A Computational Study on the 2-Edge-Connectivity Augmentation Problem

Jørgen Bang-Jensen, Marco Chiarandini, Peter Morling

Department of Mathematics and Computer Science
University of Southern Denmark

Graph Theory 2007
Fredericia, Denmark, December 6-9, 2007

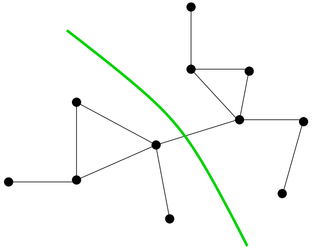
Based on: J. Bang-Jensen, M. Chiarandini, P. Morling (2007).
A computational investigation on heuristic algorithms for 2-edge-connectivity augmentation.
Tech. Rep. DMF-2007-07-005, The Danish Mathematical Society. Submitted to journal.

1. 2-Edge-Connectivity Augmentation
    - The Problem
    - Test Instances
2. Basic Heuristics
    - Construction Heuristics
    - Local Search Algorithms
    - Analysis
3. Advanced Heuristics
    - Design
    - Experimental Analysis

1. 2-Edge-Connectivity Augmentation
   - The Problem
   - Test Instances
2. Basic Heuristics
   - Construction Heuristics
   - Local Search Algorithms
   - Analysis
3. Advanced Heuristics
   - Design
   - Experimental Analysis

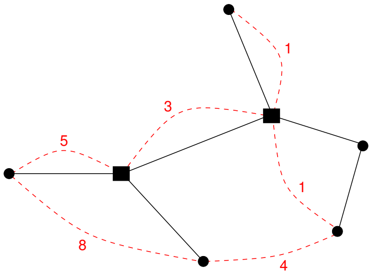
2-Edge-Connectivity Augmentation

A graph $G = (V, E)$ is 2-edge-connected if every non-trivial cut $(U, V - U)$ contains at least 2 edges.



2-Edge-Connectivity Augmentation

A graph $G = (V, E)$ is 2-edge-connected if every non-trivial cut $(U, V - U)$ contains at least 2 edges.

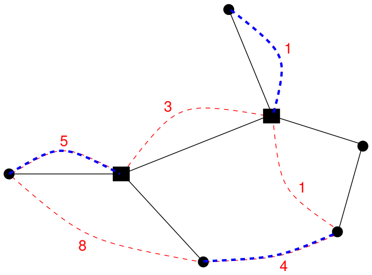


Weighted 2-edge-connectivity augmentation problem

Input: a graph $G = (V, E)$ and a set E' of possible edges to add with a non-negative weight.
Task: Find a minimum cost subset $X \subseteq E'$ so that the graph $A = (V, E \cup X)$ is 2-edge-connected.

2-Edge-Connectivity Augmentation

A graph $G = (V, E)$ is 2-edge-connected if every non-trivial cut $(U, V - U)$ contains at least 2 edges.



Weighted 2-edge-connectivity augmentation problem

Input: a graph $G = (V, E)$ and a set E' of possible edges to add with a non-negative weight.
Task: Find a minimum cost subset $X \subseteq E'$ so that the graph $A = (V, E \cup X)$ is 2-edge-connected.

2-Edge-Connectivity Augmentation

- Connectivity problems arise in the context of designing Survivable Networks.

2-Edge-Connectivity Augmentation

- Connectivity problems arise in the context of designing Survivable Networks.

- We restrict to the case where G is already connected (E1-2AUG).
- $G' = (V, E \cup E')$ may contain parallel edges.
- The E1-2AUG is NP-hard [Frederickson, JáJá, 1981]

2-Edge-Connectivity Augmentation

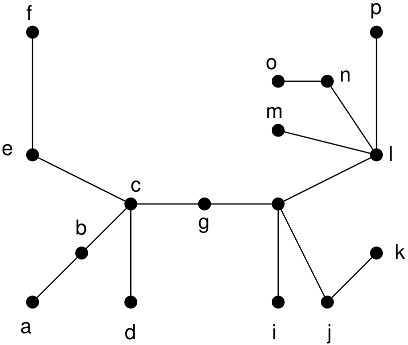
- Connectivity problems arise in the context of designing Survivable Networks.

- We restrict to the case where G is already connected (E1-2AUG).
- $G' = (V, E \cup E')$ may contain parallel edges.
- The E1-2AUG is NP-hard [Frederickson, JáJá, 1981]

- An augmentation $X \subseteq E'$ is proper if $A = (V, E \cup X)$ is 2-edge-connected.
- It can be checked in $O(|V| + |E|)$ [Tarjan, 1974].

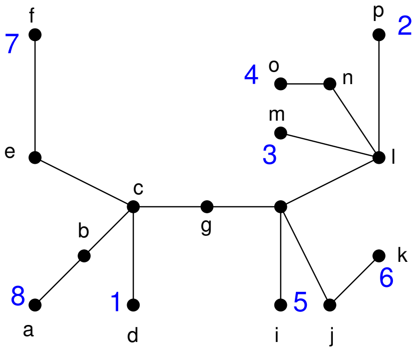
A Polynomially Solvable Case

- $G' = (V, E \cup E')$ is (at least) complete
- uniform weights



A Polynomially Solvable Case

- $G' = (V, E \cup E')$ is (at least) complete
- uniform weights

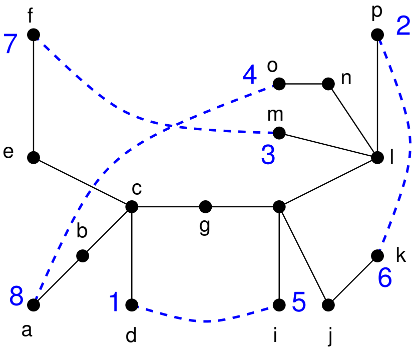


T: a tree with k leaves

Function pair(T) [Eswaran, Tajan, 1976]
Step 1: Fix a leaf u of T and perform a DFS from u labeling the leaves $u_1, u_2, \ldots, u_k$ as they are encountered;
Step 2: $X = \{u_i u_{i+\lfloor \frac{k}{2} \rfloor} : 1 \leq i \leq \lceil \frac{k}{2} \rceil\}$;
return X.

A Polynomially Solvable Case

- $G' = (V, E \cup E')$ is (at least) complete
- uniform weights

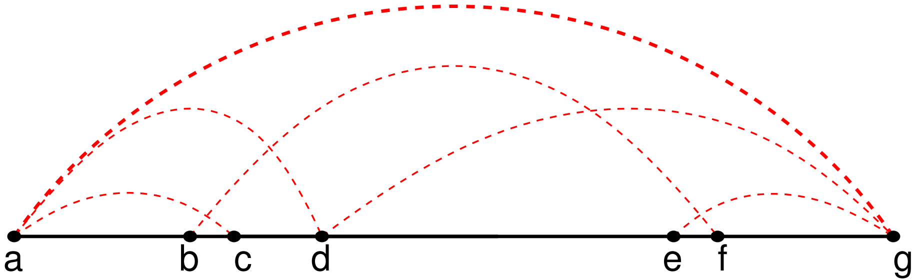


T: a tree with k leaves

Function pair(T) [Eswaran, Tajan, 1976]
Step 1: Fix a leaf u of T and perform a DFS from u labeling the leaves $u_1, u_2, \ldots, u_k$ as they are encountered;
Step 2: $X = \{u_i u_{i+\lfloor \frac{k}{2} \rfloor} : 1 \leq i \leq \lceil \frac{k}{2} \rceil\}$;
return X.

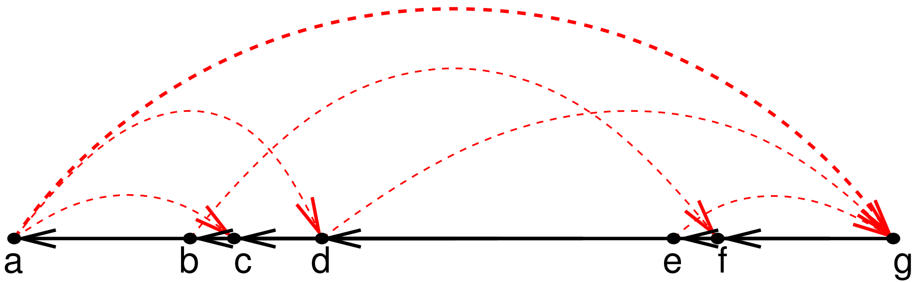
A Polynomially Solvable Case

- The graph G is a path



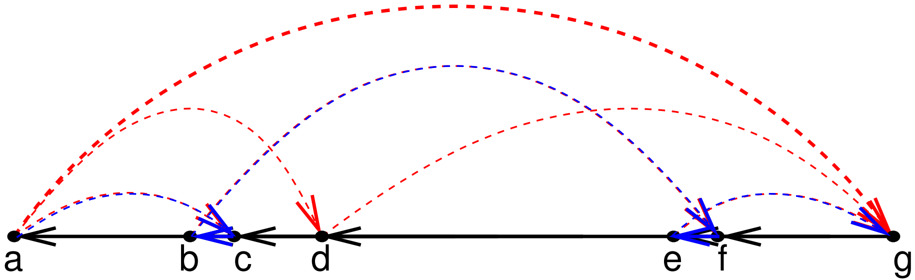
A Polynomially Solvable Case

- The graph G is a path



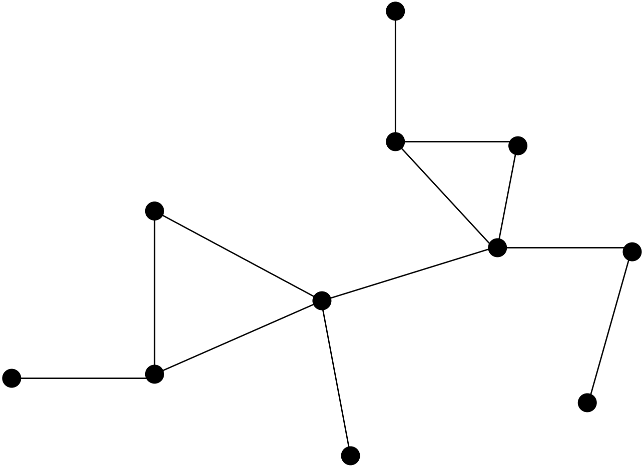
A Polynomially Solvable Case

- The graph G is a path



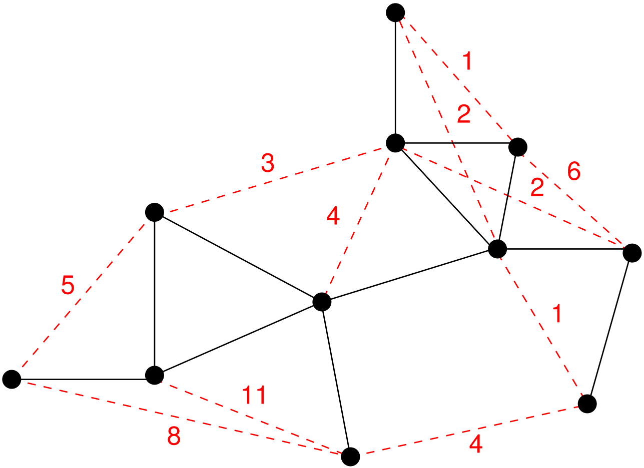
Reductions

- Reduction to augmentation of spanning tree: $O(|V| + |E|)$ [Tarjan, 1974]



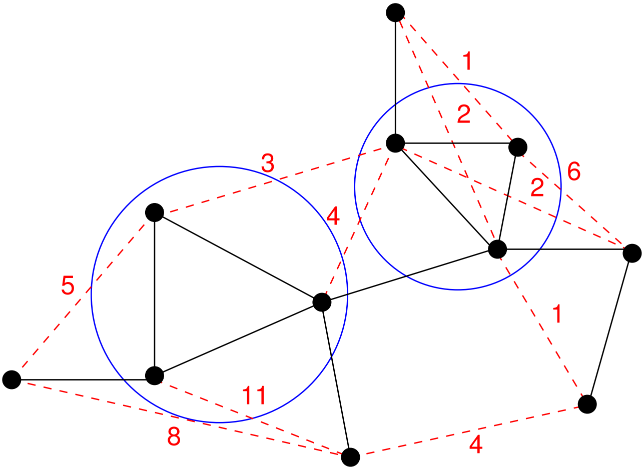
Reductions

- Reduction to augmentation of spanning tree: $O(|V| + |E|)$ [Tarjan, 1974]



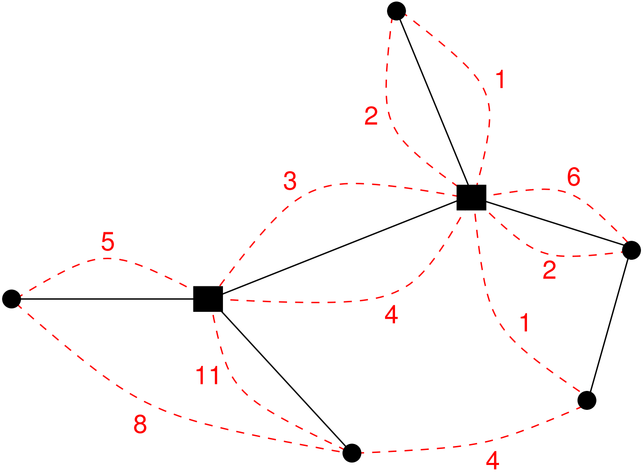
Reductions

- Reduction to augmentation of spanning tree: $O(|V| + |E|)$ [Tarjan, 1974]



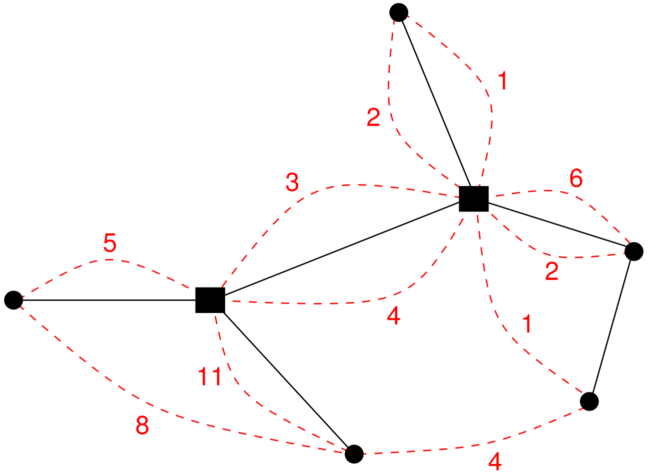
Reductions

- Reduction to augmentation of spanning tree: $O(|V| + |E|)$ [Tarjan, 1974]



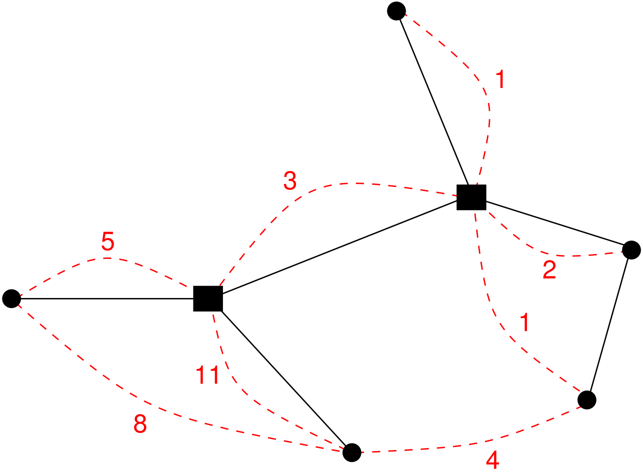
Reductions

- Reduction to augmentation of spanning tree: $O(|V| + |E|)$ [Tarjan, 1974]
- Edge elimination $O(|V|^2)$ [Frederickson, JáJá, 1981]



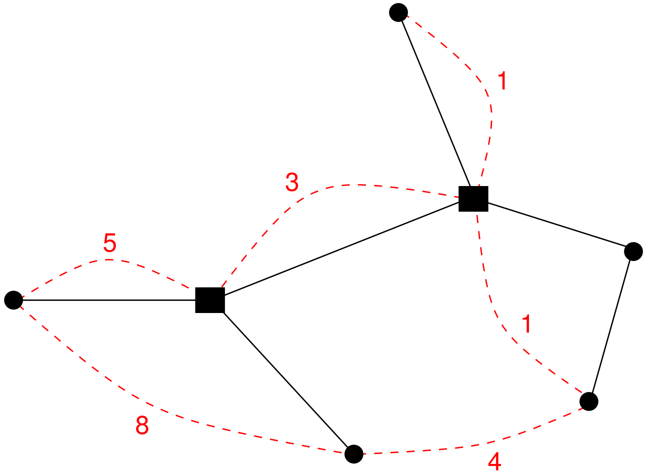
Reductions

- Reduction to augmentation of spanning tree: $O(|V| + |E|)$ [Tarjan, 1974]
- Edge elimination $O(|V|^2)$ [Frederickson, JáJá, 1981]



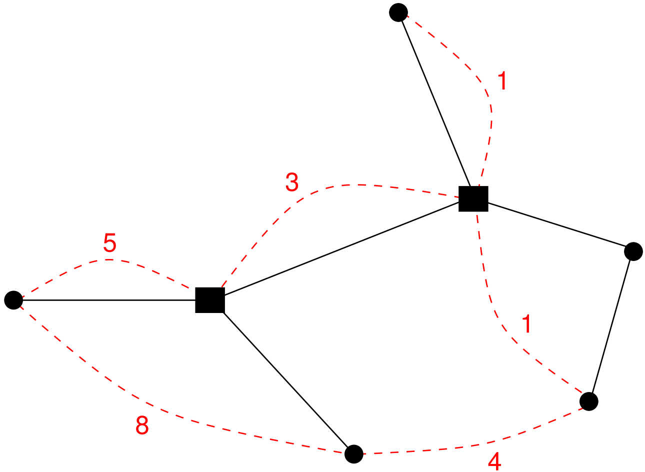
Reductions

- Reduction to augmentation of spanning tree: $O(|V| + |E|)$ [Tarjan, 1974]
- Edge elimination $O(|V|^2)$ [Frederickson, JáJá, 1981]



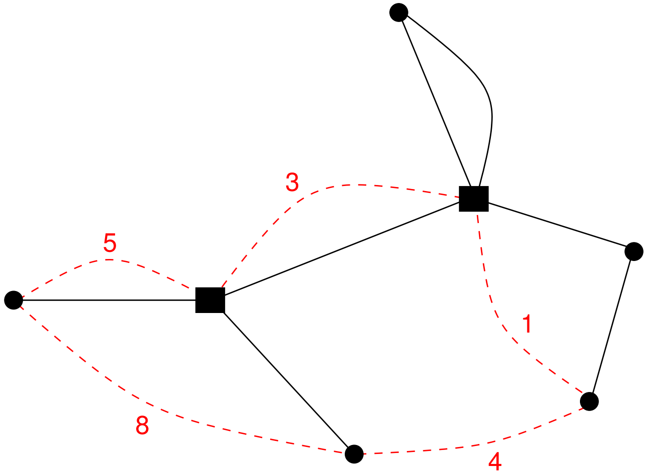
Reductions

- Reduction to augmentation of spanning tree: $O(|V| + |E|)$ [Tarjan, 1974]
- Edge elimination $O(|V|^2)$ [Frederickson, JáJá, 1981]
- Edge fixing $O(|E'||F|)$



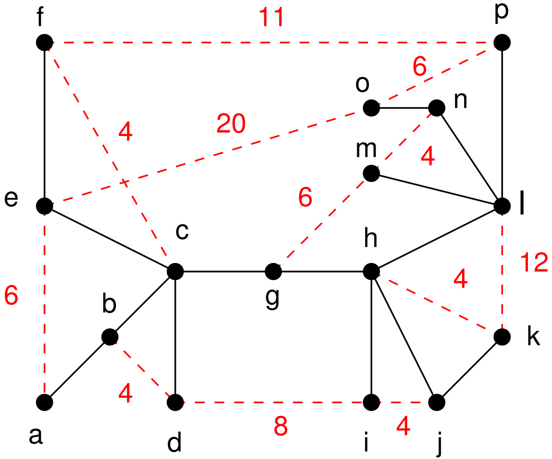
Reductions

- Reduction to augmentation of spanning tree: $O(|V| + |E|)$ [Tarjan, 1974]
- Edge elimination $O(|V|^2)$ [Frederickson, JáJá, 1981]
- Edge fixing $O(|E'||F|)$



Set Covering Formulation

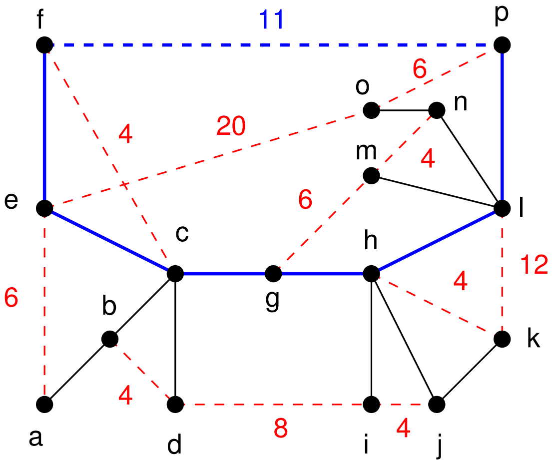
[Conforti, Galluccio and Proietti, 2004]



The dotted blue edge covers the edges on the blue path in T

Set Covering Formulation

[Conforti, Galluccio and Proietti, 2004]



The dotted blue edge covers the edges on the blue path in T

Set Covering Formulation

[Conforti, Galluccio and Proietti, 2004]

$F = \{f_1, f_2, \ldots, f_{n-1}\}$ edges of the spanning tree T
$E' = \{e_1, e_2, \ldots, e_m\}$ augmenting edges

Set Covering Formulation

[Conforti, Galluccio and Proietti, 2004]

$F = \{f_1, f_2, \ldots, f_{n-1}\}$ edges of the spanning tree T
$E' = \{e_1, e_2, \ldots, e_m\}$ augmenting edges

for $e_j = uv$, we have

$$M_{ij} = \begin{cases} 1 & \text{if the unique } (uv)\text{-path in } T \text{ contains } f_i, \\ 0 & \text{otherwise.} \end{cases}$$

Set Covering Formulation

[Conforti, Galluccio and Proietti, 2004]

$F = \{f_1, f_2, \ldots, f_{n-1}\}$ edges of the spanning tree T
$E' = \{e_1, e_2, \ldots, e_m\}$ augmenting edges

for $e_j = uv$, we have

$$M_{ij} = \begin{cases} 1 & \text{if the unique } (uv)\text{-path in } T \text{ contains } f_i, \\ 0 & \text{otherwise.} \end{cases}$$

$$\begin{array}{ll} \min & \vec{\omega}^T \vec{x} \\ \text{s.t.} & \begin{array}{c} f_1 : \\ f_2 : \\ \vdots \\ \vdots \\ f_n : \end{array} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ \vdots & \ddots & & & & \vdots \\ \vdots & & & & & \vdots \\ 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} \geq \begin{bmatrix} 1 \\ 1 \\ \vdots \\ \vdots \\ 1 \end{bmatrix} \\ & \vec{x} \in \{0,1\}^m \end{array}$$

Trimming an Augmentation

An edge augmentation is said minimal
if no further edge can be deleted without creating a bridge in the graph A.

Every optimal edge augmentation is minimal.

Trimming an Augmentation

An edge augmentation is said minimal
if no further edge can be deleted without creating a bridge in the graph A.

Every optimal edge augmentation is minimal.

T the spanning tree

Trimming an Augmentation

An edge augmentation is said minimal
if no further edge can be deleted without creating a bridge in the graph A.

Every optimal edge augmentation is minimal.

T the spanning tree
X and Y two disjoint sets of edges not in T

Trimming an Augmentation

An edge augmentation is said minimal
if no further edge can be deleted without creating a bridge in the graph A.

Every optimal edge augmentation is minimal.

T the spanning tree
X and Y two disjoint sets of edges not in T
$X \cup Y$ is a proper augmentation

Trimming an Augmentation

An edge augmentation is said minimal
if no further edge can be deleted without creating a bridge in the graph A.

Every optimal edge augmentation is minimal.

T the spanning tree
X and Y two disjoint sets of edges not in T
$X \cup Y$ is a proper augmentation

Function trim(T,X,Y)
Order the edges in Y as $y_1, y_2, \ldots, y_q$, $q = |Y|$

Trimming an Augmentation

An edge augmentation is said minimal
if no further edge can be deleted without creating a bridge in the graph A.

Every optimal edge augmentation is minimal.

T the spanning tree
X and Y two disjoint sets of edges not in T
$X \cup Y$ is a proper augmentation

Function trim(T,X,Y)
Order the edges in Y as $y_1, y_2, \ldots, y_q$, $q = |Y|$
$Y' := Y$
for $i = 1$ to q **do**
  if $T + X + Y' - y_i$ is 2-edge-connected % $O(|V| + |F| + |X| + |Y|)$
  then
    $Y' := Y' - y_i$
return Y'

Trimming an Augmentation

An edge augmentation is said minimal
if no further edge can be deleted without creating a bridge in the graph A.

Every optimal edge augmentation is minimal.

T the spanning tree
X and Y two disjoint sets of edges not in T
$X \cup Y$ is a proper augmentation

Function trim(T,X,Y)
Order the edges in Y as $y_1, y_2, \ldots, y_q$, $q = |Y|$ $\Leftarrow$ non increasing order of weights
$Y' := Y$
for $i = 1$ to q **do**
  if $T + X + Y' - y_i$ is 2-edge-connected % $O(|V| + |F| + |X| + |Y|)$
  then
    $Y' := Y' - y_i$
return Y'

Previous Work

2-approximations:

- G. Frederickson, J. JáJá (1981), "Approximation algorithms for several graph augmentation problems." *SIAM Journal on Computing*, vol. 10
- S. Khuller, R. Thurimella (1993), "Approximation algorithms for graph augmentation." *Journal of Algorithms*, vol. 14

3/2-approximation for case with uniform weigths:

- G. Even, J. Feldman, G. Kortsarz, Z. Nutov (2001), "A 3/2-approximation algorithm for augmenting the edge-connectivity of a graph from 1 to 2 using a subset of a given edge set." In vol. 2129 of *Lecture Notes in Computer Science*.

Polynomially solvable instances:

- K. Eswaran, R. Tarjan (1976), "Augmentation problems." *SIAM J. on Comp.*.
- M. Conforti, A. Galluccio, G. Proietti (2004), "Edge-connectivity augmentation and network matrices." In vol. 3353 of *Lecture Notes in Computer Science*.

Previous Work

Previous computational studies:

- S. Khuller, B. Raghavachari, A. Zhu (1999), "A uniform framework for approximating weighted connectivity problems." *SODA '99*.
- G. R. Raidl, I. Ljubic (2002), Evolutionary local search for the edge-biconnectivity augmentation problem. *Information Processing Letters*, vol. 82.
- F. Xhafa (2003), "An implementation of a generic memetic algorithm for the edge biconnectivity augmentation problem." *Tech. Rep.* Polytechnic University of Catalonia, Barcelona.

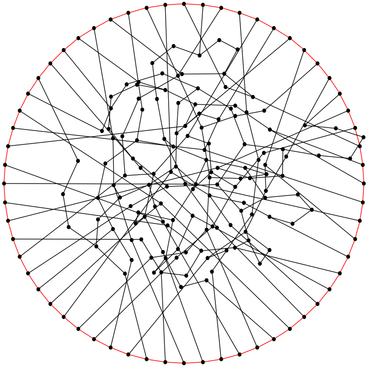
Existing Benchmarks

[Raidl, Ljubic, 2002]

Inst.	$\|V\|$	$\|E'\|$	CPLEX Time	LMS Time
A3	40	29	0.02	0.00
B1	60	55	0.05	0.00
B6	70	81	0.01	0.00
D3	90	366	0.31	0.04
D5	100	398	0.36	0.03
E1	200	19701	14.87	0.12
E2	300	11015	23.20	8.15
E3	400	7621	30.07	11.99
M1	70	290	0.20	0.01
N1	100	1104	0.82	0.05
N2	110	1161	0.94	0.08
R1	200	9715	11.25	0.21
R2	200	9745	8.87	0.27

Test Instances

Tree and Cycle Instances (T+C)
(NP-hard already with uniform weights [Cheriyan, Jordán, Ravi (1999)])

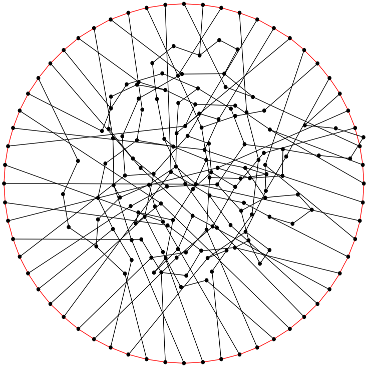


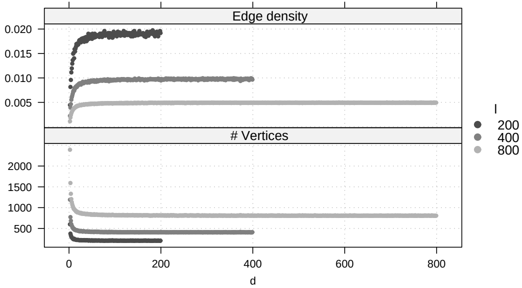
l number of leaves in the graph
d max degree of vertices in T

Test Instances

Tree and Cycle Instances (T+C)

(NP-hard already with uniform weights [Cheriyan, Jordán, Ravi (1999)])





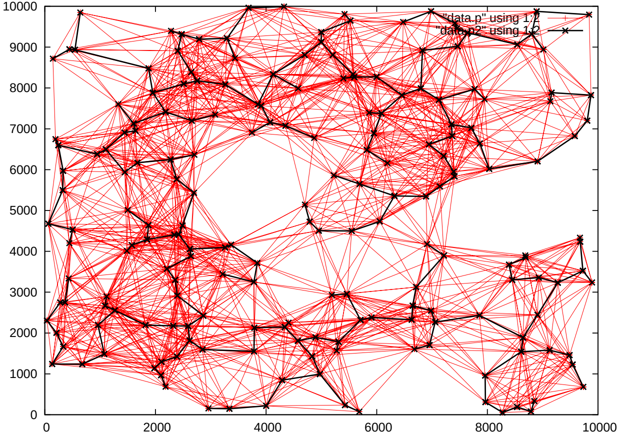
l number of leaves in the graph

d max degree of vertices in T

Test Instances

Geometric Instances (Euc)

200 vertices
0.1 edge density



Test Instances

Geometric (Euc) and Uniform (Unif) Instances

Instances	$\lvert V\rvert$	$\lvert E'\rvert$	w_{min}	w_{max}	$\rho(G)$	$\lvert V_{pre}\rvert$	$\lvert E'_{pre}\rvert$	$\rho(G_{pre})$	$\rho(M)$
Euc-200-0.1	200	1899	191	1999	0.105	199	729	0.047	0.038
Euc-200-0.5	200	9666	191	5100	0.496	200	2499	0.136	0.082
Euc-200-1	200	19701	191	13215	1	200	4846	0.254	0.125
Euc-400-0.1	400	7984	84	2000	0.105	400	2473	0.036	0.03
Euc-400-0.5	400	38948	84	5100	0.493	400	8899	0.117	0.065
Euc-400-1	400	79401	84	13408	1	400	17499	0.224	0.1
Euc-800-0.1	800	32880	61	2000	0.105	800	8455	0.029	0.025
Euc-800-0.5	800	158203	61	5100	0.498	800	31617	0.101	0.053
Euc-800-1	800	318801	61	13686	1	800	62038	0.197	0.078

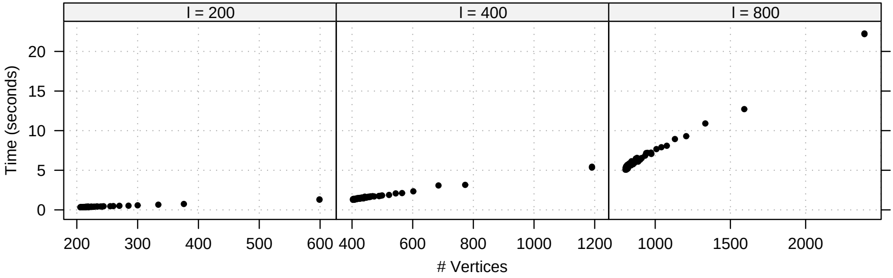
Test Instances

Geometric (Euc) and Uniform (Unif) Instances

Instances	$\lvert V\rvert$	$\lvert E'\rvert$	w_{min}	w_{max}	$\rho(G)$	$\lvert V_{pre}\rvert$	$\lvert E'_{pre}\rvert$	$\rho(G_{pre})$	$\rho(M)$
Euc-200-0.1	200	1899	191	1999	0.105	199	729	0.047	0.038
Euc-200-0.5	200	9666	191	5100	0.496	200	2499	0.136	0.082
Euc-200-1	200	19701	191	13215	1	200	4846	0.254	0.125
Euc-400-0.1	400	7984	84	2000	0.105	400	2473	0.036	0.03
Euc-400-0.5	400	38948	84	5100	0.493	400	8899	0.117	0.065
Euc-400-1	400	79401	84	13408	1	400	17499	0.224	0.1
Euc-800-0.1	800	32880	61	2000	0.105	800	8455	0.029	0.025
Euc-800-0.5	800	158203	61	5100	0.498	800	31617	0.101	0.053
Euc-800-1	800	318801	61	13686	1	800	62038	0.197	0.078
Unif-200-0.1	200	1781	589	9993	0.1	200	1138.5	0.067	0.074
Unif-200-0.5	200	9724	105	9998	0.499	200	3955	0.209	0.072
Unif-200-0.9	200	17683	59	9998	0.899	200	5899	0.306	0.071
Unif-400-0.1	400	7588	250	9998	0.1	400	4681	0.064	0.051
Unif-400-0.5	400	39479	49	9998	0.5	400	15814	0.203	0.047
Unif-400-0.9	400	71373	28	9999	0.899	400	22338	0.285	0.048
Unif-800-0.1	800	31237	109	9999	0.1	800	19495	0.063	0.032
Unif-800-0.5	800	159011	23	9999	0.5	800	63758	0.202	0.031
Unif-800-0.9	800	286806	13	9999	0.9	800	88973	0.281	0.033

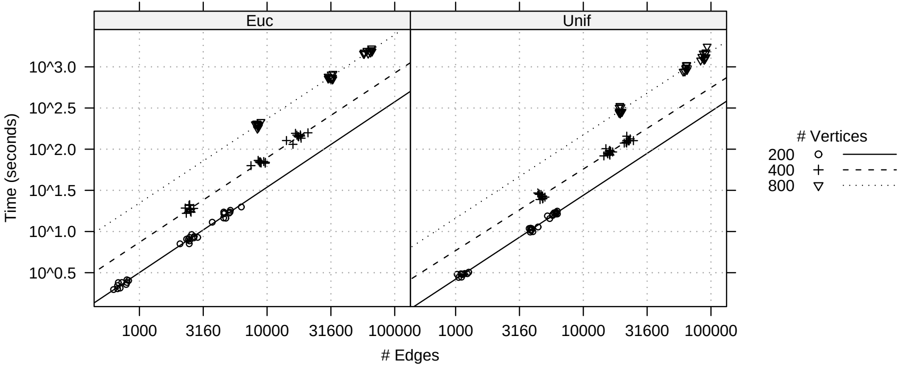
Exact Solution

Tree and Cycle Instances (T+C)



Exact Solution

Geometric (Euc) and Uniform (Unif) Instances



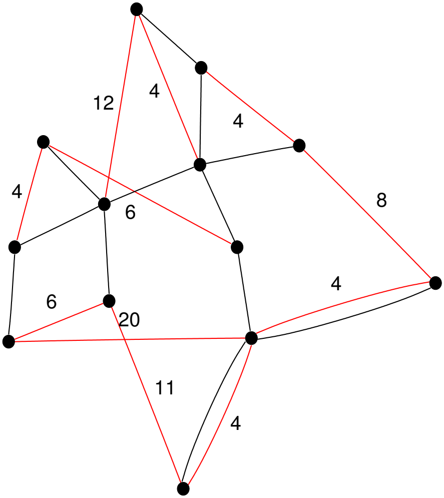
1 2-Edge-Connectivity Augmentation
- The Problem
- Test Instances

2 Basic Heuristics
- Construction Heuristics
- Local Search Algorithms
- Analysis

3 Advanced Heuristics
- Design
- Experimental Analysis

Random Addition

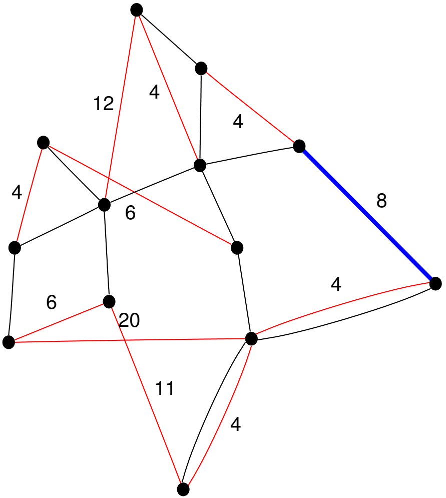
4 4 4 4 4 6 6 8 11 12 20



Function random_add(G, T)
$U := F$ (No edge in T is covered);
$E'' := E'$; $X := \emptyset$;
while $U \neq \emptyset$ **do**
    Choose a random edge $uv \in E''$
    Delete uv from E'';
    P_{uv} edge set of (uv)-path in T;
    if $P_{uv} \cap U \neq \emptyset$ **then**
        $X := X + e$;
        $U := U \setminus P_{uv}$;
$X' := \text{trim}(T, \emptyset, X)$;
return X'

Random Addition

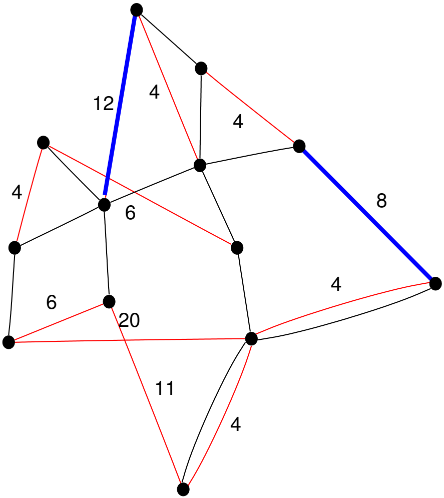
4 4 4 4 4 6 6 **8** 11 12 20



Function random_add(G, T)
$U := F$ (No edge in T is covered);
$E'' := E'$; $X := \emptyset$;
while $U \neq \emptyset$ **do**
  Choose a random edge $uv \in E''$
  Delete uv from E'';
  P_{uv} edge set of (uv)-path in T;
  if $P_{uv} \cap U \neq \emptyset$ **then**
    $X := X + e$;
    $U := U \setminus P_{uv}$;
$X' := \text{trim}(T, \emptyset, X)$;
return X'

Random Addition

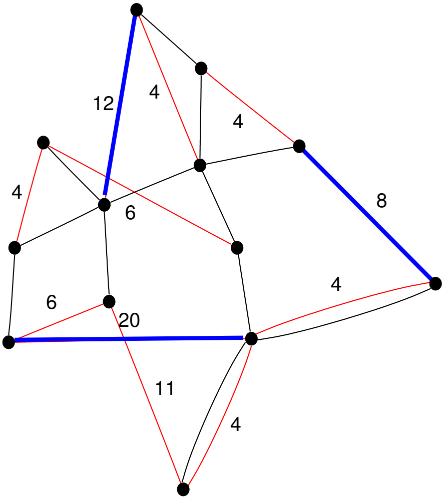
4 4 4 4 4 6 6 **8** 11 **12** 20



Function random_add(G, T)
$U := F$ (No edge in T is covered);
$E'' := E'$; $X := \emptyset$;
while $U \neq \emptyset$ **do**
  Choose a random edge $uv \in E''$
  Delete uv from E'';
  P_{uv} edge set of (uv)-path in T;
  if $P_{uv} \cap U \neq \emptyset$ **then**
    $X := X + e$;
    $U := U \setminus P_{uv}$;
$X' := \text{trim}(T, \emptyset, X)$;
return X'

Random Addition

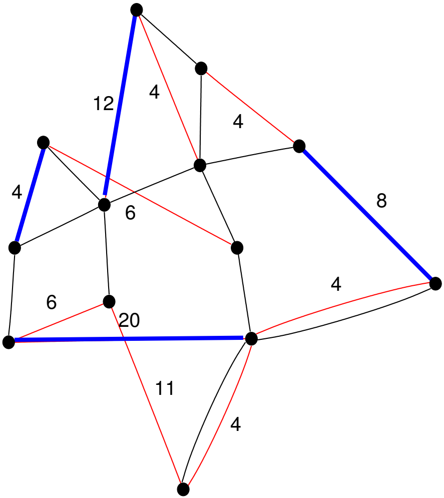
4 4 4 4 4 6 6 **8** 11 **12** **20**



Function random_add(G, T)
$U := F$ (No edge in T is covered);
$E'' := E'$; $X := \emptyset$;
while $U \neq \emptyset$ **do**
  Choose a random edge $uv \in E''$
  Delete uv from E'';
  P_{uv} edge set of (uv)-path in T;
  if $P_{uv} \cap U \neq \emptyset$ **then**
    $X := X + e$;
    $U := U \setminus P_{uv}$;
$X' := \text{trim}(T, \emptyset, X)$;
return X'

Random Addition

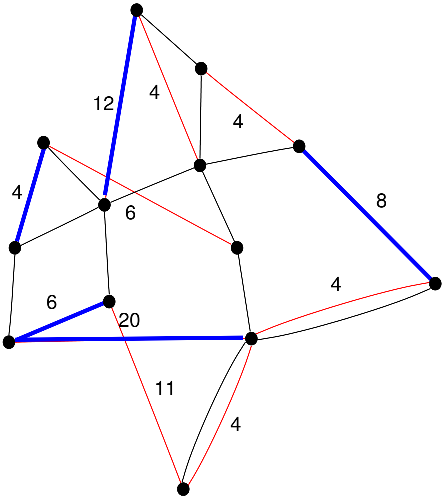
4 4 **4** 4 4 6 6 **8** 11 **12** **20**



Function random_add(G, T)
$U := F$ (No edge in T is covered);
$E'' := E'$; $X := \emptyset$;
while $U \neq \emptyset$ **do**
  Choose a random edge $uv \in E''$
  Delete uv from E'';
  P_{uv} edge set of (uv)-path in T;
  if $P_{uv} \cap U \neq \emptyset$ **then**
    $X := X + e$;
    $U := U \setminus P_{uv}$;
$X' := \text{trim}(T, \emptyset, X)$;
return X'

Random Addition

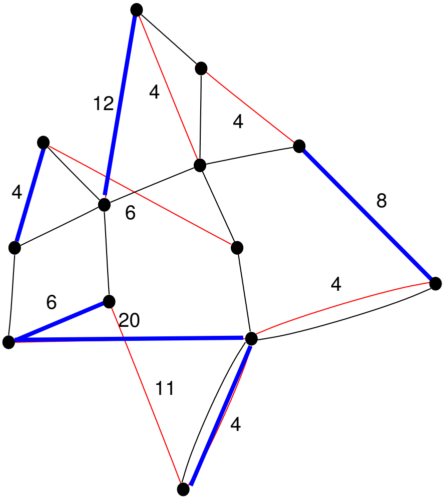
4 4 **4** 4 4 **6** 6 **8** 11 **12** **20**



Function random_add(G, T)
$U := F$ (No edge in T is covered);
$E'' := E'$; $X := \emptyset$;
while $U \neq \emptyset$ **do**
  Choose a random edge $uv \in E''$
  Delete uv from E'';
  P_{uv} edge set of (uv)-path in T;
  if $P_{uv} \cap U \neq \emptyset$ **then**
    $X := X + e$;
    $U := U \setminus P_{uv}$;
$X' := \text{trim}(T, \emptyset, X)$;
return X'

Random Addition

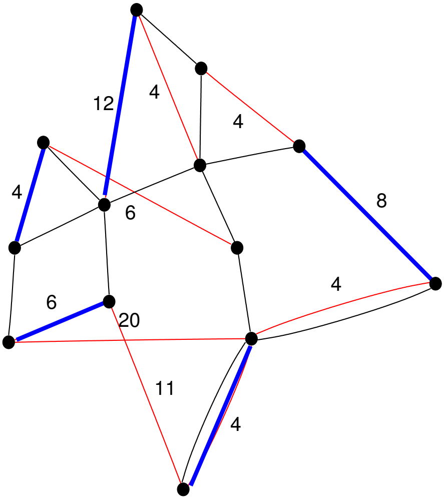
4 4 **4** 4 4 **6** 6 **8** 11 **12** **20**



Function random_add(G, T)
$U := F$ (No edge in T is covered);
$E'' := E'$; $X := \emptyset$;
while $U \neq \emptyset$ **do**
  Choose a random edge $uv \in E''$
  Delete uv from E'';
  P_{uv} edge set of (uv)-path in T;
  if $P_{uv} \cap U \neq \emptyset$ **then**
    $X := X + e$;
    $U := U \setminus P_{uv}$;
$X' := \text{trim}(T, \emptyset, X)$;
return X'

Random Addition

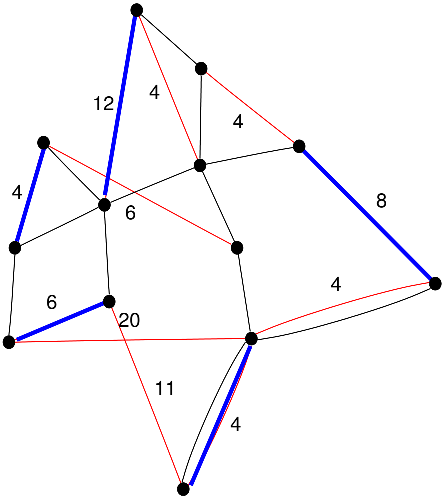
4 4 **4** 4 4 **6** 6 **8** 11 **12** **20**



Function random_add(G, T)
$U := F$ (No edge in T is covered);
$E'' := E'$; $X := \emptyset$;
while $U \neq \emptyset$ **do**
  Choose a random edge $uv \in E''$
  Delete uv from E'';
  P_{uv} edge set of (uv)-path in T;
  if $P_{uv} \cap U \neq \emptyset$ **then**
    $X := X + e$;
    $U := U \setminus P_{uv}$;
$X' := \text{trim}(T, \emptyset, X)$;
return X'

Random Addition

4 4 **4** 4 4 **6** 6 **8** 11 **12** **20** $\Rightarrow$ Cost: 44

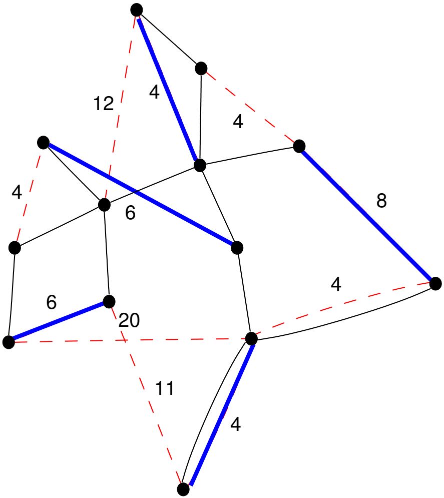


Function random_add(G, T)
$U := F$ (No edge in T is covered);
$E'' := E'$; $X := \emptyset$;
while $U \neq \emptyset$ **do**
  Choose a random edge $uv \in E''$
  Delete uv from E'';
  P_{uv} edge set of (uv)-path in T;
  if $P_{uv} \cap U \neq \emptyset$ **then**
    $X := X + e$;
    $U := U \setminus P_{uv}$;
$X' := \text{trim}(T, \emptyset, X)$;
return X'

$O(\min(|V|\,|E'|)\,|F|)$ time

Lightest Addition

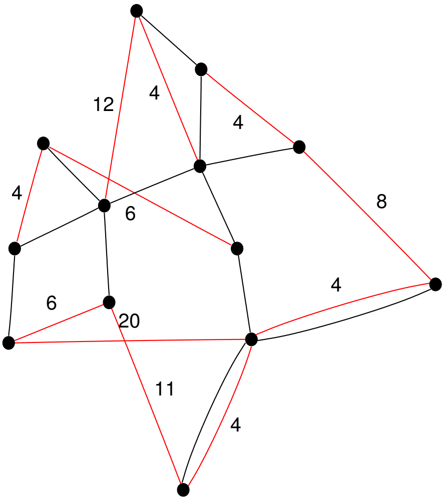
4 4 4 4 4 6 6 8 11 12 20 $\Rightarrow$ Cost: 28



Function lightest_add(G,T)
$U := F$ (No edge in T is covered);
$E'' := E'$; $X := \emptyset$;
while $U \neq \emptyset$ **do**
  Choose the cheapest $uv \in E''$
  Delete uv from E'';
  P_{uv} edge set of (uv)-path in T;
  if $P_{uv} \cap U \neq \emptyset$ **then**
    $X := X + e$;
    $U := U \setminus P_{uv}$;
$X' :=$ trim$(T, \emptyset, X)$;
return X'

Greedy Covering

4 4 4 4 4 6 6 8 11 12 20

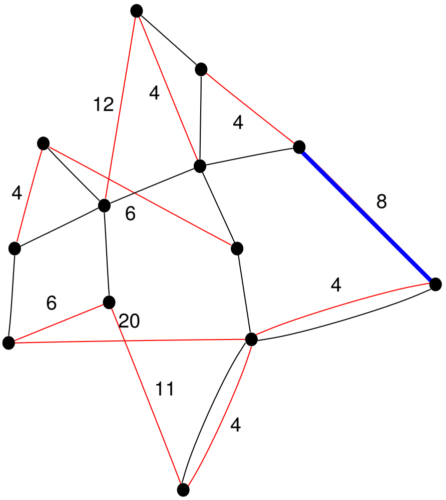


$\mathcal{C} = \{I_1, I_2, \ldots, I_m\}$

Function greedy_cov(M,$\mathcal{C}$, ω)
$Z := F$; $X := \emptyset$;
while $Z \neq \emptyset$ **do**
  Select $I_j : I_j \in \mathcal{C}$ s.t. $|I_j \cap Z| \neq \emptyset$ and I_j minimizes $\frac{\omega(I_j)}{|I_j \cap Z|}$;
  $Z := Z \setminus I_j$;
  $\mathcal{C} := \mathcal{C} \setminus \{I_j\}$;
  $X := X \cup \{j\}$;
$X :=$trim$(M, \emptyset, X)$;
return X.

Greedy Covering

4 4 4 4 4 6 6 **8** 11 12 20

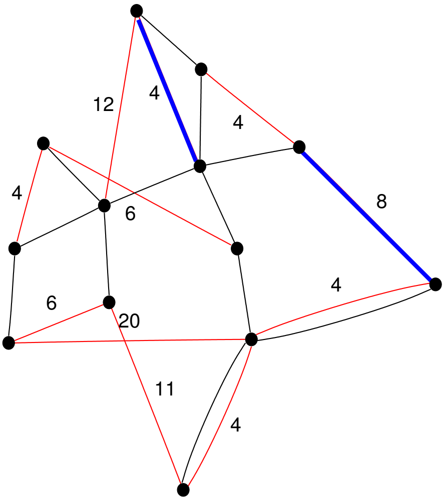


$\mathcal{C} = \{I_1, I_2, \ldots, I_m\}$

Function greedy_cov(M,$\mathcal{C}$, ω)
$Z := F$; $X := \emptyset$;
while $Z \neq \emptyset$ **do**
  Select $I_j : I_j \in \mathcal{C}$ s.t. $|I_j \cap Z| \neq \emptyset$ and I_j minimizes $\frac{\omega(I_j)}{|I_j \cap Z|}$;
  $Z := Z \setminus I_j$;
  $\mathcal{C} := \mathcal{C} \setminus \{I_j\}$;
  $X := X \cup \{j\}$;
$X :=$trim$(M, \emptyset, X)$;
return X.

Greedy Covering

4 4 4 4 4 6 6 **8** 11 12 20

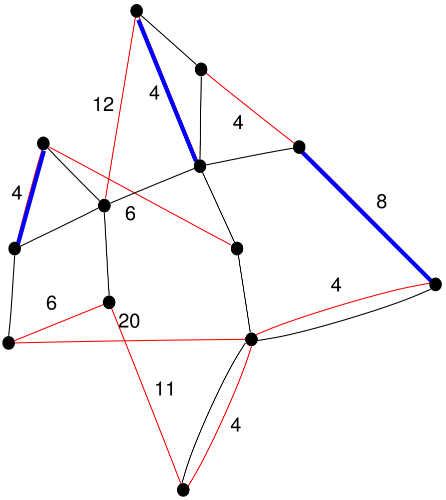


$\mathcal{C} = \{I_1, I_2, \ldots, I_m\}$

Function greedy_cov(M,$\mathcal{C}$, ω)
$Z := F$; $X := \emptyset$;
while $Z \neq \emptyset$ **do**
  Select $I_j : I_j \in \mathcal{C}$ s.t. $|I_j \cap Z| \neq \emptyset$
  and I_j minimizes $\frac{\omega(I_j)}{|I_j \cap Z|}$;
  $Z := Z \setminus I_j$;
  $\mathcal{C} := \mathcal{C} \setminus \{I_j\}$;
  $X := X \cup \{j\}$;
$X :=$trim$(M, \emptyset, X)$;
return X.

Greedy Covering

4 **4** 4 4 4 6 6 **8** 11 12 **20**

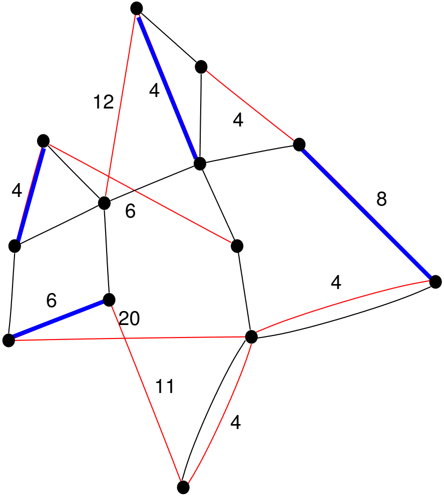


$\mathcal{C} = \{I_1, I_2, \ldots, I_m\}$

Function greedy_cov(M,$\mathcal{C}$, ω)
$Z := F$; $X := \emptyset$;
while $Z \neq \emptyset$ **do**
    Select $I_j : I_j \in \mathcal{C}$ s.t. $|I_j \cap Z| \neq \emptyset$ and I_j minimizes $\frac{\omega(I_j)}{|I_j \cap Z|}$;
    $Z := Z \setminus I_j$;
    $\mathcal{C} := \mathcal{C} \setminus \{I_j\}$;
    $X := X \cup \{j\}$;
$X :=$trim$(M, \emptyset, X)$;
return X.

Greedy Covering

4 **4** 4 4 4 **6** 6 **8** 11 12 **20**

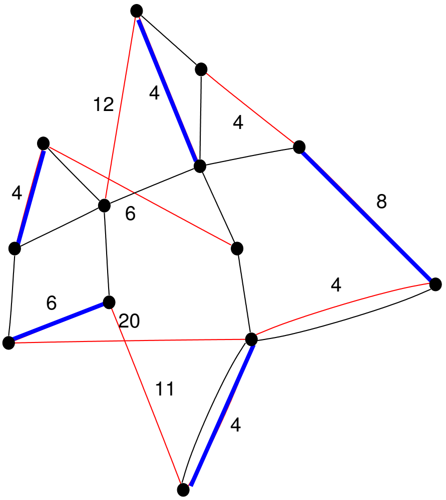


$\mathcal{C} = \{I_1, I_2, \dots, I_m\}$

Function greedy_cov(M,$\mathcal{C}$, ω)
$Z := F$; $X := \emptyset$;
while $Z \neq \emptyset$ **do**
  Select $I_j : I_j \in \mathcal{C}$ s.t. $|I_j \cap Z| \neq \emptyset$
  and I_j minimizes $\frac{\omega(I_j)}{|I_j \cap Z|}$;
  $Z := Z \setminus I_j$;
  $\mathcal{C} := \mathcal{C} \setminus \{I_j\}$;
  $X := X \cup \{j\}$;
$X :=$trim$(M, \emptyset, X)$;
return X.

Greedy Covering

4 **4** **4** 4 4 **6** 6 **8** 11 12 **20**

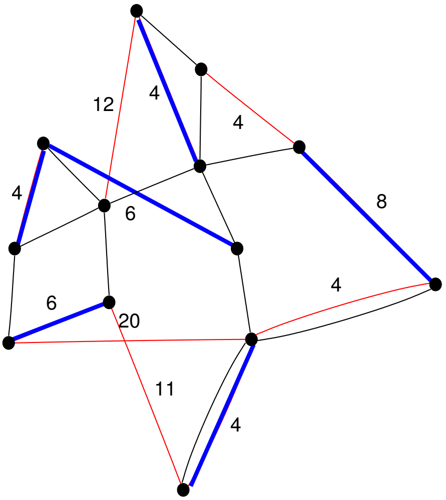


$\mathcal{C} = \{I_1, I_2, \ldots, I_m\}$

Function greedy_cov(M,$\mathcal{C}$, ω)
$Z := F$; $X := \emptyset$;
while $Z \neq \emptyset$ **do**
  Select $I_j : I_j \in \mathcal{C}$ s.t. $|I_j \cap Z| \neq \emptyset$
  and I_j minimizes $\frac{\omega(I_j)}{|I_j \cap Z|}$;
  $Z := Z \setminus I_j$;
  $\mathcal{C} := \mathcal{C} \setminus \{I_j\}$;
  $X := X \cup \{j\}$;
$X :=$trim$(M, \emptyset, X)$;
return X.

Greedy Covering

4 4 4 4 4 **6 6 8** 11 12 **20**



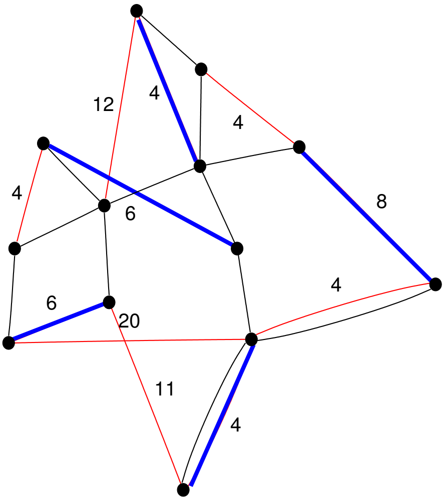
$\mathcal{C} = \{I_1, I_2, \ldots, I_m\}$

Function greedy_cov(M,$\mathcal{C}$, ω)
$Z := F$; $X := \emptyset$;
while $Z \neq \emptyset$ **do**
- Select $I_j : I_j \in \mathcal{C}$ s.t. $|I_j \cap Z| \neq \emptyset$ and I_j minimizes $\frac{\omega(I_j)}{|I_j \cap Z|}$;
- $Z := Z \setminus I_j$;
- $\mathcal{C} := \mathcal{C} \setminus \{I_j\}$;
- $X := X \cup \{j\}$;

$X :=$trim$(M, \emptyset, X)$;
return X.

Greedy Covering

4 4 4 4 4 **6 6 8** 11 12 **20**

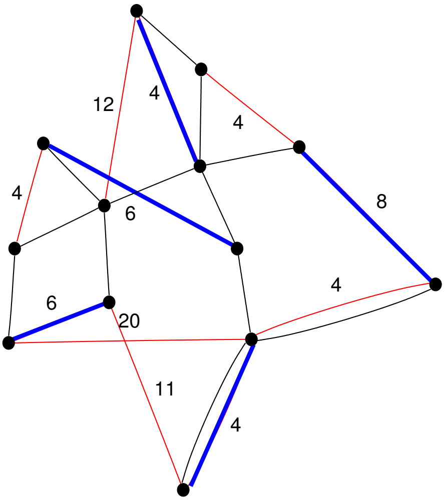


$\mathcal{C} = \{I_1, I_2, \ldots, I_m\}$

Function greedy_cov(M,$\mathcal{C}$, ω)
$Z := F$; $X := \emptyset$;
while $Z \neq \emptyset$ **do**
  Select $I_j : I_j \in \mathcal{C}$ s.t. $|I_j \cap Z| \neq \emptyset$ and I_j minimizes $\frac{\omega(I_j)}{|I_j \cap Z|}$;
  $Z := Z \setminus I_j$;
  $\mathcal{C} := \mathcal{C} \setminus \{I_j\}$;
  $X := X \cup \{j\}$;
$X :=$trim$(M, \emptyset, X)$;
return X.

Greedy Covering

4 4 4 4 4 **6 6 8** 11 12 **20** ⇒ Cost: 28

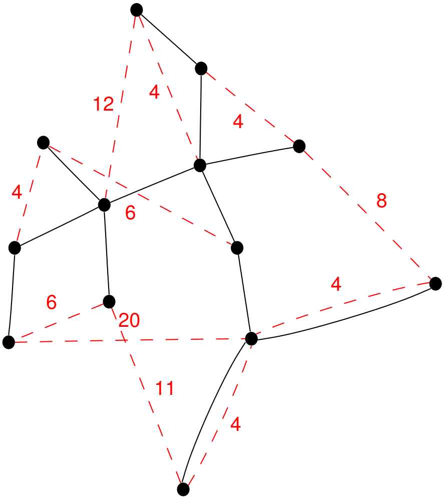


$\mathcal{C} = \{I_1, I_2, \ldots, I_m\}$

Function greedy_cov(M,$\mathcal{C}$, ω)
$Z := F$; $X := \emptyset$;
while $Z \neq \emptyset$ **do**
  Select $I_j : I_j \in \mathcal{C}$ s.t. $|I_j \cap Z| \neq \emptyset$
  and I_j minimizes $\frac{\omega(I_j)}{|I_j \cap Z|}$;
  $Z := Z \setminus I_j$;
  $\mathcal{C} := \mathcal{C} \setminus \{I_j\}$;
  $X := X \cup \{j\}$;
$X :=$trim$(M, \emptyset, X)$;
return X.

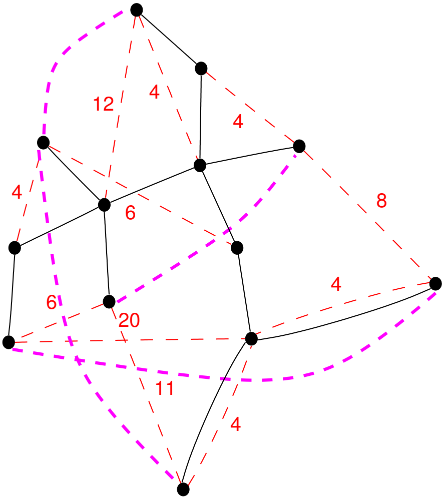
- $O(\min(|V|\,|E'|)\,|V|\,|E'|)$ time
- $O(\ln|V| + 1)$-approximation

Shortest Path



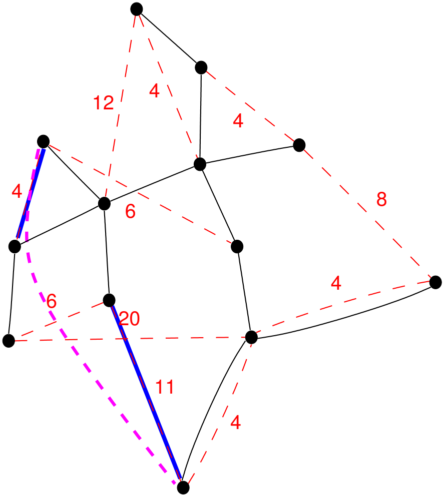
Function shortest_path(G,T)
$\{u_1v_1, u_2v_2, \ldots, u_pv_p\}$ set of connections returned by pair(T);
$Z := F$; $X := \emptyset$;
for $i = 1$ to p **do**
    Digraph D_i;
    P shortest path $u_i \rightarrow v_i$ in D_i;
    $Y := (E(P) \cap E')$;
    $X := X \cup Y$;
    $C(P)$ edges in Z covered by Y;
    $Z := Z \setminus C(P)$;
X':=trim(T,$\emptyset$,X);
return X'

Shortest Path



Function shortest_path(G,T)
$\{u_1v_1, u_2v_2, \ldots, u_pv_p\}$ set of connections returned by pair(T);
$Z := F$; $X := \emptyset$;
for $i = 1$ to p **do**
  Digraph D_i;
  P shortest path $u_i \to v_i$ in D_i;
  $Y := (E(P) \cap E')$;
  $X := X \cup Y$;
  $C(P)$ edges in Z covered by Y;
  $Z := Z \setminus C(P)$;
X':=trim$(T,\emptyset,X)$;
return X'

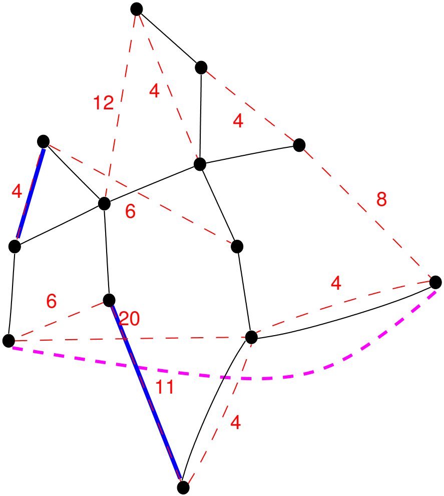
Shortest Path



Function shortest_path(G,T)
$\{u_1v_1, u_2v_2, \ldots, u_pv_p\}$ set of connections returned by pair(T);
$Z := F$; $X := \emptyset$;
for $i = 1$ to p **do**
  Digraph D_i;
  P shortest path $u_i \rightarrow v_i$ in D_i;
  $Y := (E(P) \cap E')$;
  $X := X \cup Y$;
  $C(P)$ edges in Z covered by Y;
  $Z := Z \setminus C(P)$;
X':=trim(T,$\emptyset$,X);
return X'

(i) edges of $P_{u_i v_i} \cap Z$: towards u_i
(ii) other edges: directed 2-cycle;
(iii) costs: edges in X have cost zero; edges in E' have original costs

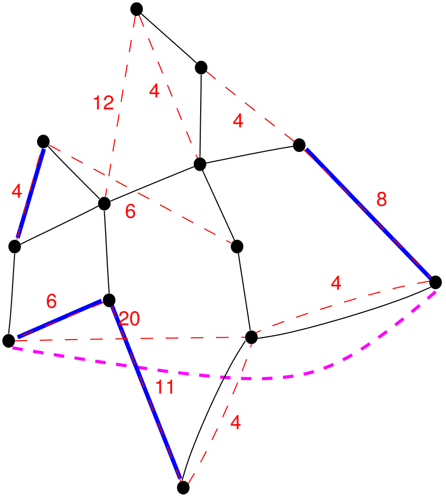
Shortest Path



Function shortest_path(G,T)
$\{u_1v_1, u_2v_2, \ldots, u_pv_p\}$ set of connections returned by pair(T);
$Z := F$; $X := \emptyset$;
for $i = 1$ to p **do**
  Digraph D_i;
  P shortest path $u_i \to v_i$ in D_i;
  $Y := (E(P) \cap E')$;
  $X := X \cup Y$;
  $C(P)$ edges in Z covered by Y;
  $Z := Z \setminus C(P)$;
X':=trim(T,$\emptyset$,X);
return X'

(i) edges of $P_{u_iv_i} \cap Z$: towards u_i
(ii) other edges: directed 2-cycle;
(iii) costs: edges in X have cost zero; edges in E' have original costs

Shortest Path

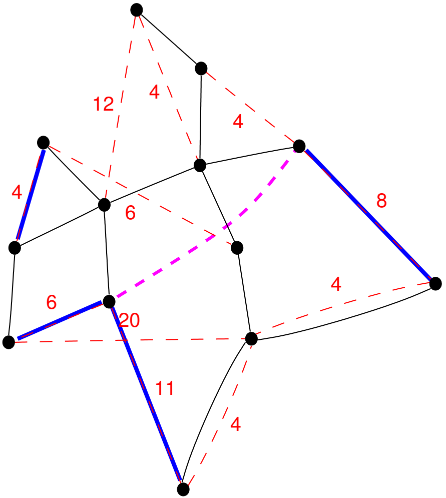


Function shortest_path(G,T)
$\{u_1v_1, u_2v_2, \ldots, u_pv_p\}$ set of connections returned by pair(T);
$Z := F$; $X := \emptyset$;
for $i = 1$ to p **do**
- Digraph D_i;
- P shortest path $u_i \to v_i$ in D_i;
- $Y := (E(P) \cap E')$;
- $X := X \cup Y$;
- $C(P)$ edges in Z covered by Y;
- $Z := Z \setminus C(P)$;

X':=trim(T,$\emptyset$,X);
return X'

(i) edges of $P_{u_i v_i} \cap Z$: towards u_i
(ii) other edges: directed 2-cycle;
(iii) costs: edges in X have cost zero; edges in E' have original costs

Shortest Path

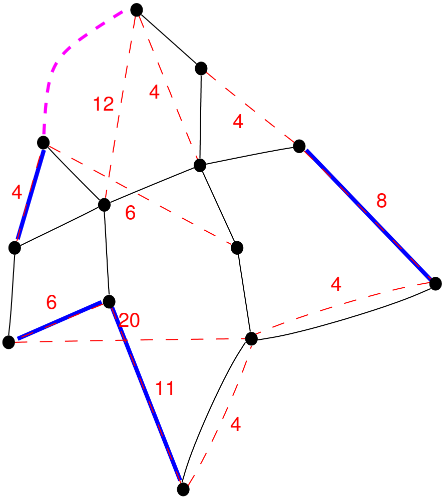


Function shortest_path(G,T)
$\{u_1v_1, u_2v_2, \ldots, u_pv_p\}$ set of connections returned by pair(T);
$Z := F$; $X := \emptyset$;
for $i = 1$ to p **do**
- Digraph D_i;
- P shortest path $u_i \rightarrow v_i$ in D_i;
- $Y := (E(P) \cap E')$;
- $X := X \cup Y$;
- $C(P)$ edges in Z covered by Y;
- $Z := Z \setminus C(P)$;

X':=trim(T,$\emptyset$,X);
return X'

(i) edges of $P_{u_iv_i} \cap Z$: towards u_i
(ii) other edges: directed 2-cycle;
(iii) costs: edges in X have cost zero; edges in E' have original costs

Shortest Path

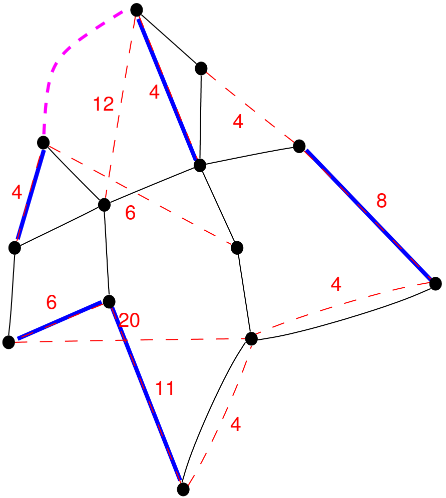


Function shortest_path(G,T)
$\{u_1v_1, u_2v_2, \ldots, u_pv_p\}$ set of connections returned by pair(T);
$Z := F$; $X := \emptyset$;
for $i = 1$ to p **do**
- Digraph D_i;
- P shortest path $u_i \to v_i$ in D_i;
- $Y := (E(P) \cap E')$;
- $X := X \cup Y$;
- $C(P)$ edges in Z covered by Y;
- $Z := Z \setminus C(P)$;

X':=trim(T,$\emptyset$,X);
return X'

(i) edges of $P_{u_i v_i} \cap Z$: towards u_i
(ii) other edges: directed 2-cycle;
(iii) costs: edges in X have cost zero; edges in E' have original costs

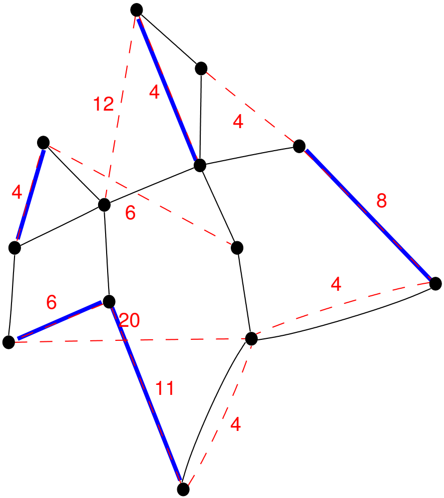
Shortest Path



Function shortest_path(G,T)
$\{u_1v_1, u_2v_2, \ldots, u_pv_p\}$ set of connections returned by pair(T);
$Z := F$; $X := \emptyset$;
for $i = 1$ to p **do**
  Digraph D_i;
  P shortest path $u_i \to v_i$ in D_i;
  $Y := (E(P) \cap E')$;
  $X := X \cup Y$;
  $C(P)$ edges in Z covered by Y;
  $Z := Z \setminus C(P)$;
X':=trim(T,$\emptyset$,X);
return X'

(i) edges of $P_{u_iv_i} \cap Z$: towards u_i
(ii) other edges: directed 2-cycle;
(iii) costs: edges in X have cost zero; edges in E' have original costs

Shortest Path

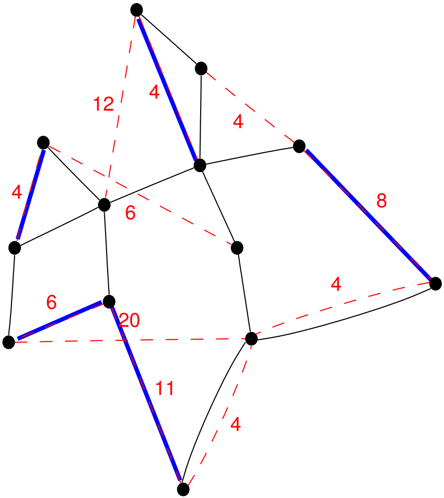


Function shortest_path(G,T)
$\{u_1v_1, u_2v_2, \ldots, u_pv_p\}$ set of connections returned by pair(T);
$Z := F$; $X := \emptyset$;
for $i = 1$ to p **do**
  Digraph D_i;
  P shortest path $u_i \to v_i$ in D_i;
  $Y := (E(P) \cap E')$;
  $X := X \cup Y$;
  $C(P)$ edges in Z covered by Y;
  $Z := Z \setminus C(P)$;
X':=trim(T,$\emptyset$,X);
return X'

(i) edges of $P_{u_iv_i} \cap Z$: towards u_i
(ii) other edges: directed 2-cycle;
(iii) costs: edges in X have cost zero; edges in E' have original costs

Shortest Path

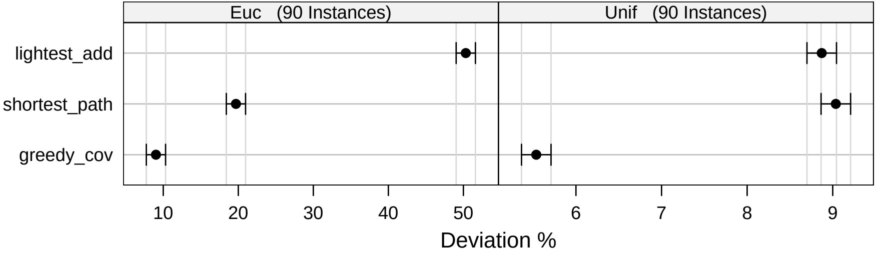
⇒ Cost: 33 $O(\min(|V|\,|E'|)\,|V|^2)$ time



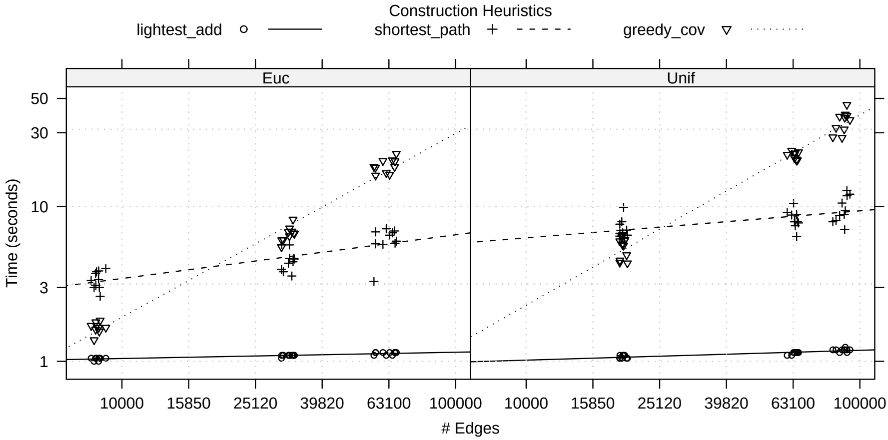
Function shortest_path(G,T)
$\{u_1v_1, u_2v_2, \ldots, u_pv_p\}$ set of connections returned by pair(T);
$Z := F$; $X := \emptyset$;
for $i = 1$ to p **do**
  Digraph D_i;
  P shortest path $u_i \to v_i$ in D_i;
  $Y := (E(P) \cap E')$;
  $X := X \cup Y$;
  $C(P)$ edges in Z covered by Y;
  $Z := Z \setminus C(P)$;
X':=trim(T,$\emptyset$,X);
return X'

(i) edges of $P_{u_iv_i} \cap Z$: towards u_i
(ii) other edges: directed 2-cycle;
(iii) costs: edges in X have cost zero; edges in E' have original costs

Comparison based on quality of approximation



Comparison based on computation time



Local Search Algorithms

Candidate solutions: any set X that is a proper augmentation

Local Search Algorithms

Candidate solutions: any set X that is a proper augmentation

Neighborhood Structure:

Local Search Algorithms

Candidate solutions: any set X that is a proper augmentation

Neighborhood Structure:
Exchange neighborhoods are not good because the problem is over-constrained.

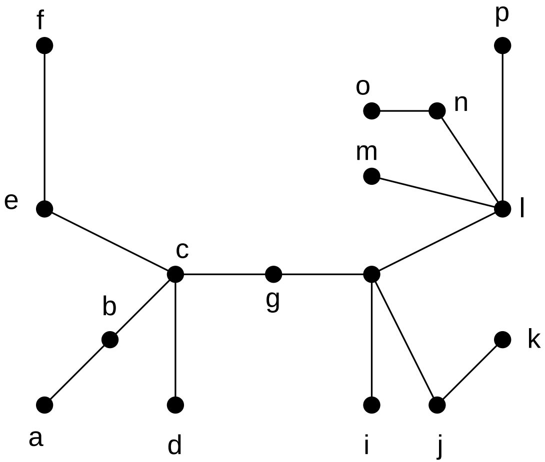
Three local search neighborhoods exploiting:

- the set covering formulation
  - **k-cov:** ruin and repair neighborhood
  - **k-add:** addition neighborhood
- the graph structure
  - **k-sp:** shortest path based neighborhood (very large-scale neighborhood)

Shortest Path Neighborhood

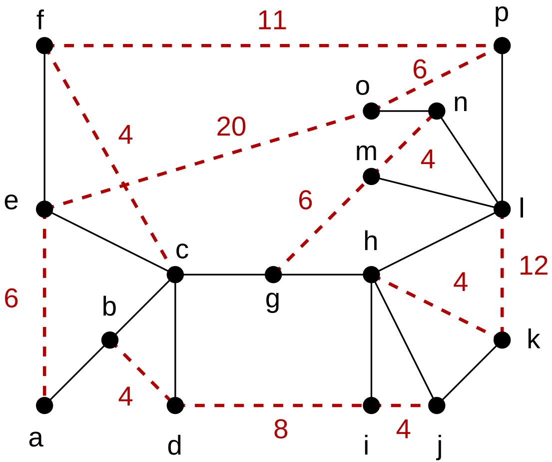
Shortest path reconstruction

k-sp:



Shortest path reconstruction

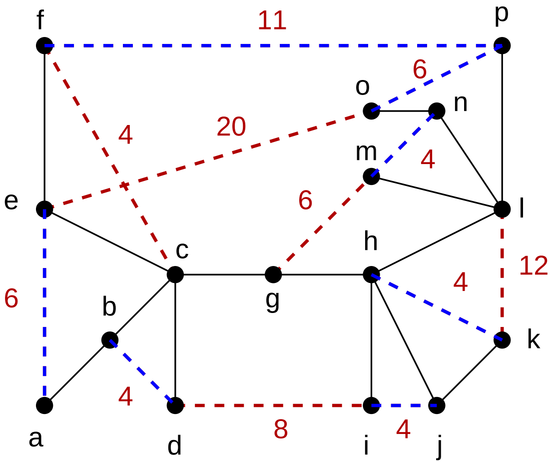
k-sp:



Shortest Path Neighborhood

Shortest path reconstruction

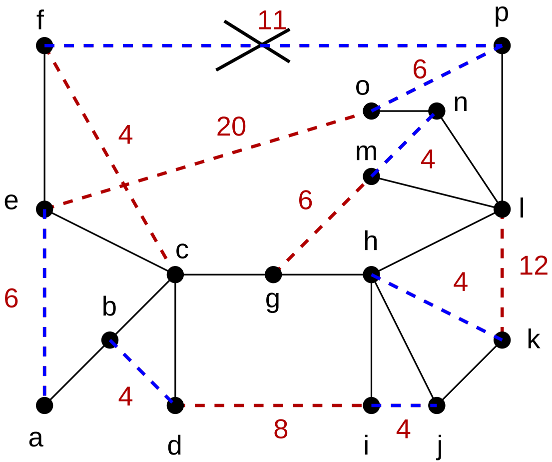
k-sp:



Shortest Path Neighborhood

Shortest path reconstruction

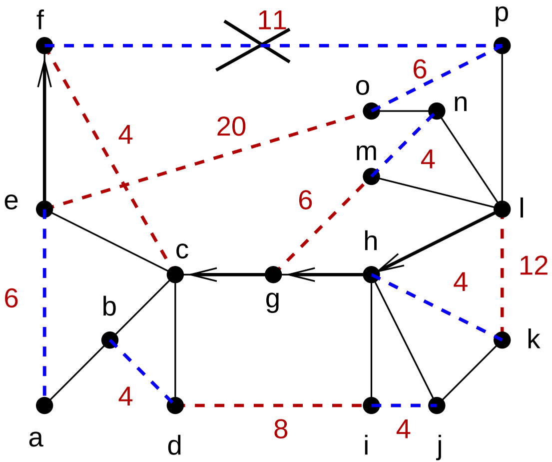
k-sp:



Shortest Path Neighborhood

Shortest path reconstruction

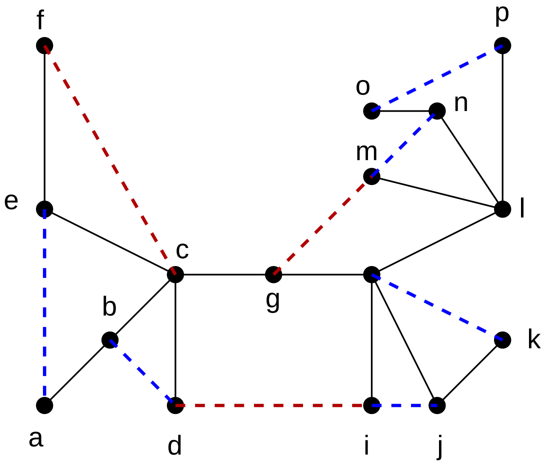
k-sp:



Shortest Path Neighborhood

Shortest path reconstruction

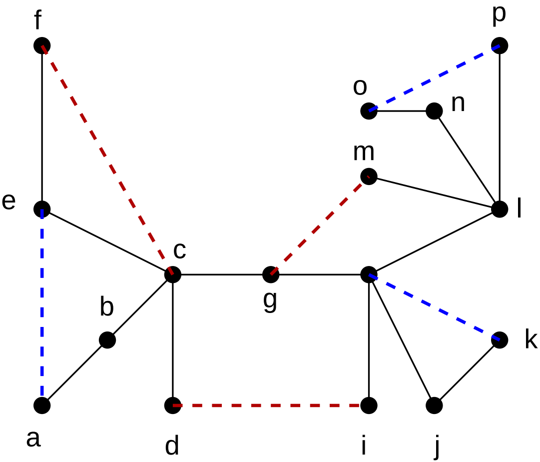
k-sp:



Shortest Path Neighborhood

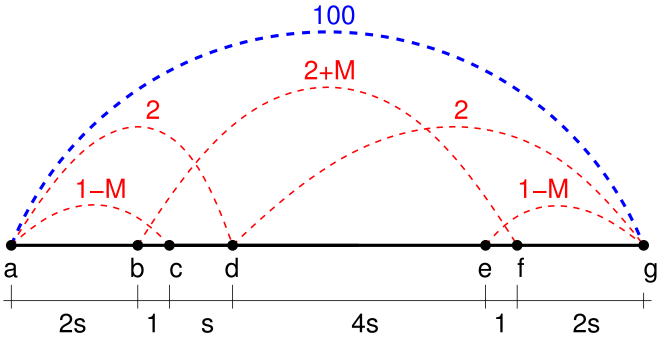
Shortest path reconstruction

k-sp:



Neighborhoods

$M \in \mathbb{R}_0^+ \;\; s \in \mathbb{N}_0$



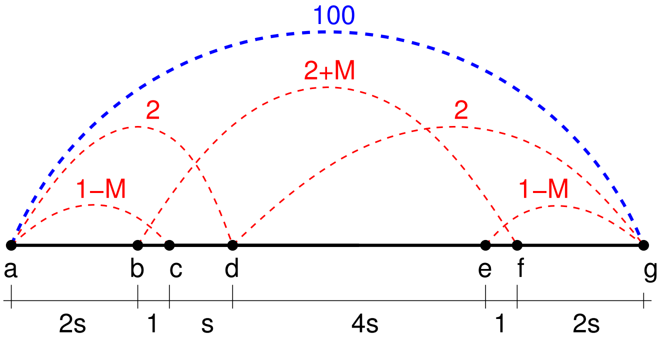
1-add ⇒ no improvement.

1-cov ⇒ remove ag and introduce dg and ad ⇒ new cost: 4

1-shp ⇒ remove ag and introduce ac, bf and eg ⇒ new cost: $4 - M$

Neighborhoods

$M \in \mathbb{R}_0^+ \; s \in \mathbb{N}_0$



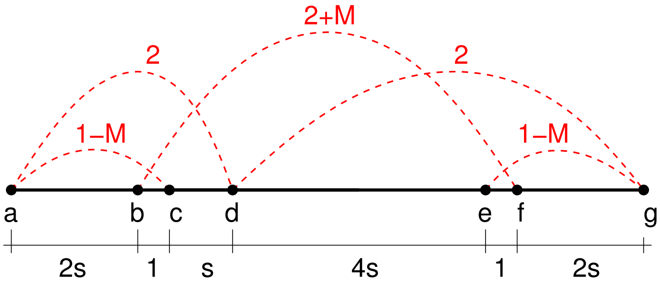
`1-add` ⇒ no improvement.

`1-cov` ⇒ remove ag and introduce dg and ad ⇒ new cost: 4

`1-shp` ⇒ remove ag and introduce ac, bf and eg ⇒ new cost: $4 - M$

Neighborhoods

$M \in \mathbb{R}_0^+ \; s \in \mathbb{N}_0$



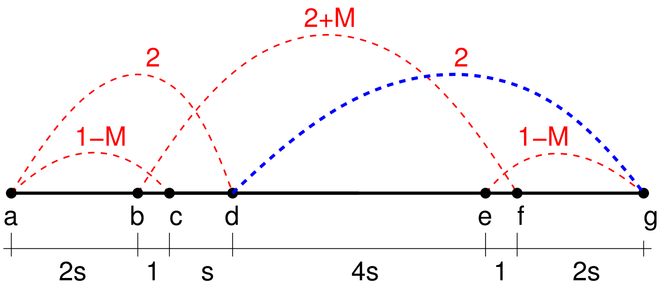
1-add $\Rightarrow$ no improvement.

1-cov $\Rightarrow$ remove ag and introduce dg and ad $\Rightarrow$ new cost: 4

1-shp $\Rightarrow$ remove ag and introduce ac, bf and eg $\Rightarrow$ new cost: $4 - M$

Neighborhoods

$M \in \mathbb{R}_0^+ \; s \in \mathbb{N}_0$



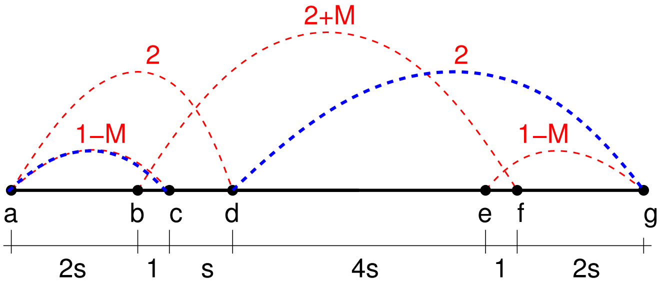
1-add ⇒ no improvement.

1-cov ⇒ remove ag and introduce dg and ad ⇒ new cost: 4

1-shp ⇒ remove ag and introduce ac, bf and eg ⇒ new cost: $4 - M$

Neighborhoods

$M \in \mathbb{R}_0^+ \; s \in \mathbb{N}_0$



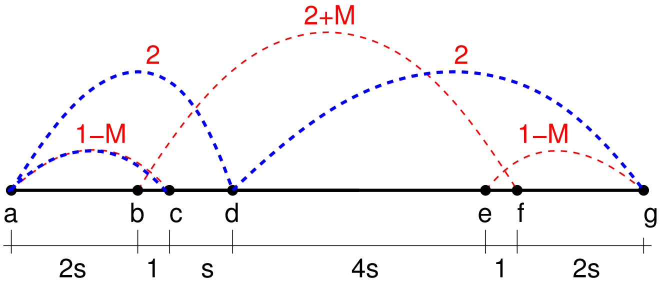
1-add $\Rightarrow$ no improvement.

1-cov $\Rightarrow$ remove ag and introduce dg and ad $\Rightarrow$ new cost: 4

1-shp $\Rightarrow$ remove ag and introduce ac, bf and eg $\Rightarrow$ new cost: $4 - M$

Neighborhoods

$M \in \mathbb{R}_0^+ \quad s \in \mathbb{N}_0$



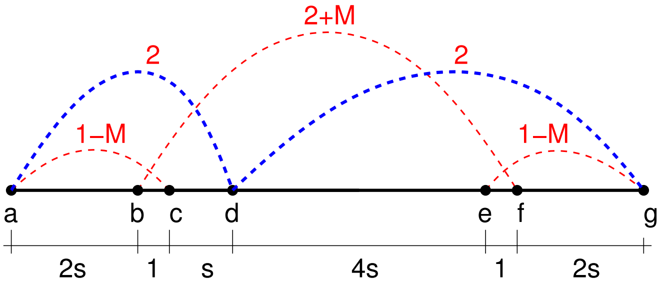
1-add $\Rightarrow$ no improvement.

1-cov $\Rightarrow$ remove ag and introduce dg and ad $\Rightarrow$ new cost: 4

1-shp $\Rightarrow$ remove ag and introduce ac, bf and eg $\Rightarrow$ new cost: $4 - M$

Neighborhoods

$M \in \mathbb{R}_0^+ \; s \in \mathbb{N}_0$



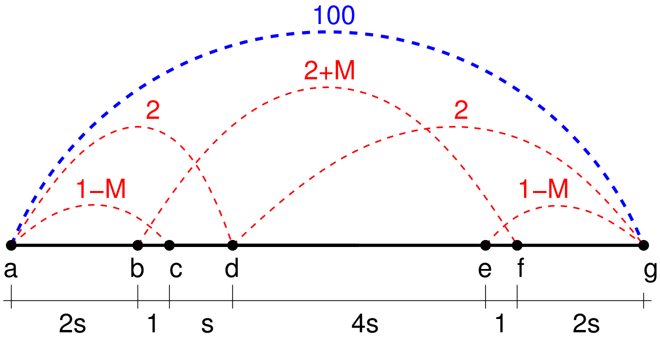
1-add $\Rightarrow$ no improvement.

1-cov $\Rightarrow$ remove ag and introduce dg and ad $\Rightarrow$ new cost: 4

1-shp $\Rightarrow$ remove ag and introduce ac, bf and eg $\Rightarrow$ new cost: $4 - M$

Neighborhoods

$M \in \mathbb{R}_0^+ \ s \in \mathbb{N}_0$



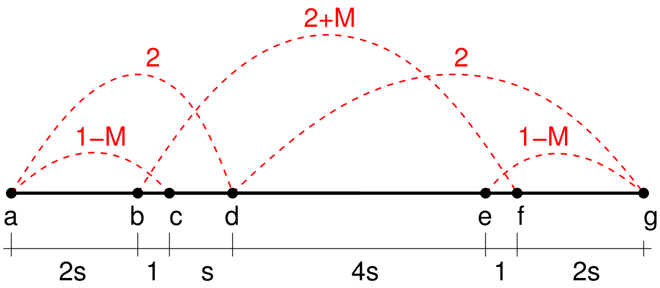
1-add $\Rightarrow$ no improvement.

1-cov $\Rightarrow$ remove ag and introduce dg and ad $\Rightarrow$ new cost: 4

1-shp $\Rightarrow$ remove ag and introduce ac, bf and eg $\Rightarrow$ new cost: $4 - M$

Neighborhoods

$M \in \mathbb{R}_0^+ \; s \in \mathbb{N}_0$



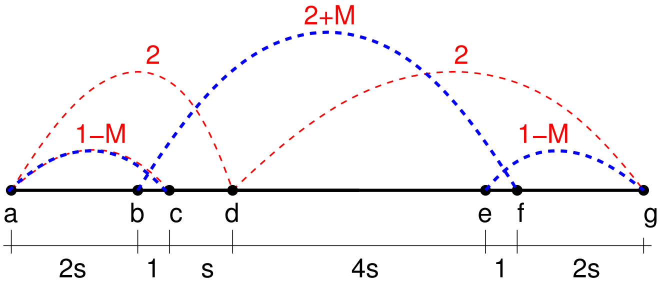
1-add $\Rightarrow$ no improvement.

1-cov $\Rightarrow$ remove ag and introduce dg and ad $\Rightarrow$ new cost: 4

1-shp $\Rightarrow$ remove ag and introduce ac, bf and eg $\Rightarrow$ new cost: $4 - M$

Neighborhoods

$M \in \mathbb{R}_0^+ \; s \in \mathbb{N}_0$



1-add $\Rightarrow$ no improvement.

1-cov $\Rightarrow$ remove ag and introduce dg and ad $\Rightarrow$ new cost: 4

1-shp $\Rightarrow$ remove ag and introduce ac, bf and eg $\Rightarrow$ new cost: $4 - M$

Experimental Design

- **3+3 factors**:

Initial Solution:	{`greedy-cov`, `shortest-path`, `lightest-add`}
Neighborhood Type:	{`k-add`, `k-cov`, `k-sp`}
k:	$\{1, 3, 5\}$
Size:	$\{200, 400, 800\}$
Graph type:	{Geometric , Uniform}
Edge density:	$\{0.1, 0.5, 0.9\}$

- **Response**:

Quality:	percentage deviation from optimal solution
Run-time:	time to local optimum

- **Data collected**: one run per algorithmic configuration on five single instances from the 18 instance classes.

Mixed-Nested Design Analysis

The Mixed Procedure

Cov Parm	Estimate	Error	Value	Pr Z
inst	3.7623	0.7095	5.30	<.0001
Residual	19.5141	0.5767	33.84	<.0001

Type 3 Tests of Fixed Effects

Effect	DF	DF	F Value	Pr > F
initial	2	2290	323.25	<.0001
neighborhood	2	2290	105.18	<.0001
l	2	2290	42.72	<.0001
type	1	80	105.65	<.0001
size	1	80	1.04	0.3105
dens	2	80	0.37	0.6948
initial*neighborhood	4	2290	50.22	<.0001
initial*l	4	2290	61.86	<.0001
initial*type	2	2290	1248.31	<.0001
size*initial	2	2290	6.86	0.0011
initial*dens	4	2290	2.22	0.0645

...

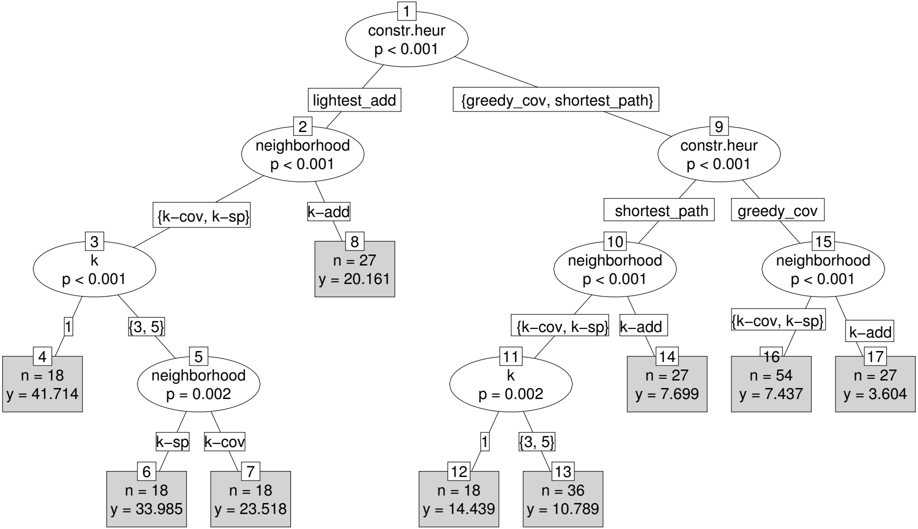
Least Squares Means

algo	Estimate	Std.Error	DF	t Value	Pr > |t|
greedy_cov.l-add.1	3.1247	0.5086	1336	6.14	<.0001
greedy_cov.l-add.3	3.2907	0.5086	1336	6.47	<.0001
greedy_cov.l-add.5	3.4624	0.5086	1336	6.81	<.0001
greedy_cov.l-cov.1	6.4922	0.5086	1336	12.77	<.0001
greedy_cov.l-cov.3	6.3530	0.5086	1336	12.49	<.0001
greedy_cov.l-cov.5	6.2631	0.5086	1336	12.32	<.0001

...

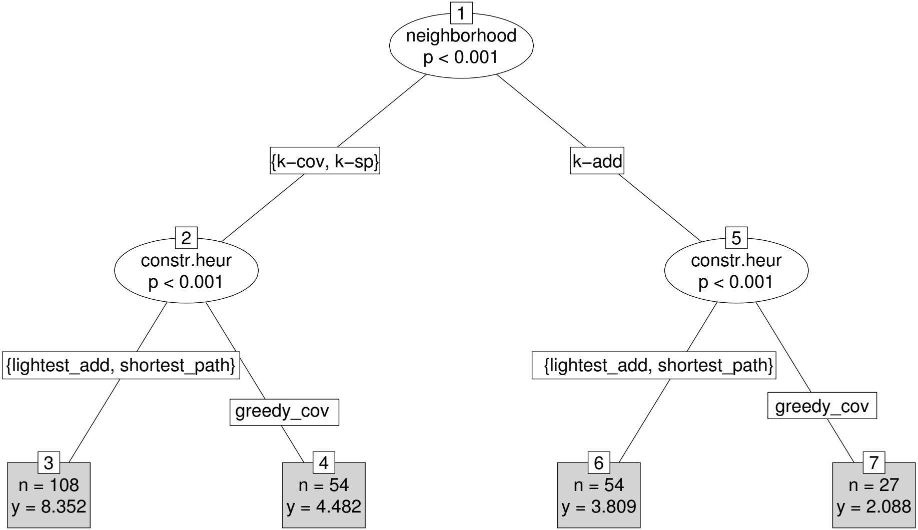
Regression Tree

Euclidean instances



Regression Tree

Uniform instances



Resume

- **Initial Solution:** is not very important for the quality after the local search
- **Neighborhood Type:** the set covering approach (ruin and repair) is the best.
- $\mathbf{k} = \{\mathbf{1,3,5}\}$: the three local search algorithms behave differntly.
  For $\mathrm{k\text{-}cov}$, $\mathrm{k} = 1$ is enough.

1. 2-Edge-Connectivity Augmentation
   - The Problem
   - Test Instances
2. Basic Heuristics
   - Construction Heuristics
   - Local Search Algorithms
   - Analysis
3. Advanced Heuristics
   - Design
   - Experimental Analysis

A Hybrid Heuristic

We developed an hybrid heuristic assembling together ideas from known set covering solvers:

A Hybrid Heuristic

We developed an hybrid heuristic assembling together ideas from known set covering solvers:

1. Lagrangian relaxation with subgradient optimization
   [Caprara, Fischetti and Toth, 1999]

A Hybrid Heuristic

We developed an hybrid heuristic assembling together ideas from known set covering solvers:

1. Lagrangian relaxation with subgradient optimization [Caprara, Fischetti and Toth, 1999]

$$z_{SCP} = \min\{\omega^T x : Mx \geq 1, x \in \{0,1\}^m\}$$

A Hybrid Heuristic

We developed an hybrid heuristic assembling together ideas from known set covering solvers:

1. Lagrangian relaxation with subgradient optimization
[Caprara, Fischetti and Toth, 1999]

$$z_{SCP} = \min\{\omega^T x : Mx \geq 1, x \in \{0,1\}^m\}$$

$$z_{LR}(u) = \min_{x \in \{0,1\}^m} \sum_{j \in E'} c_j(u) x_j + \sum_{i \in F} u_i$$

$$c_j(u) = w_j - \sum_{i \in F} M_{ij} u_i$$

A Hybrid Heuristic

We developed an hybrid heuristic assembling together ideas from known set covering solvers:

1. Lagrangian relaxation with subgradient optimization [Caprara, Fischetti and Toth, 1999]

$$z_{SCP} = \min\{\omega^T x : Mx \geq 1, x \in \{0,1\}^m\}$$

$$z_{LR}(u) = \min_{x \in \{0,1\}^m} \sum_{j \in E'} c_j(u) x_j + \sum_{i \in F} u_i$$

$$c_j(u) = w_j - \sum_{i \in F} M_{ij} u_i$$

$$u_i^{k+1} = \max\left\{u_i^k + \frac{\lambda^k (UB_{LD} - z_{LD})}{\|s(u^k)\|} s(u^k), 0\right\}$$

A Hybrid Heuristic

We developed an hybrid heuristic assembling together ideas from known set covering solvers:

1. Lagrangian relaxation with subgradient optimization [Caprara, Fischetti and Toth, 1999]

$$z_{SCP} = \min\{\omega^T x : Mx \geq 1, x \in \{0,1\}^m\}$$

$$z_{LR}(u) = \min_{x \in \{0,1\}^m} \sum_{j \in E'} c_j(u) x_j + \sum_{i \in F} u_i$$

$$c_j(u) = w_j - \sum_{i \in F} M_{ij} u_i$$

$$u_i^{k+1} = \max\left\{ u_i^k + \frac{\lambda^k (UB_{LD} - z_{LD})}{\|s(u^k)\|} s(u^k), 0 \right\}$$

2. Pricing scheme [Caprara, Fischetti and Toth, 1999]

A Hybrid Heuristic

We developed an hybrid heuristic assembling together ideas from known set covering solvers:

1. Lagrangian relaxation with subgradient optimization [Caprara, Fischetti and Toth, 1999]

$$z_{SCP} = \min\{\omega^T x : Mx \geq 1, x \in \{0,1\}^m\}$$

$$z_{LR}(u) = \min_{x \in \{0,1\}^m} \sum_{j \in E'} c_j(u) x_j + \sum_{i \in F} u_i$$

$$c_j(u) = w_j - \sum_{i \in F} M_{ij} u_i$$

$$u_i^{k+1} = \max\left\{u_i^k + \frac{\lambda^k (UB_{LD} - z_{LD})}{\|s(u^k)\|} s(u^k), 0\right\}$$

2. Pricing scheme [Caprara, Fischetti and Toth, 1999]
3. Iterated Greedy [Marchiori, Steenback, 2000]

A Hybrid Heuristic

We developed an hybrid heuristic assembling together ideas from known set covering solvers:

1. Lagrangian relaxation with subgradient optimization [Caprara, Fischetti and Toth, 1999]

$$z_{SCP} = \min\{\omega^T x : Mx \geq 1, x \in \{0,1\}^m\}$$

$$z_{LR}(u) = \min_{x \in \{0,1\}^m} \sum_{j \in E'} c_j(u) x_j + \sum_{i \in F} u_i$$

$$c_j(u) = w_j - \sum_{i \in F} M_{ij} u_i$$

$$u_i^{k+1} = \max\left\{ u_i^k + \frac{\lambda^k (UB_{LD} - z_{LD})}{\|s(u^k)\|} s(u^k), 0 \right\}$$

2. Pricing scheme [Caprara, Fischetti and Toth, 1999]
3. Iterated Greedy [Marchiori, Steenback, 2000]
4. Iterated Local Search

Lagrangian Multi-Start Heuristic

Function LMS (M, ω)
$u_i^0 = \min_{j \in J_i} w_j / |I_j|$ for all $i \in F$;
$M^{core} :=$ pricing (M, ω, u^0);
$X :=$ greedy_cov (M^{core}, ω); % state $:= 1$
$(X, u^*) :=$ subgradient_phase (M^{core}, u_0); % state $:= 2$
$X :=$ local_optimization (X); % state $:= 3$
$M^{core} :=$ pricing (M, ω, u^*); % state $:= 4$
$(X, u^*) :=$ subgradient_phase (M^{core}, ω, u^*); % state $:= 5$
$j := 0$; improved $:=$ FALSE;
repeat
  for $i := 1$ **to** 100 **do**
    $\bar{X} :=$ destruction $(M^{core}, \omega, X, u^*)$ % select a partial cover from X
    $X :=$ construction $(M^{core}, \omega, \bar{X}, u^*)$;
    $X :=$ local_optimization (M^{core}, ω, X); % state++
    j++; update improved
  if not improved and $j \geq 200$ **then**
    $X :=$ perturbation (M^{core}, X);
    $X :=$ local_optimization (M^{core}, ω, X);
    $j := 0$; improved $:=$ FALSE;
  $M^{core} :=$ pricing (M, ω, u^*);
  $(X, u^*) :=$ subgradient_phase (M^{core}, ω, u^*);
until time limit not exceeded ;

Lagrangian Multi-Start Heuristic

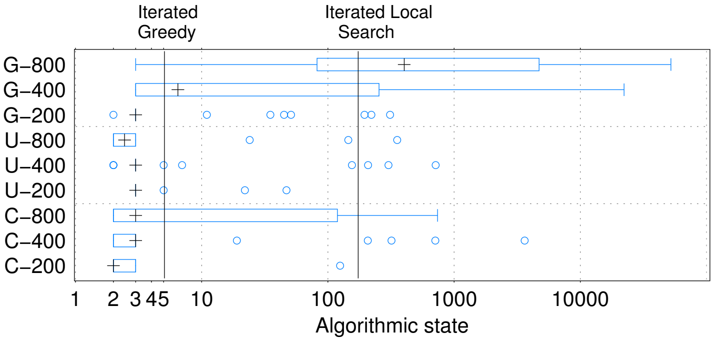
```
Function LMS (M, ω)
u_i^0 = min_{j∈J_i} w_j/|I_j| for all i ∈ F;
M^core := pricing (M, ω, u^0);
X := greedy_cov (M^core, ω);                                    % state := 1
(X, u*) := subgradient_phase (M^core, u_0);                      % state := 2
X := local_optimization (X);                                     % state := 3
M^core := pricing (M, ω, u*);                                    % state := 4
(X, u*) := subgradient_phase (M^core, ω, u*);                    % state := 5
j := 0; improved := FALSE;
repeat
    for i := 1 to 100 do
        X̄ := destruction (M^core, ω, X, u*)            % select a partial cover from X
        X := construction (M^core, ω, X̄, u*);
        X := local_optimization (M^core, ω, X);                  % state++
        j++; update improved
    if not improved and j ≥ 200 then
        X := perturbation (M^core, X);
        X := local_optimization (M^core, ω, X);
        j := 0; improved := FALSE;
    M^core := pricing (M, ω, u*);
    (X, u*) := subgradient_phase (M^core, ω, u*);
until time limit not exceeded ;
```

Lagrangian Multi-Start Heuristic

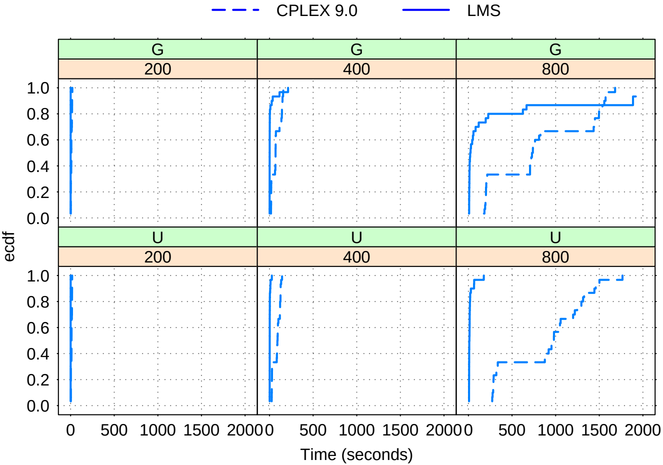
```
Function LMS (M, ω)
u_i^0 = min_{j∈J_i} w_j/|I_j| for all i ∈ F;
M^core := pricing (M, ω, u^0);
X := greedy_cov (M^core, ω);                              % state := 1
(X, u*) := subgradient_phase (M^core, u_0);               % state := 2
X := local_optimization (X);                              % state := 3
M^core := pricing (M, ω, u*);                             % state := 4
(X, u*) := subgradient_phase (M^core, ω, u*);             % state := 5
j := 0; improved := FALSE;
repeat
    for i := 1 to 100 do
        X̄ := destruction (M^core, ω, X, u*)      % select a partial cover from X
        X := construction (M^core, ω, X̄, u*);
        X := local_optimization (M^core, ω, X);            % state++
        j++; update improved
    if not improved and j ≥ 200 then
        X := perturbation (M^core, X);
        X := local_optimization (M^core, ω, X);
        j := 0; improved := FALSE;
    M^core := pricing (M, ω, u*);
    (X, u*) := subgradient_phase (M^core, ω, u*);
until time limit not exceeded ;
```

Algorithm Analysis

Are all components needed?



- Empirical cumulative distributions of the time to find the optimal solution.
- For each plot 30 instances.
- Time limit 2000 seconds.



Summary

Conclusions

- The set covering approach resulted the best to tackle this problem.

Summary

Conclusions

- The set covering approach resulted the best to tackle this problem.
- A very fast and accurate hybrid heuristic was developed.

Summary

Conclusions

- The set covering approach resulted the best to tackle this problem.
- A very fast and accurate hybrid heuristic was developed.
- On the instances studied the E1-2AUG problem seems *computationally* not very hard.

Summary

Conclusions

- The set covering approach resulted the best to tackle this problem.
- A very fast and accurate hybrid heuristic was developed.
- On the instances studied the E1-2AUG problem seems *computationally* not very hard.

Further Work

- Better exploit the availability of a lower bound in the LMS heuristic
- Search harder instances

A Computational Study on the 2-Edge-Connectivity Augmentation Problem

Jørgen Bang-Jensen, Marco Chiarandini, Peter Morling

Department of Mathematics and Computer Science
University of Southern Denmark

UNIVERSITY OF SOUTHERN DENMARK

Graph Theory 2007
Fredericia, Denmark, December 6-9, 2007

Based on: J. Bang-Jensen, M. Chiarandini, P. Morling (2007).
A computational investigation on heuristic algorithms for 2-edge-connectivity augmentation.
Tech. Rep. DMF-2007-07-005, The Danish Mathematical Society. Submitted to journal.