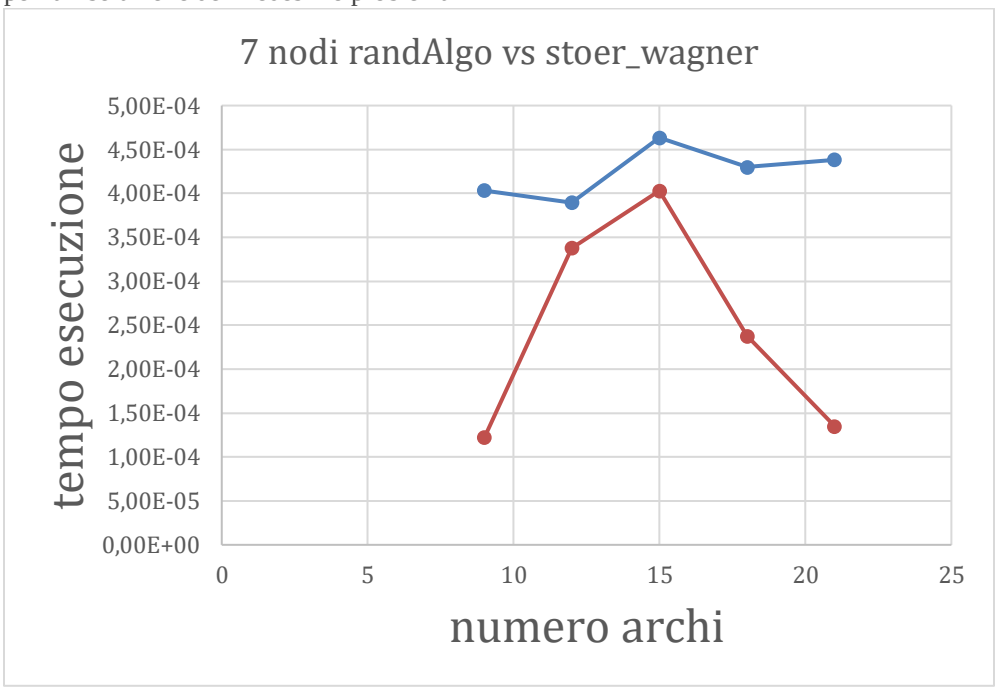


Figure 1 confronto tempi d'esecuzione randAlgo (Rosso) stoerWagner(Blu) su grafi casuali di 7 nodi

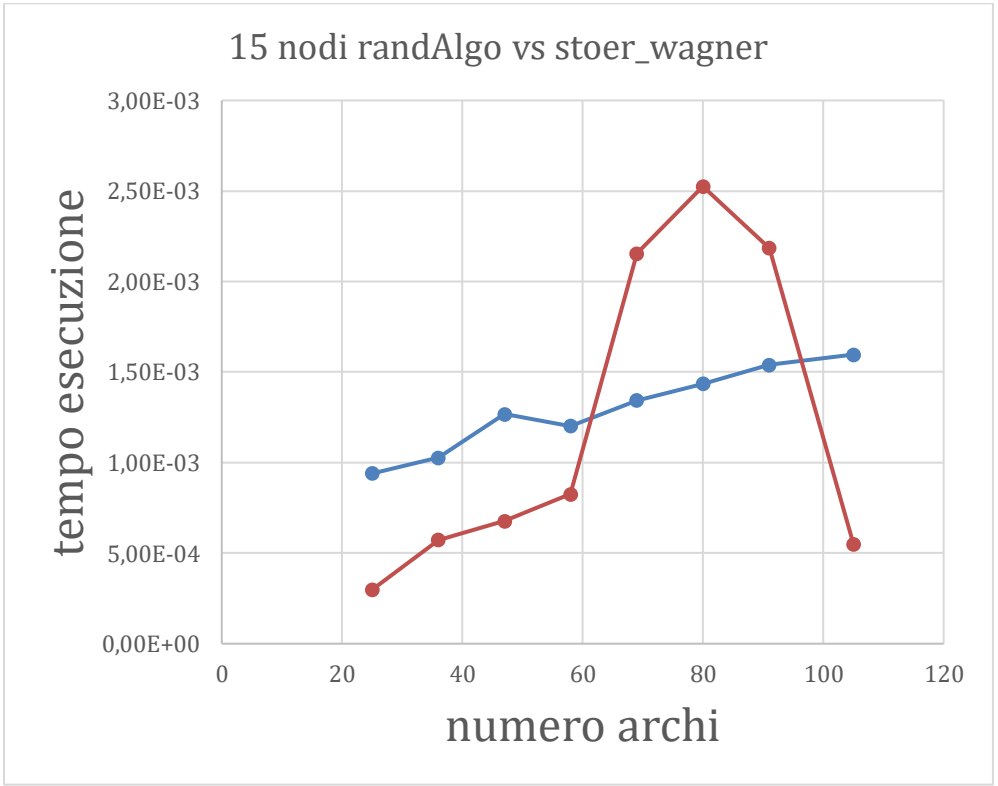


Figure 2 confronto tempi d'esecuzione randAlgo (Rosso) stoerWagner(Blu) su grafi casuali di 15 nodi

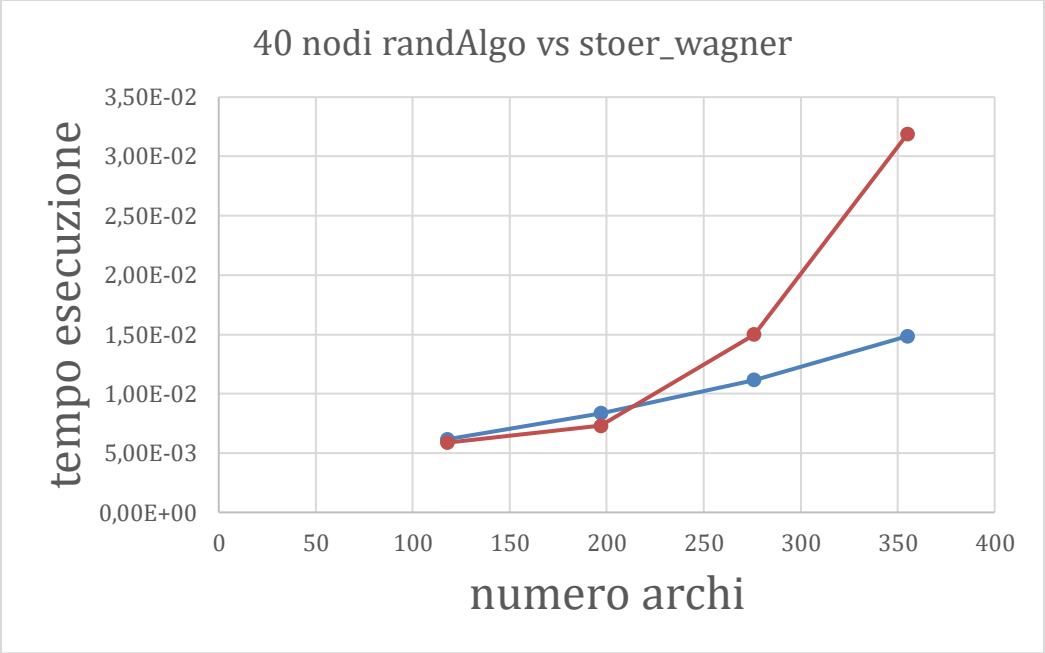


Figure 3 confronto tempi d'esecuzione randAlgo (Rosso) stoerWagner(Blu) su grafi casuali di 40 nodi

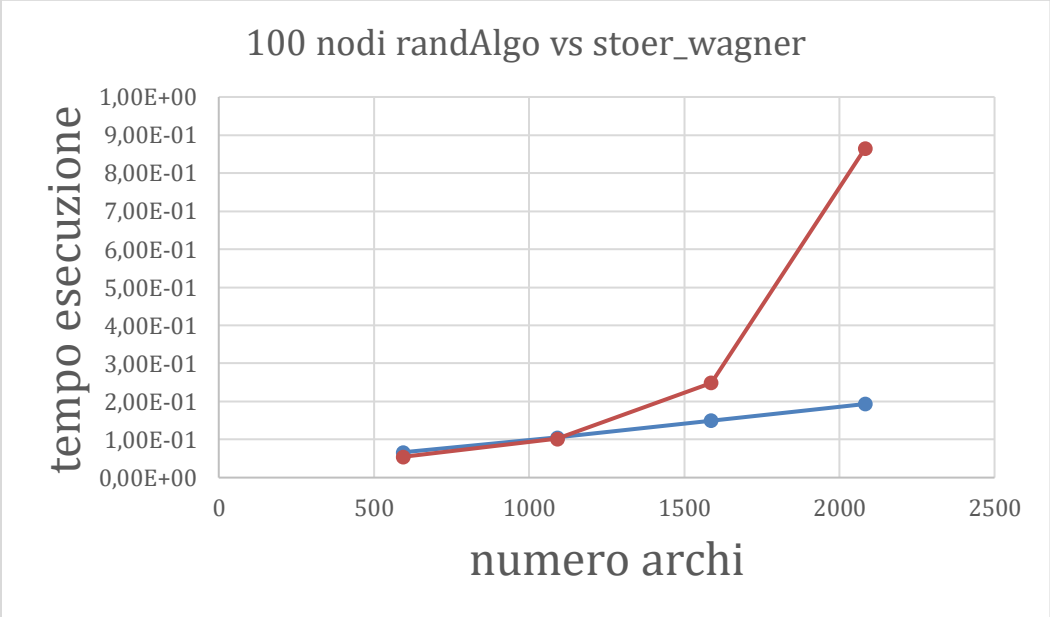


Figure 4 confronto tempi d'esecuzione randAlgo (Rosso) stoerWagner(Blu) su grafi casuali di 100 nodi

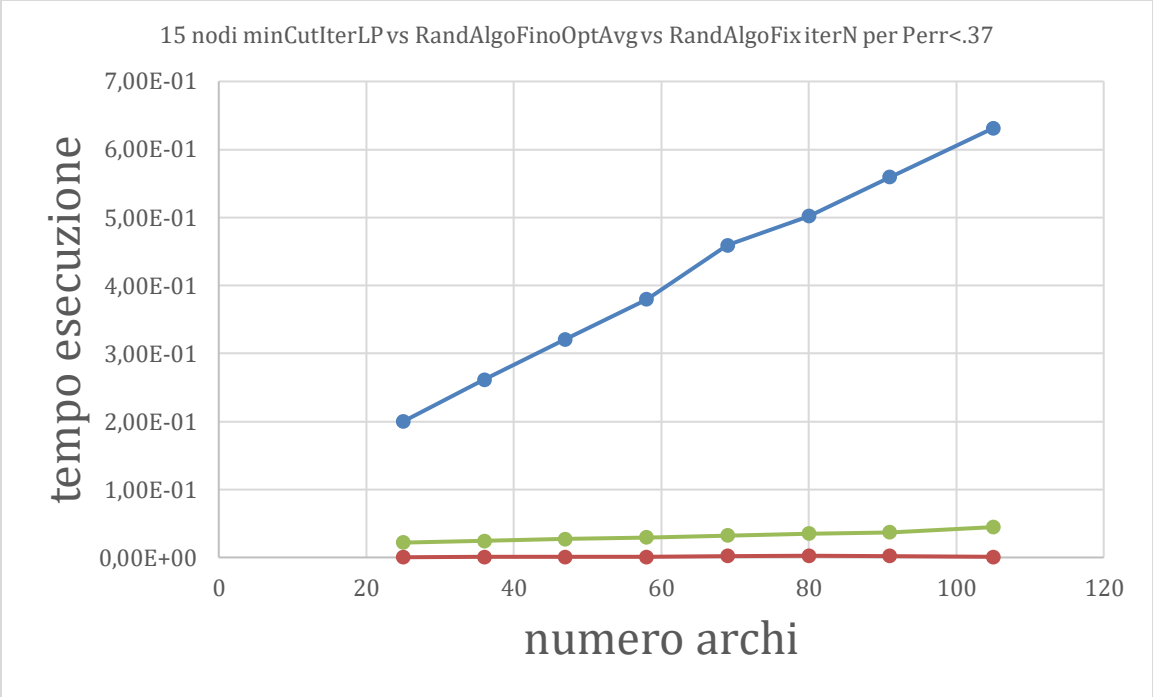


Figure 5 confronto tempi d'esecuzione randAlgo fino a raggiungerel'ottimo (Rosso), randAlgo eseguito per il numero d'iterazioni teorico necessari per una Prob. d'err < 0.37 (verde) , minCutIterativePL (Blu) su grafi casuali di 15 nodi

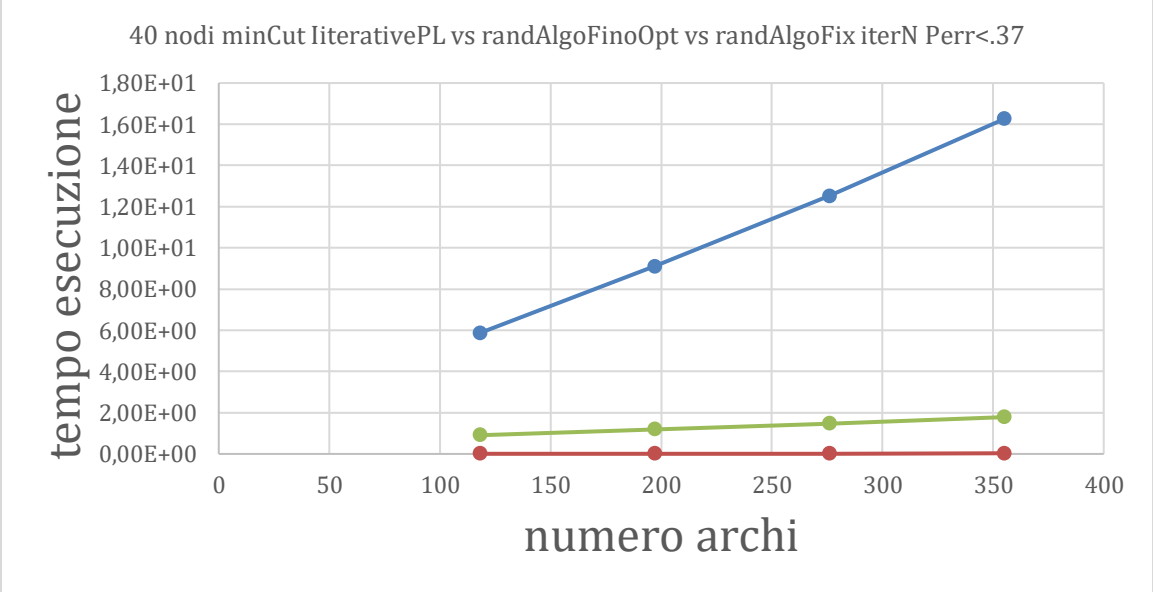


Figure 6 confronto tempi d'esecuzione randAlgo fino a raggiungerel'ottimo (Rosso), randAlgo eseguito per il numero d'iterazioni teorico necessari per una Prob. d'err < 0.37 (verde) , minCutIterativePL (Blu) su grafi casuali di 40 nodi

Come è possibile vedere sia aumentando il numero di archi che di nodi comporta un maggiore tempo di esecuzione per tutti gl'algoritmi.
Le performance dell'algoritmo randomizzato sono sempre migliori delle performance dell'algoritmo basato sulla programmazione lineare, e confrontabili con le performance dell'algoritmo stoer wagner

3.2 VERIFICA PROBABILITÀ D'ERRORE DELL'ALGORITMO RANDOMIZZATO

la probabilità d'errore teorica dell'algoritmo randomizzato di non trovare il minimo taglio con dato numero di ripetizioni è approssimabile ad $\frac{1}{e}$ con $\frac{n*n}{2}$ ripetizioni dell'algoritmo randomizzato come espresso in [3].
In generale si può ottenere il numero di ripetizioni teoricamente necessario per ottenere una data probabilità d'errore, in particolare con $\binom{n}{2}\ln(n)$ iterazioni si può ottenere una probabilità d'errore inferiore a $\frac{1}{n}$ [4]
Prendendo come riferimento il numero di iterazioni necessario per ottenere una probabilità d'errore limitata da $\frac{1}{e}$ [3], da una serie di esperimenti condotti su vari grafi casuali si può notare come questa implementazione dell'algoritmo randomizzato raggiunga l'ottimo sempre sotto tale soglia di riferimento, come visibile nella figura 7

PARAMETRI GRAFI RANDOM			statistiche su num. iterazioni registrate fino a ottimo			
minCut size	num nodi	num archi	AVG IT	STD.DEV IterNum	MAX IterNum	num iteraz per Prob d'err = .37
1	7	9	1.39	0,6012613005301035	4	24,5
2	7	12	3.86	3,345492973	19	24,5
3	7	15	4.37	4,217939082	20	24,5
4	7	18	2.46	1,486776732	7	24,5
6	7	21	1.33	0,652191333393457	4	24,5
1	11	16	3.02	2,374017606	15	60,5
2	11	22	4.85	4,110530945	18	60,5
3	11	28	5.3	4,700096711	26	60,5
3	11	34	4.35	4,080936217	20	60,5
3	11	40	2.72	2,644147339	20	60,5
6	11	46	3.39	2,369396633	13	60,5
10	11	55	1.33	0,6674994798166929	4	60,5
1	15	25	1.28	0,6043612539305653	4	112,5
1	15	36	2.32	1,607306549	8	112,5
3	15	47	2.5	2,067057637	14	112,5
5	15	58	2.79	3,029468067	21	112,5
6	15	69	6.84	7,402865948	56	112,5
8	15	80	7.48	7,338125156	38	112,5
10	15	91	6.06	5,09271613	27	112,5
14	15	105	1.35	0,6415731927193978,	4	112,5
2	40	118	5	4,788454127	21	800
5	40	197	4.76	3,934103674	20	800
7	40	276	7.9	6,122487116	29	800
11	40	355	14.06	1,169807029	50	800

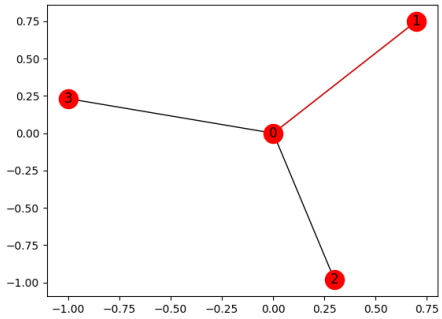
Figure 7

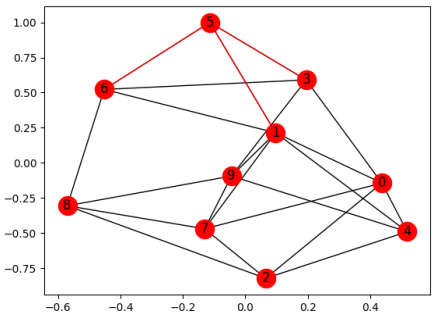
4 TEST

Al fine di testare la validità delle soluzioni trovate, oltre che mediante una rappresentazione grafica dei grafi casuali generati e dei minimi tagli ottenuti su di essi, ho verificato i valori dei tagli ottenuti dagli algoritmi sopra descritti con un i valori prodotti dall'algoritmo stoer_wagner [5], presente nella libreria networkx.

5 RAPPRESENTAZIONE.

tramite le librerie python networkx e matplot ho realizzato alcune funzioni wrapper utili a rappresentare graficamente un grafo ed un suo taglio dato.





REFERENCES

[1] The dual of the maximum flow problem A. Agnetis https://www3.diism.unisi.it/~agnetis/mincutENG.pdf
[2] https://en.wikipedia.org/wiki/Pr%C3%BCfer_sequence
[3] L.stougie https://personal.vu.nl/l.stougie/Courses/RA/Lecture1.pdf
[4] https://en.wikipedia.org/wiki/Karger%27s_algorithm
[5] https://en.wikipedia.org/wiki/Stoer–Wagner_algorithm