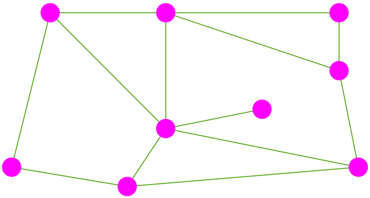
Scientific Computing

Parallele Algorithmen

Prof. Dr. Stefan Funken, Prof. Dr. Alexander Keller,
Prof. Dr. Karsten Urban | 17. Januar 2007

Graph

In the following let $\mathcal{G} = (V, E)$ be a **graph**
with **nodes V** (vertices) and **undirected edges E**.



Let $\mathbf{n} = |\mathbf{V}|$ and $\mathbf{e} = |\mathbf{E}|$.

Partition of a Graph

Let $\pi : V \mapsto \{1, \ldots, p\}$ be a partition of a graph $\mathcal{G}$ that distributes the nodes among p clusters $V_1, \ldots, V_p$.

Partition of a Graph

Let $\pi : V \mapsto \{1, \ldots, p\}$ be a partition of a graph $\mathcal{G}$
that distributes the nodes among p clusters $V_1, \ldots, V_p$.

If $p = 2$, π is called bisection.

Partition of a Graph

Let $\pi : V \mapsto \{1, \dots, p\}$ be a partition of a graph $\mathcal{G}$
that distributes the nodes among p clusters $V_1, \dots, V_p$.

If $p = 2$, π is called bisection.

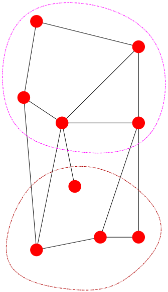
The major characteristics of a partition are its balance and its cut size.

Definition (Balance)

The **balance** is defined by

$$bal(\pi) := \max_{1 \leq \ell \leq p} |V_\ell| - \min_{1 \leq \ell \leq p} |V_\ell|.$$

If $bal(\pi) \leq 1$, π is called a **balanced partition**.



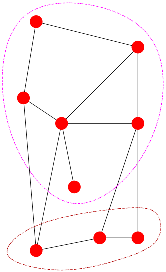
balanced partition

Definition (Balance)

The **balance** is defined by

$$bal(\pi) := \max_{1\le\ell\le p} |V_\ell| - \min_{1\le\ell\le p} |V_\ell|.$$

If $bal(\pi) \le 1$, π is called a **balanced partition**.



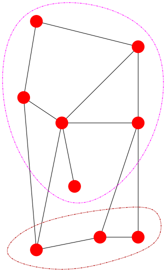
unbalanced partition

Definition (Balance)

The **balance** is defined by

$$bal(\pi) := \max_{1 \le \ell \le p} |V_\ell| - \min_{1 \le \ell \le p} |V_\ell|.$$

If $bal(\pi) \le 1$, π is called a **balanced partition**.



unbalanced partition

Remark

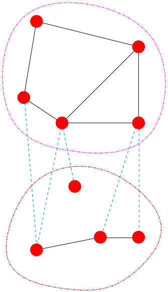
A **low balance** ensures an **even distribution** of the total process-work among all processors.

Definition (Cut Size)

Let

$$cut(\pi) := |\{\{v, w\} \in E : \pi(v) \neq \pi(w)\}|$$

be the **cut size** of π.



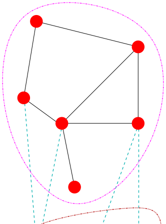
cut size: 5

Definition (Cut Size)

Let

$$cut(\pi) := |\{\{v, w\} \in E : \pi(v) \neq \pi(w)\}|$$

be the **cut size** of π.

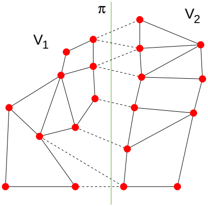


Bisection Problem

Partitioning Problem

Find a partition π that minimizes the cut size while keeping the balance as low as possible.

It is called **bisection problem** if $p = 2$.

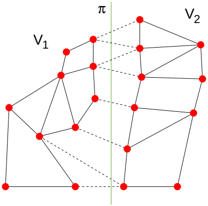


Bisection Problem

Partitioning Problem

Find a partition π that minimizes the cut size while keeping the balance as low as possible.

It is called **bisection problem** if $p = 2$.



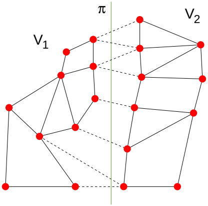
n	:	20
e	:	38
cut size	:	9
balance	:	0

Bisection Problem

Partitioning Problem

Find a partition π that minimizes the cut size while keeping the balance as low as possible.

It is called **bisection problem** if $p = 2$.



n	:	20
e	:	38
cut size	:	9
balance	:	0

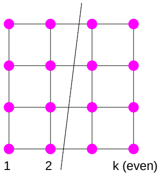
The number of possible bisections of a graph is
$\frac{1}{2}\binom{n}{n/2}$ (n even) resp. $\binom{n}{(n-1)/2}$ (n odd).
For our example we have 92.378 possible bisections.

Bisection Problem

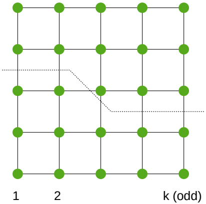
Definition (Bisection Width)

$$bw(\mathcal{G}) := \min\{cut(\pi) | p = 2, bal(\pi) \leq 1\}$$

is the **bisection width** of the graph $\mathcal{G}$.



$bw (G) = k$



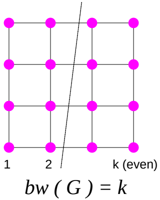
$bw (G) = k+1$

Bisection Problem

Definition (Bisection Width)

$$bw(\mathcal{G}) := \min\{cut(\pi) | p = 2, bal(\pi) \leq 1\}$$

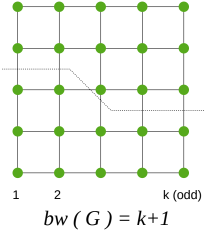
is the **bisection width** of the graph $\mathcal{G}$.



Remark

The problem of calculating the bisection width for an abitrary graph is *NP*-complete.

[T.Lengauer: Cobinatorial Algorithms for Integrated Circuit Layout, B.G. Teubner, 1990]

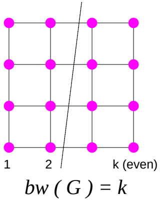


Bisection Problem

Definition (Bisection Width)

$$bw(\mathcal{G}) := \min\{cut(\pi) | p = 2, bal(\pi) \leq 1\}$$

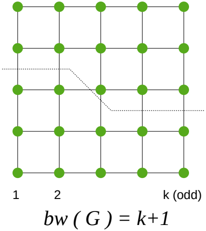
is the **bisection width** of the graph $\mathcal{G}$.



Remark

The problem of calculating the bisection width for an abitrary graph is NP-complete.

[T.Lengauer: Cobinatorial Algorithms for Integrated Circuit Layout, B.G. Teubner, 1990]



Therefore, heuristics are used to compute in adequate time a bisection with a cut as low as possible.

Partitioning Problem

The partitioning of the fe-mesh, should also take into account

- shape of the resulting subdomains (possible influence on solver),

Partitioning Problem

The partitioning of the fe-mesh, should also take into account

- shape of the resulting subdomains (possible influence on solver),
- weaker constraint on the balance possible,

Partitioning Problem

The partitioning of the fe-mesh, should also take into account

- shape of the resulting subdomains (possible influence on solver),
- weaker constraint on the balance possible,
- necessary to partition in parallel,

Partitioning Problem

The partitioning of the fe-mesh, should also take into account

- shape of the resulting subdomains (possible influence on solver),
- weaker constraint on the balance possible,
- necessary to partition in parallel,
- allow motion/deformation of the domain,

Partitioning Problem

The partitioning of the fe-mesh, should also take into account

- shape of the resulting subdomains (possible influence on solver),
- weaker constraint on the balance possible,
- necessary to partition in parallel,
- allow motion/deformation of the domain,
- easy data-structure/handling for solver/post-processing,...

Partitioning Problem

The partitioning of the fe-mesh, should also take into account

- shape of the resulting subdomains (possible influence on solver),
- weaker constraint on the balance possible,
- necessary to partition in parallel,
- allow motion/deformation of the domain,
- easy data-structure/handling for solver/post-processing,...
- allow adaptivity,

Partitioning Problem

The partitioning of the fe-mesh, should also take into account

- shape of the resulting subdomains (possible influence on solver),
- weaker constraint on the balance possible,
- necessary to partition in parallel,
- allow motion/deformation of the domain,
- easy data-structure/handling for solver/post-processing,...
- allow adaptivity,
- partioning into $p = 2^k$ processors, etc.

Heuristics for Graph Partitioning

Global methods: graph description as input and generate a balanced bisection

Heuristics for Graph Partitioning

Global methods: graph description as input and generate a balanced bisection

- simple node-numbering bisection
- coordinate sorting
- nearest-neighbour bisection
- connectivity bisection
- greedy bisection
- inertial bisection
- spectral bisection
- ...

Heuristics for Graph Partitioning

Global methods: graph description as input and generate a balanced bisection

- simple node-numbering bisection
- coordinate sorting
- nearest-neighbour bisection
- connectivity bisection
- greedy bisection
- inertial bisection
- spectral bisection
- ...

Local methods: graph and bisection as input and try to improve the partition

Heuristics for Graph Partitioning

Global methods: graph description as input and generate a balanced bisection

- simple node-numbering bisection
- coordinate sorting
- nearest-neighbour bisection
- connectivity bisection
- greedy bisection
- inertial bisection
- spectral bisection
- ...

Local methods: graph and bisection as input and try to improve the partition

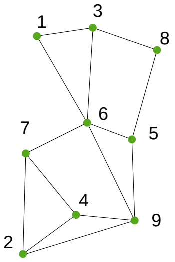
- Kerningham-Lin
- simulated annealing
- ...

Simple Node-Numbering Bisection

Given nodes $v_1, \ldots, v_n$.

Linear bisection: $\pi(v_i) = \begin{cases} 1, & i < n/2 \\ 2, & i \geq n/2 \end{cases}$

Scattered bisection: $\pi(v_i) = \begin{cases} 1, & i \text{ even} \\ 2, & i \text{ odd} \end{cases}$

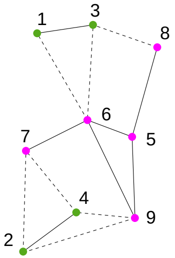


Simple Node-Numbering Bisection

Given nodes $v_1, \ldots, v_n$.

Linear bisection: $$\pi(v_i) = \begin{cases} 1, & i < n/2 \\ 2, & i \geq n/2 \end{cases}$$

Scattered bisection: $$\pi(v_i) = \begin{cases} 1, & i \text{ even} \\ 2, & i \text{ odd} \end{cases}$$



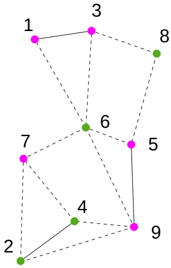
linear bisection

Simple Node-Numbering Bisection

Given nodes $v_1, \ldots, v_n$.

Linear bisection: $\pi(v_i) = \begin{cases} 1, & i < n/2 \\ 2, & i \geq n/2 \end{cases}$

Scattered bisection: $\pi(v_i) = \begin{cases} 1, & i \text{ even} \\ 2, & i \text{ odd} \end{cases}$



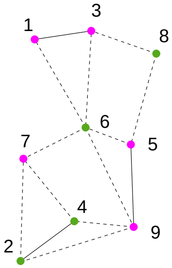
scattered bisection

Simple Node-Numbering Bisection

Given nodes $v_1, \ldots, v_n$.

Linear bisection: $$\pi(v_i) = \begin{cases} 1, & i < n/2 \\ 2, & i \geq n/2 \end{cases}$$

Scattered bisection: $$\pi(v_i) = \begin{cases} 1, & i \text{ even} \\ 2, & i \text{ odd} \end{cases}$$



scattered bisection

Remark

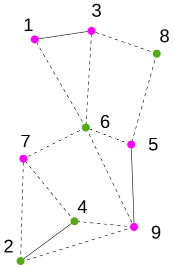
- simple and fast

Simple Node-Numbering Bisection

Given nodes $v_1, \ldots, v_n$.

Linear bisection: $$\pi(v_i) = \begin{cases} 1, & i < n/2 \\ 2, & i \geq n/2 \end{cases}$$

Scattered bisection: $$\pi(v_i) = \begin{cases} 1, & i \text{ even} \\ 2, & i \text{ odd} \end{cases}$$



scattered bisection

Remark

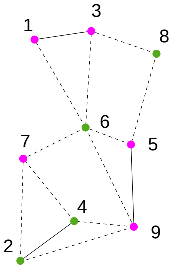
- simple and fast
- produces a balanced bisection

Simple Node-Numbering Bisection

Given nodes $v_1, \ldots, v_n$.

Linear bisection: $$\pi(v_i) = \begin{cases} 1, & i < n/2 \\ 2, & i \geq n/2 \end{cases}$$

Scattered bisection: $$\pi(v_i) = \begin{cases} 1, & i \text{ even} \\ 2, & i \text{ odd} \end{cases}$$



scattered bisection

Remark

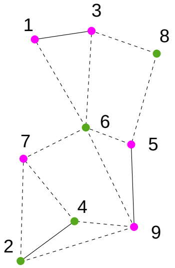
- simple and fast
- produces a balanced bisection
- generally, poor cut size (no attention to adjaceny information of graph)

Simple Node-Numbering Bisection

Given nodes $v_1, \ldots, v_n$.

Linear bisection: $$\pi(v_i) = \begin{cases} 1, & i < n/2 \\ 2, & i \geq n/2 \end{cases}$$

Scattered bisection: $$\pi(v_i) = \begin{cases} 1, & i \text{ even} \\ 2, & i \text{ odd} \end{cases}$$



scattered bisection

Remark

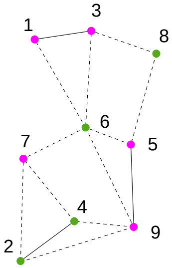
- simple and fast
- produces a balanced bisection
- generally, poor cut size (no attention to adjaceny information of graph)
- special cases, succeed in good bisection

Simple Node-Numbering Bisection

Given nodes $v_1, \ldots, v_n$.

Linear bisection: $$\pi(v_i) = \begin{cases} 1, & i < n/2 \\ 2, & i \geq n/2 \end{cases}$$

Scattered bisection: $$\pi(v_i) = \begin{cases} 1, & i \text{ even} \\ 2, & i \text{ odd} \end{cases}$$



scattered bisection

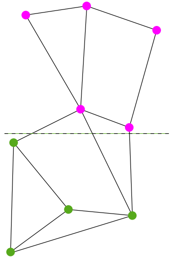
Remark

- simple and fast
- produces a balanced bisection
- generally, poor cut size (no attention to adjaceny information of graph)
- special cases, succeed in good bisection
- no dimensional restriction

Coordinate Sorting

First step: the longest expansion
of any dimension is determined

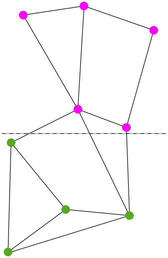
Second step: nodes are sorted according
to their coordinates in that dimension



Coordinate Sorting

First step: the longest expansion
of any dimension is determined

Second step: nodes are sorted according
to their coordinates in that dimension



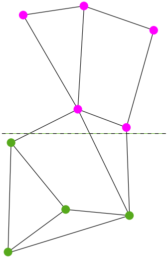
Remark

- simple and fast

Coordinate Sorting

First step: the longest expansion of any dimension is determined

Second step: nodes are sorted according to their coordinates in that dimension



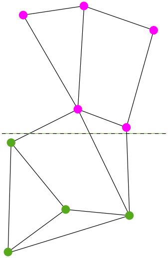
Remark

- simple and fast
- produces a balanced bisection

Coordinate Sorting

First step: the longest expansion of any dimension is determined

Second step: nodes are sorted according to their coordinates in that dimension



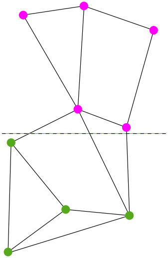
Remark

- simple and fast
- produces a balanced bisection
- does not take advantage of adjaceny information of the graph

Coordinate Sorting

First step: the longest expansion
of any dimension is determined

Second step: nodes are sorted according
to their coordinates in that dimension

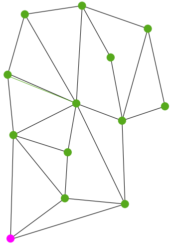


Remark

- simple and fast
- produces a balanced bisection
- does not take advantage of adjaceny information of the graph
- no dimensional restriction

Nearest-Neighbour Bisection

i) initialize V_1 with one node from V

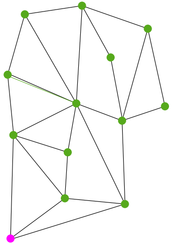


$\mathbf{bal}(\pi) = \mathbf{12}$

Nearest-Neighbour Bisection

i) initialize V_1 with one node from V

ii) all nodes from $V \setminus V_1$ adjacent to any node of V_1 are identified and moved to V_1



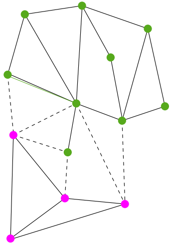
$\mathbf{bal}(\pi) = \mathbf{12}$

Nearest-Neighbour Bisection

i) initialize V_1 with one node from V

ii) all nodes from $V \setminus V_1$ adjacent to any node of V_1 are identified and moved to V_1

iii) continue with i) until V_1 reaches the required size of $\lfloor \frac{n}{2} \rfloor$



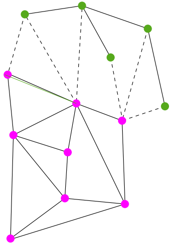
$\mathbf{bal}(\pi) = \mathbf{5}$

Nearest-Neighbour Bisection

i) initialize V_1 with one node from V

ii) all nodes from $V \setminus V_1$ adjacent to any node of V_1 are identified and moved to V_1

iii) continue with i) until V_1 reaches the required size of $\lfloor \frac{n}{2} \rfloor$



$\mathbf{bal}(\pi) = \mathbf{3}$

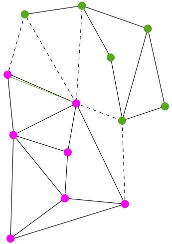
Nearest-Neighbour Bisection

i) initialize V_1 with one node from V

ii) all nodes from $V \setminus V_1$ adjacent to any node of V_1 are identified and moved to V_1

iii) continue with i) until V_1 reaches the required size of $\lfloor \frac{n}{2} \rfloor$

iv) (do not take all possible nodes in the last step!)



$\mathbf{bal}(\pi) = \mathbf{1}$

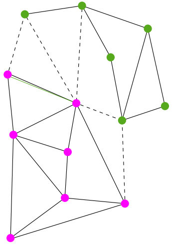
Nearest-Neighbour Bisection

i) initialize V_1 with one node from V

ii) all nodes from $V \setminus V_1$ adjacent to any node of V_1 are identified and moved to V_1

iii) continue with i) until V_1 reaches the required size of $\lfloor \frac{n}{2} \rfloor$

iv) (do not take all possible nodes in the last step!)



$\mathbf{bal}(\pi) = \mathbf{1}$

Remark

- fast, more complex data structure necessary

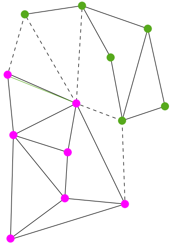
Nearest-Neighbour Bisection

i) initialize V_1 with one node from V

ii) all nodes from $V \setminus V_1$ adjacent to any node of V_1 are identified and moved to V_1

iii) continue with i) until V_1 reaches the required size of $\lfloor \frac{n}{2} \rfloor$

iv) (do not take all possible nodes in the last step!)



$\mathbf{bal}(\pi) = \mathbf{1}$

Remark

- fast, more complex data structure necessary
- balanced bisection only with modification iv)

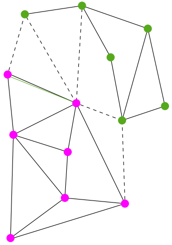
Nearest-Neighbour Bisection

i) initialize V_1 with one node from V

ii) all nodes from $V \setminus V_1$ adjacent to any node of V_1 are identified and moved to V_1

iii) continue with i) until V_1 reaches the required size of $\lfloor \frac{n}{2} \rfloor$

iv) (do not take all possible nodes in the last step!)



$\mathbf{bal}(\pi) = \mathbf{1}$

Remark

- fast, more complex data structure necessary
- balanced bisection only with modification iv)
- direct p-partitioning possible (run up to size n/p)

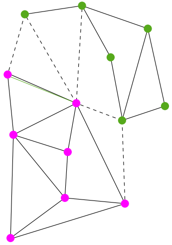
Nearest-Neighbour Bisection

i) initialize V_1 with one node from V

ii) all nodes from $V \setminus V_1$ adjacent to any node of V_1 are identified and moved to V_1

iii) continue with i) until V_1 reaches the required size of $\lfloor \frac{n}{2} \rfloor$

iv) (do not take all possible nodes in the last step!)



$\mathbf{bal}(\pi) = \mathbf{1}$

Remark

- fast, more complex data structure necessary
- balanced bisection only with modification iv)
- direct p-partitioning possible (run up to size n/p)
- building process depends on initial node

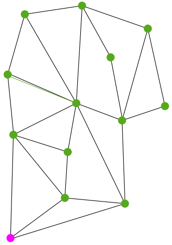
Connectivity Bisection

For two nodes $v, w \in V$ let

$$distance(v, w) := |\text{shortest path connecting } v \text{ and } w|$$

be the distance between v and w.

i) determine two vertices with a (near) maximum distance



$\mathbf{bal}(\pi) = \mathbf{11}$

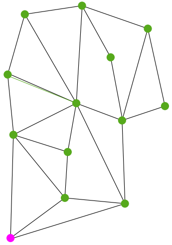
Connectivity Bisection

For two nodes $v, w \in V$ let

$$distance(v, w) := |\text{shortest path connecting } v \text{ and } w|$$

be the distance between v and w.

i) determine two vertices with a (near) maximum distance
ii) all other nodes are sorted in order of increasing distance from one of the extremal nodes



$\mathbf{bal}(\pi) = \mathbf{11}$

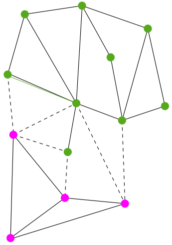
Connectivity Bisection

For two nodes $v, w \in V$ let

$$distance(v, w) := |\text{shortest path connecting } v \text{ and } w|$$

be the distance between v and w.

i) determine two vertices with a (near) maximum distance
ii) all other nodes are sorted in order of increasing distance from one of the extremal nodes
iii) nodes are assigned to V_1 and V_2 according to this list



$\mathbf{bal}(\pi) = \mathbf{5}$

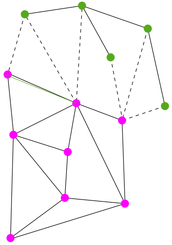
Connectivity Bisection

For two nodes $v, w \in V$ let

$$distance(v, w) := |\text{shortest path connecting } v \text{ and } w|$$

be the distance between v and w.

i) determine two vertices with a (near) maximum distance
ii) all other nodes are sorted in order of increasing distance from one of the extremal nodes
iii) nodes are assigned to V_1 and V_2 according to this list



$\mathbf{bal}(\pi) = \mathbf{3}$

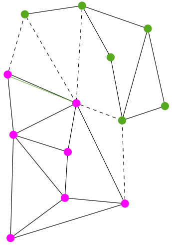
Connectivity Bisection

For two nodes $v, w \in V$ let

$$distance(v, w) := |\text{shortest path connecting } v \text{ and } w|$$

be the distance between v and w.

i) determine two vertices with a (near) maximum distance
ii) all other nodes are sorted in order of increasing distance from one of the extremal nodes
iii) nodes are assigned to V_1 and V_2 according to this list



$\mathbf{bal}(\pi) = \mathbf{1}$

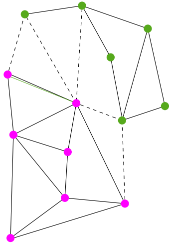
Connectivity Bisection

For two nodes $v, w \in V$ let

$$distance(v, w) := |\text{shortest path connecting } v \text{ and } w|$$

be the distance between v and w.

i) determine two vertices with a (near) maximum distance
ii) all other nodes are sorted in order of increasing distance from one of the extremal nodes
iii) nodes are assigned to V_1 and V_2 according to this list



$\mathbf{bal}(\pi) = \mathbf{1}$

Remark

- combination of coordinate sorting and nearest-neighbour bisection

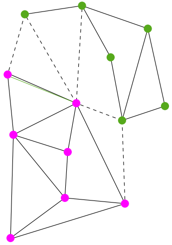
Connectivity Bisection

For two nodes $v, w \in V$ let

$$distance(v, w) := |\text{shortest path connecting } v \text{ and } w|$$

be the distance between v and w.

i) determine two vertices with a (near) maximum distance
ii) all other nodes are sorted in order of increasing distance from one of the extremal nodes
iii) nodes are assigned to V_1 and V_2 according to this list



$\mathbf{bal}(\pi) = \mathbf{1}$

Remark

- combination of coordinate sorting and nearest-neighbour bisection
- no coordinate information necessary

Basic Tool for Local Rearrangements

Definition

For a node $v \in V$ let

$deg(v) := \|\{w \in V : (v,w) \in E\}\|$	be its **degree**
$int(v) := \|\{w \in V : (v,w) \in E, \pi(v) = \pi(w)\}\|$	be its number of **internal edges**
$ext(v) := \|\{w \in V : (v,w) \in E, \pi(v) \neq \pi(w)\}\|$	be its number of **external edges**

Basic Tool for Local Rearrangements

Definition

For a node $v \in V$ let

$deg(v) := \|\{w \in V : (v,w) \in E\}\|$	be its **degree**
$int(v) := \|\{w \in V : (v,w) \in E, \pi(v) = \pi(w)\}\|$	be its number of **internal edges**
$ext(v) := \|\{w \in V : (v,w) \in E, \pi(v) \neq \pi(w)\}\|$	be its number of **external edges**

Definition (diff-value)

Let $v \in V$.

$$diff(v) := ext(v) - int(v)$$

Basic Tool for Local Rearrangements

Definition

For a node $v \in V$ let

$deg(v) := \|\{w \in V : (v,w) \in E\}\|$	be its **degree**
$int(v) := \|\{w \in V : (v,w) \in E, \pi(v) = \pi(w)\}\|$	be its number of **internal edges**
$ext(v) := \|\{w \in V : (v,w) \in E, \pi(v) \neq \pi(w)\}\|$	be its number of **external edges**

Definition (diff-value)

Let $v \in V$.

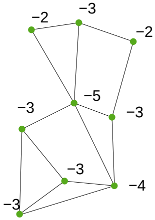
$$diff(v) := ext(v) - int(v)$$

Remark

- The **diff-value** of a node represents the change of the **cut-size** if this node to a different cluster of a bisection.
- The **diff-value** is helpful for predicting the change of the **cut-size** if the rearrangements of the bisection are accomplished by moves of single nodes.

Greedy Bisection (greedy $\simeq$ gefräßig)

i) initialise $V_1 = \{1, \ldots, n\}$ and $V_2 = \emptyset$, ($cutsize = 0, balance = n$)

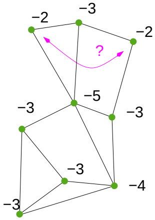


$\mathbf{bal}(\pi) = \mathbf{9}$

Greedy Bisection (greedy $\simeq$ gefräßig)

i) initialise $V_1 = \{1, \ldots, n\}$ and $V_2 = \emptyset$, ($cutsize = 0, balance = n$)

ii) move a node from V_1 to V_2 which minimally increases the cut size
take node with highest **diff-value**
(update after each move)

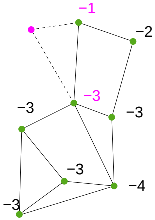


bal$(\pi) =$ **9**

Greedy Bisection (greedy $\simeq$ gefräßig)

i) initialise $V_1 = \{1, \ldots, n\}$ and $V_2 = \emptyset$, ($cutsize = 0, balance = n$)

ii) move a node from V_1 to V_2 which minimally increases the cut size
take node with highest **diff-value**
(update after each move)

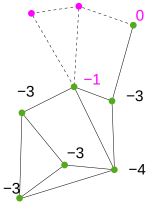


bal$(\pi) = $ **7**

Greedy Bisection (greedy $\simeq$ gefräßig)

i) initialise $V_1 = \{1, \ldots, n\}$ and $V_2 = \emptyset$, ($cutsize = 0, balance = n$)

ii) move a node from V_1 to V_2 which minimally increases the cut size
take node with highest **diff-value**
(update after each move)

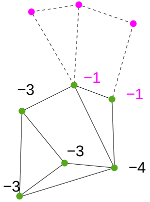


$\mathbf{bal}(\pi) = \mathbf{5}$

Greedy Bisection (greedy $\simeq$ gefräßig)

i) initialise $V_1 = \{1, \ldots, n\}$ and $V_2 = \emptyset$, ($cutsize = 0, balance = n$)

ii) move a node from V_1 to V_2 which minimally increases the cut size
take node with highest **diff-value**
(update after each move)



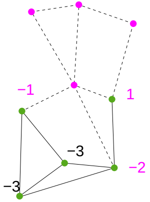
bal$(\pi) = \mathbf{3}$

Greedy Bisection (greedy $\simeq$ gefräßig)

i) initialise $V_1 = \{1, \ldots, n\}$ and $V_2 = \emptyset$,
($cutsize = 0, balance = n$)

ii) move a node from V_1 to V_2 which minimally increases the cut size
take node with highest **diff-value**
(update after each move)

iii) after $\lfloor \frac{n}{2} \rfloor$ moves the bisection is balanced



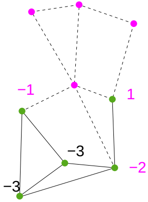
$\mathbf{bal}(\pi) = \mathbf{1}$

Greedy Bisection (greedy $\simeq$ gefräßig)

i) initialise $V_1 = \{1, \ldots, n\}$ and $V_2 = \emptyset$, ($cutsize = 0, balance = n$)

ii) move a node from V_1 to V_2 which minimally increases the cut size
take node with highest **diff-value**
(update after each move)

iii) after $\lfloor \frac{n}{2} \rfloor$ moves the bisection is balanced



$\mathbf{bal}(\pi) = \mathbf{1}$

Remark

- very fast
- produces bisections with reasonable cut sizes
- efficient updating of **diff-value** possible (using dynamical data structures)

Inertia of Moments

Consider a **rigid body** as system of **mass points**.

Inertia of Moments

Consider a **rigid body** as system of **mass points**.

Let $\sum m\vec{x} = 0$ (center of gravity vanishes).

Inertia of Moments

Consider a **rigid body** as system of **mass points**.

Let $\sum m\vec{x} = 0$ (center of gravity vanishes).

The **principal inertia axes** are hypothetical axes, on which the center of mass is located, and around which the rigid body would spin if it were in free space unencumbered by bearing or gravitational forces.

Inertia of Moments

Consider a **rigid body** as system of **mass points**.

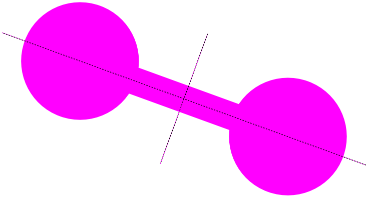
Let $\sum m\vec{x} = 0$ (center of gravity vanishes).

The **principal inertia axes** are hypothetical axes, on which the center of mass is located, and around which the rigid body would spin if it were in free space unencumbered by bearing or gravitational forces.

The **principal axes** are the **eigenvectors** of the tensor

$$I := \begin{pmatrix} \sum m(y^2+x^2) & -\sum mxy & -\sum mxz \\ -\sum mxy & \sum m(x^2+z^2) & -\sum myz \\ -\sum mxz & -\sum myz & \sum m(x^2+y^2) \end{pmatrix}$$

and the **principal moments** (of inertia) are its **eigenvalues**.



Inertial Bisection

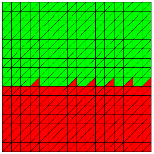
Nodes will be considered as **mass points**,

Inertial Bisection

Nodes will be considered as **mass points**,
principle inertia axes will be calculated,

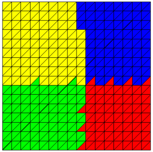
Inertial Bisection

Nodes will be considered as **mass points**, **principle inertia axes** will be calculated, and the **domain** will be **divided** into two regions by a cutting plane **orthogonal** to the **maximum inertia axis** so that $bal(\pi) \leq 1$



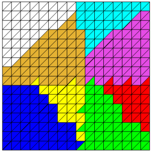
Inertial Bisection

Nodes will be considered as **mass points**, **principle inertia axes** will be calculated, and the **domain** will be **divided** into two regions by a cutting plane **orthogonal** to the **maximum inertia axis** so that $bal(\pi) \leq 1$



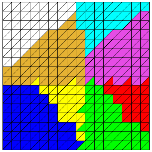
Inertial Bisection

Nodes will be considered as **mass points**, **principle inertia axes** will be calculated, and the **domain** will be **divided** into two regions by a cutting plane **orthogonal** to the **maximum inertia axis** so that $bal(\pi) \leq 1$



Inertial Bisection

Nodes will be considered as **mass points**, **principle inertia axes** will be calculated, and the **domain** will be **divided** into two regions by a cutting plane **orthogonal** to the **maximum inertia axis** so that $bal(\pi) \leq 1$

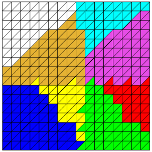


Remark

- fast

Inertial Bisection

Nodes will be considered as **mass points**, **principle inertia axes** will be calculated, and the **domain** will be **divided** into two regions by a cutting plane **orthogonal** to the **maximum inertia axis** so that $bal(\pi) \leq 1$

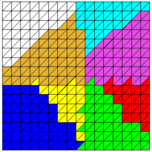


Remark

- fast
- coordinates have to be provided

Inertial Bisection

Nodes will be considered as **mass points**, **principle inertia axes** will be calculated, and the **domain** will be **divided** into two regions by a cutting plane **orthogonal** to the **maximum inertia axis** so that $bal(\pi) \leq 1$

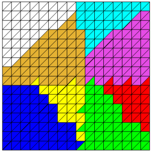


Remark

- fast
- coordinates have to be provided
- adjacency information is not considered

Inertial Bisection

Nodes will be considered as **mass points**, **principle inertia axes** will be calculated, and the **domain** will be **divided** into two regions by a cutting plane **orthogonal** to the **maximum inertia axis** so that $bal(\pi) \leq 1$



Remark

- fast
- coordinates have to be provided
- adjacency information is not considered
- computes reasonable bisections for most applications

Fiedler vector

Definition and Theorem: Laplacian Matrix $L(\mathcal{G})$

Let $\mathcal{G}(V,E)$ be given, $n := |V|$.
We define $L(\mathcal{G}) = (\ell_{ij})$ by

$$\ell_{ij} = \begin{cases} -1, & i \neq j \text{ and } (i,j) \in E \\ 0, & i \neq j \text{ and } (i,j) \notin E \\ deg(i), & i = j \end{cases}$$

Fiedler vector

Definition and Theorem: Laplacian Matrix $L(\mathcal{G})$

Let $\mathcal{G}(V, E)$ be given, $n := |V|$.
We define $L(\mathcal{G}) = (\ell_{ij})$ by

$$\ell_{ij} = \begin{cases} -1, & i \neq j \text{ and } (i,j) \in E \\ 0, & i \neq j \text{ and } (i,j) \notin E \\ deg(i), & i = j \end{cases}$$

The Laplacian matrix L is symmetric and
the sum of each row and column is 0.
(Hence 0 is an eigenvalue of L with eigenvector $\mathbf{1}$.)

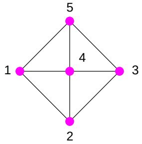
Fiedler vector

Definition and Theorem: Laplacian Matrix $L(\mathcal{G})$

Let $\mathcal{G}(V, E)$ be given, $n := |V|$.
We define $L(\mathcal{G}) = (\ell_{ij})$ by

$$\ell_{ij} = \begin{cases} -1, & i \neq j \text{ and } (i,j) \in E \\ 0, & i \neq j \text{ and } (i,j) \notin E \\ deg(i), & i = j \end{cases}$$

The Laplacian matrix L is symmetric and
the sum of each row and column is 0.
(Hence 0 is an eigenvalue of L with eigenvector $\mathbf{1}$.)



$$\begin{pmatrix} 3 & -1 & & -1 & -1 \\ -1 & 3 & -1 & -1 & \\ & -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & & -1 & -1 & 3 \end{pmatrix}$$

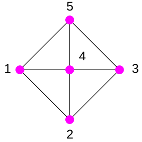
Fiedler vector

Definition and Theorem: Laplacian Matrix $L(\mathcal{G})$

Let $\mathcal{G}(V, E)$ be given, $n := |V|$.
We define $L(\mathcal{G}) = (\ell_{ij})$ by

$$\ell_{ij} = \begin{cases} -1, & i \neq j \text{ and } (i,j) \in E \\ 0, & i \neq j \text{ and } (i,j) \notin E \\ deg(i), & i = j \end{cases}$$

The Laplacian matrix L is symmetric and the sum of each row and column is 0.
(Hence 0 is an eigenvalue of L with eigenvector $\mathbf{1}$.)



$$\begin{pmatrix} 3 & -1 & & -1 & -1 \\ -1 & 3 & -1 & -1 & \\ & -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & & -1 & -1 & 3 \end{pmatrix}$$

Definition and Theorem: Fiedler Vector

The **eigenvector** $\vec{y}$ of the **second smallest eigenvalue** λ_2 of the Laplacian matrix is called **Fiedler vector**.

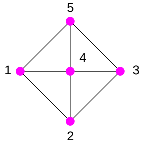
Fiedler vector

Definition and Theorem: Laplacian Matrix $L(\mathcal{G})$

Let $\mathcal{G}(V, E)$ be given, $n := |V|$.
We define $L(\mathcal{G}) = (\ell_{ij})$ by

$$\ell_{ij} = \begin{cases} -1, & i \neq j \text{ and } (i,j) \in E \\ 0, & i \neq j \text{ and } (i,j) \notin E \\ deg(i), & i = j \end{cases}$$

The Laplacian matrix L is symmetric and the sum of each row and column is 0.
(Hence 0 is an eigenvalue of L with eigenvector $\mathbf{1}$.)



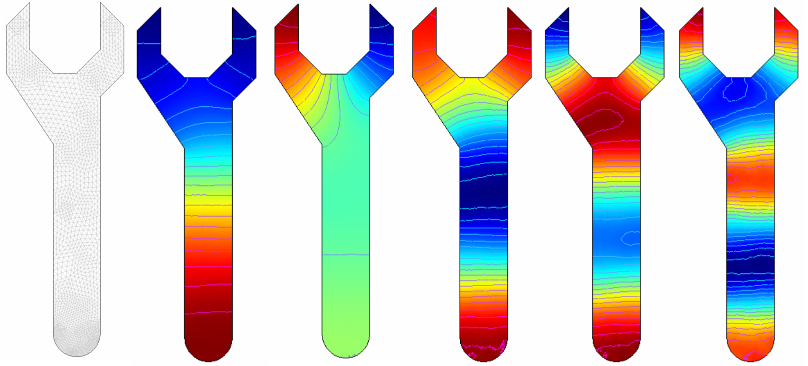
$$\begin{pmatrix} 3 & -1 & & -1 & -1 \\ -1 & 3 & -1 & -1 & \\ & -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & & -1 & -1 & 3 \end{pmatrix}$$

Definition and Theorem: Fiedler Vector

The **eigenvector** $\vec{y}$ of the **second smallest eigenvalue** λ_2 of the Laplacian matrix is called **Fiedler vector**.
Let $c \in \mathbb{R}$. If $\mathcal{G}$ is a **connected graph**, those nodes v of $\mathcal{G}$ with $y_v \geq c$ and those with $y_v < c$ each form a **connected subgraph** of $\mathcal{G}$.

Numerical Examples / Simple 2d Geometry

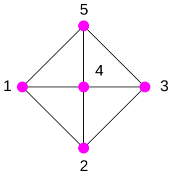
Initial mesh and eigenmodes 2-6.



initial mesh $\lambda_2 = 3.21$ $\lambda_2 = 4.52$ $\lambda_2 = 5.78$ $\lambda_2 = 10.52$ $\lambda_2 = 13.37$

Spectral Bisection

i) construct the Laplacian matrix

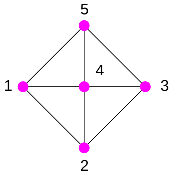


$$\begin{pmatrix} 3 & -1 & & -1 & -1 \\ -1 & 3 & -1 & -1 & \\ & -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & & -1 & -1 & 3 \end{pmatrix}$$

Spectral Bisection

i) construct the Laplacian matrix

ii) compute Fiedler vector $\vec{y}$



$$\begin{pmatrix} 3 & -1 & & -1 & -1 \\ -1 & 3 & -1 & -1 & \\ & -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & & -1 & -1 & 3 \end{pmatrix}$$

Spectral Bisection

i) construct the Laplacian matrix

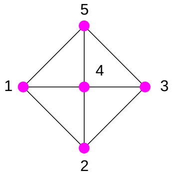
ii) compute Fiedler vector $\vec{y}$

iii) partition the nodes of $\mathcal{G}$ according to the median value y_m of the components of $\vec{y}$

$$V_1 = \{v \in V : y_v < y_m\} \text{ and}$$

$$V_2 = \{v \in V : y_v > y_m\}$$

distribute $\{v \in V : y_v = y_m\}$ among V_1, V_2, s.t. $bal(\pi) \leq 1$



$$\begin{pmatrix} 3 & -1 & & -1 & -1 \\ -1 & 3 & -1 & -1 & \\ & -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & & -1 & -1 & 3 \end{pmatrix}$$

Spectral Bisection

i) construct the Laplacian matrix

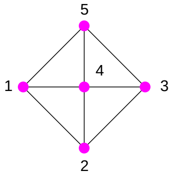
ii) compute Fiedler vector $\vec{y}$

iii) partition the nodes of $\mathcal{G}$ according to the median value y_m of the components of $\vec{y}$

$$V_1 = \{v \in V : y_v < y_m\} \text{ and}$$

$$V_2 = \{v \in V : y_v > y_m\}$$

distribute $\{v \in V : y_v = y_m\}$ among V_1, V_2, s.t. $bal(\pi) \leq 1$



$$\begin{pmatrix} 3 & -1 & & -1 & -1 \\ -1 & 3 & -1 & -1 & \\ & -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & & -1 & -1 & 3 \end{pmatrix}$$

Remark

- calculation of eigenvectors is time-consuming

Spectral Bisection

i) construct the Laplacian matrix

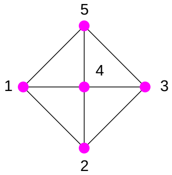
ii) compute Fiedler vector $\vec{y}$

iii) partition the nodes of $\mathcal{G}$ according to the median value y_m of the components of $\vec{y}$

$$V_1 = \{v \in V : y_v < y_m\} \text{ and}$$

$$V_2 = \{v \in V : y_v > y_m\}$$

distribute $\{v \in V : y_v = y_m\}$ among V_1, V_2, s.t. $bal(\pi) \leq 1$



$$\begin{pmatrix} 3 & -1 & & -1 & -1 \\ -1 & 3 & -1 & -1 & \\ & -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & & -1 & -1 & 3 \end{pmatrix}$$

Remark

- calculation of eigenvectors is time-consuming
- components have to be very correct for a correct partition according to the median component.

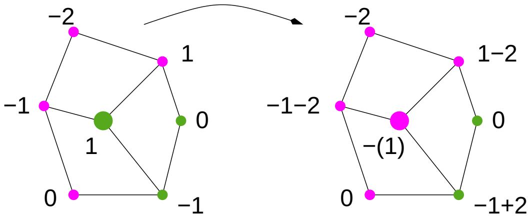
Local methods

Theorem:

Let $\pi : V \mapsto \{1,2\}$ be a bisection and $v \in V$.
If v moves to the other cluster, the **cut size** of the new bisection decreases by $\mathit{diff}(v)$ and the diff-values of the nodes in V change in teh following way:

$$v: \quad \mathit{diff}(v) = -\mathit{diff}(v)$$

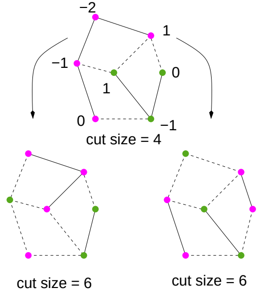
$$w \in V \setminus \{v\}: \quad \mathit{diff}(w) = \begin{cases} \mathit{diff}(w)+2, & (v,w) \in E \text{ and } \pi(w) = \pi(v) \\ \mathit{diff}(w)-2, & (v,w) \in E \text{ and } \pi(w) \neq \pi(v) \\ \mathit{diff}(w), & \mathit{otherwise} \end{cases}$$



Definition:

For a pair (v, w), $v \in V_1$, $w \in V_2$ of nodes let

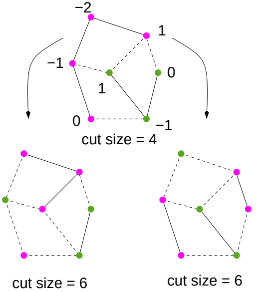
$$gain(v,w) := diff(v) + diff(w) - \begin{cases} 2, & (v,w) \in E \\ 0, & otherwise \end{cases}$$



Definition:

For a pair (v, w), $v \in V_1$, $w \in V_2$ of nodes let

$$gain(v, w) := diff(v) + diff(w) - \begin{cases} 2, & (v, w) \in E \\ 0, & otherwise \end{cases}$$



Remark:

The value of **gain(v, w)** describes the decrease in the **cut size** if v and w are exchanged.
It plays major role in the **Kerningham-Lin** algorithm.

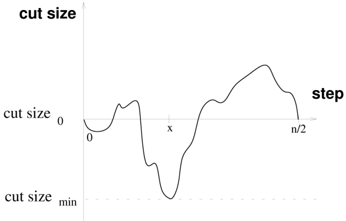
Kerningham-Lin Algorithm to Improve Bisection

Let π a partition, s.t. $bal(\pi) \leq 1$.

REPEAT

REPEAT cut size is not improved

step	pair	gain	cut size	
0			$cut\ size_0$	
1	(v_1, w_1)	$gain_1$	$cut\ size_1$	
2	(v_2, w_2)	$gain_2$	$cut\ size_2$	physical
$\vdots$	$\vdots$	$\vdots$	$\vdots$	exchange
x	(v_x, w_x)	$gain_x$	$cut\ size_{min}$	$\Leftarrow$ minimum cut size
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\lfloor \frac{n}{2} \rfloor$	$(u_{\lfloor \frac{n}{2} \rfloor}, v_{\lfloor \frac{n}{2} \rfloor})$	$gain_{\lfloor \frac{n}{2} \rfloor}$	$cutsize_{\lfloor \frac{n}{2} \rfloor}$	



Kerningham-Lin Algorithm to Improve Bisection

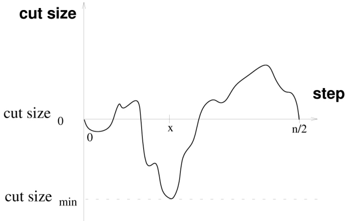
Let π a partition, s.t. *bal*$(\pi) \leq 1$.

REPEAT

compute the diff-values of all nodes, initialise all nodes as unlocked

REPEAT cut size is not improved

step	pair	gain	cut size	
0			$cut\ size_0$	
1	(v_1, w_1)	$gain_1$	$cut\ size_1$	
2	(v_2, w_2)	$gain_2$	$cut\ size_2$	physical
⋮	⋮	⋮	⋮	exchange
x	(v_x, w_x)	$gain_x$	$cut\ size_{min}$	⇐ minimum cut size
⋮	⋮	⋮	⋮	
$\lfloor \frac{n}{2} \rfloor$	$(u_{\lfloor \frac{n}{2} \rfloor}, v_{\lfloor \frac{n}{2} \rfloor})$	$gain_{\lfloor \frac{n}{2} \rfloor}$	$cutsize_{\lfloor \frac{n}{2} \rfloor}$	



Kerningham-Lin Algorithm to Improve Bisection

Let π a partition, s.t. $bal(\pi) \leq 1$.

REPEAT

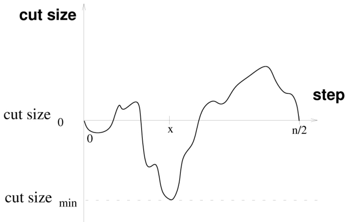
compute the diff-values of all nodes, initialise all nodes as unlocked

REPEAT $\lfloor n/2 \rfloor$ times

choose unlocked nodes $v \in V_1$ and $w \in V_2$ with $gain(v, w)$ maximal

REPEAT cut size is not improved

step	pair	gain	cut size	
0			$cut\ size_0$	
1	(v_1, w_1)	$gain_1$	$cut\ size_1$	
2	(v_2, w_2)	$gain_2$	$cut\ size_2$	physical
⋮	⋮	⋮	⋮	exchange
x	(v_x, w_x)	$gain_x$	$cut\ size_{min}$	⇐ minimum cut size
⋮	⋮	⋮	⋮	
$\lfloor \frac{n}{2} \rfloor$	$(u_{\lfloor \frac{n}{2} \rfloor}, v_{\lfloor \frac{n}{2} \rfloor})$	$gain_{\lfloor \frac{n}{2} \rfloor}$	$cutsize_{\lfloor \frac{n}{2} \rfloor}$	



Kerningham-Lin Algorithm to Improve Bisection

Let π a partition, s.t. $bal(\pi) \leq 1$.

REPEAT

compute the diff-values of all nodes, initialise all nodes as unlocked

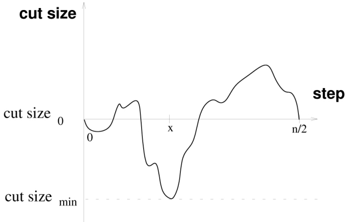
REPEAT $\lfloor n/2 \rfloor$ times

choose unlocked nodes $v \in V_1$ and $w \in V_2$ with $gain(v, w)$ maximal

exchange v and w logically and lock them;

REPEAT cut size is not improved

step	pair	gain	cut size	
0			$cut\ size_0$	
1	(v_1, w_1)	$gain_1$	$cut\ size_1$	
2	(v_2, w_2)	$gain_2$	$cut\ size_2$	physical
⋮	⋮	⋮	⋮	exchange
x	(v_x, w_x)	$gain_x$	$cut\ size_{min}$	⇐ minimum cut size
⋮	⋮	⋮	⋮	
$\lfloor \frac{n}{2} \rfloor$	$(u_{\lfloor \frac{n}{2} \rfloor}, v_{\lfloor \frac{n}{2} \rfloor})$	$gain_{\lfloor \frac{n}{2} \rfloor}$	$cutsize_{\lfloor \frac{n}{2} \rfloor}$	



Kerningham-Lin Algorithm to Improve Bisection

Let π a partition, s.t. $bal(\pi) \leq 1$.

REPEAT

compute the diff-values of all nodes, initialise all nodes as unlocked

REPEAT $\lfloor n/2 \rfloor$ times

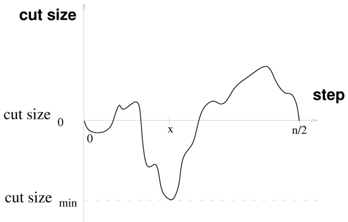
choose unlocked nodes $v \in V_1$ and $w \in V_2$ with $gain(v, w)$ maximal

exchange v and w logically and lock them;

update the *diff*-values of the neighbors;

REPEAT cut size is not improved

step	pair	gain	cut size	
0			$cut\ size_0$	
1	(v_1, w_1)	$gain_1$	$cut\ size_1$	
2	(v_2, w_2)	$gain_2$	$cut\ size_2$	physical
⋮	⋮	⋮	⋮	exchange
x	(v_x, w_x)	$gain_x$	$cut\ size_{min}$	⇐ minimum cut size
⋮	⋮	⋮	⋮	
$\lfloor \frac{n}{2} \rfloor$	$(u_{\lfloor \frac{n}{2} \rfloor}, v_{\lfloor \frac{n}{2} \rfloor})$	$gain_{\lfloor \frac{n}{2} \rfloor}$	$cutsize_{\lfloor \frac{n}{2} \rfloor}$	



Kerningham-Lin Algorithm to Improve Bisection

Let π a partition, s.t. $bal(\pi) \leq 1$.

REPEAT

compute the diff-values of all nodes, initialise all nodes as unlocked

REPEAT $\lfloor n/2 \rfloor$ times

choose unlocked nodes $v \in V_1$ and $w \in V_2$ with $gain(v, w)$ maximal

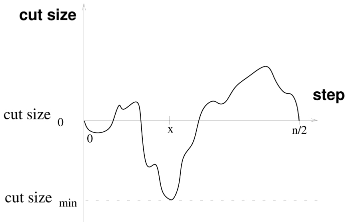
exchange v and w logically and lock them;

update the *diff*-values of the neighbors;

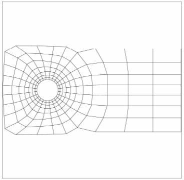
interchange physically up to the minimal cut size

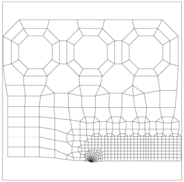
REPEAT cut size is not improved

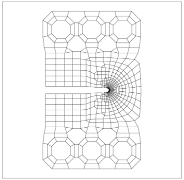
step	pair	gain	cut size	
0			$cut\ size_0$	
1	(v_1, w_1)	$gain_1$	$cut\ size_1$	
2	(v_2, w_2)	$gain_2$	$cut\ size_2$	physical
⋮	⋮	⋮	⋮	exchange
x	(v_x, w_x)	$gain_x$	$cut\ size_{min}$	⇐ minimum cut size
⋮	⋮	⋮	⋮	
$\lfloor \frac{n}{2} \rfloor$	$(u_{\lfloor \frac{n}{2} \rfloor}, v_{\lfloor \frac{n}{2} \rfloor})$	$gain_{\lfloor \frac{n}{2} \rfloor}$	$cutsize_{\lfloor \frac{n}{2} \rfloor}$	

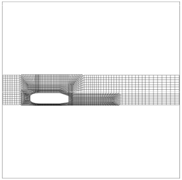


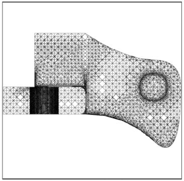
Numerical Examples / Test Graphs I

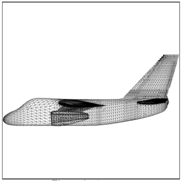

grid1_dual


netz4504_dual


ukerbe1_dual

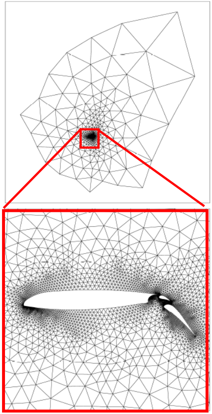

grid2_dual


brack2

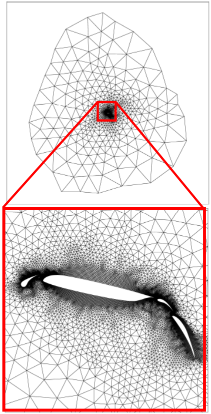

wave

The graphs ***brack2*** and ***wave*** are provided with 3-dimensional coordinates and **only** the **outer contours** of the physicals bodies **are shown**.

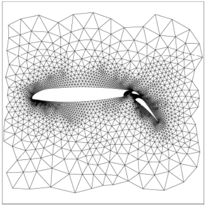
Numerical Examples / Test Graphs II



3elt



big



airfoil1

Properties of Test Graphs

Original test-graphs from different FEM-applications in 2d and 3d.
[ftp *riacs.edu*, directory */pub/grids*]

	dimension	n	e	lowest cut known
grid1_dual	2	224	420	16
netz4504_dual	2	615	1171	19
ukerbe1_dual	2	1866	3538	21
grid2_dual	2	3136	6112	32
airfoil	2	4253	12289	74
3elt	2	4720	13722	90
big	2	15606	45878	139
brack2	3	62631	366559	731
wave	3	156317	10559331	9503

Results for Test Graphs

[Robert Preis: Efficient Partitioning of Very Large Graphs with the New and Powerful Helpful-Set Heuristic]

	LIN	SCA	GRE	IN	SP	LIN	SCA	GRE	IN	SP
						+KL				
grid1_dual	152	210	26	20	20	26	**16**	26	**16**	**16**
	-	-	-	-	0.45	0.03	0.03	-	0.03	0.40
netz4504_dual	38	657	39	30	23	22	27	35	22	20
	-	-	-	0.01	0.65	0.06	0.09	-	0.04	0.74
ukerbe1_dual	22	2048	82	22	27	22	31	21	22	22
	-	-	-	0.01	1.84	0.08	0.28	0.04	0.09	1.98
grid2_dual	1862	3376	194	**32**	34	48	**32**	83	**32**	**32**
	-	-	0.02	0.02	1.58	0.43	0.41	0.05	0.12	1.65
airfoil	94	6321	121	94	138	83	91	102	83	98
	-	-	0.03	0.03	2.22	0.12	0.42	0.09	0.11	2.36
3elt	223	7018	251	209	115	**90**	190	110	121	94
	-	-	0.03	0.05	3.10	0.15	0.62	0.19	0.27	3.41
big	812	23276	248	245	162	148	228	205	200	145
	0.02	0.01	0.19	0.13	7.98	0.62	2.65	0.47	0.56	8.44
brack2	75974	192827	1083	817	827	2176	6473	**734**	783	747
	0.06	0.03	1.47	0.28	65.66	7.2	9.23	3.77	2.32	67.30
wave	40620	569226	16703	9834	9886	16855	11629	9402	9667	9611
	0.14	0.06	4.15	0.79	1430.52	11.92	23.08	15.87	7.49	1427.47

Conclusion

- The simple global methods (simple node-numbering bisection, coordinate sorting nearest-neighbour bisection, connectivity bisection) need only a **very low amount of time**, generally result in **very high cut size**.

Conclusion

- The simple global methods (simple node-numbering bisection, coordinate sorting nearest-neighbour bisection, connectivity bisection) need only a **very low amount of time**, generally result in **very high cut size**.
- The **greedy** algorithm computes is still **very fast** and computes better cut sizes than the node numbering methods.

Conclusion

- The simple global methods (simple node-numbering bisection, coordinate sorting nearest-neighbour bisection, connectivity bisection) need only a **very low amount of time**, generally result in **very high cut size**.
- The **greedy** algorithm computes is still **very fast** and computes better cut sizes than the node numbering methods.
- The **inertial method** is **very fast**, bur its results are not convincing if applied without any local heuristic. With KL cut-sizes are not more than 20% over the known optimum.

Conclusion

- The simple global methods (simple node-numbering bisection, coordinate sorting nearest-neighbour bisection, connectivity bisection) need only a **very low amount of time**, generally result in **very high cut size**.
- The **greedy** algorithm computes is still **very fast** and computes better cut sizes than the node numbering methods.
- The **inertial method** is **very fast**, bur its results are not convincing if applied without any local heuristic. With KL cut-sizes are not more than 20% over the known optimum.
- **Spectral methods** are **very expensive** but combined with KL in most cases produce partitions with a cut size of not more than 10% over the known optimum.

Conclusion

- The simple global methods (simple node-numbering bisection, coordinate sorting nearest-neighbour bisection, connectivity bisection) need only a **very low amount of time**, generally result in **very high cut size**.
- The **greedy** algorithm computes is still **very fast** and computes better cut sizes than the node numbering methods.
- The **inertial method** is **very fast**, bur its results are not convincing if applied without any local heuristic. With KL cut-sizes are not more than 20% over the known optimum.
- **Spectral methods** are **very expensive** but combined with KL in most cases produce partitions with a cut size of not more than 10% over the known optimum.
- The Kerningham-Lin algorithm improves the cut size significantly, while leading to a much higher running time.

Available Test Codes

Matlab Mesh Partitioning and Graph Separator Toolbox

It contains **Matlab code** for several graph and mesh partitioning methods, including geometric, spectral, geometric spectral, and coordinate bisection.

Graph Partitioning Software (GNU open source license)

CHACO	Leland and Hendrickson
METIS	Karypis and Kumar
PARTY	Preis
JOSTLE	Walshaw
SCOTCH	Pellegrini