Graph Partitioning

How to distribute work properly?

Manuel Gnann

Seminar parallele Numerik

Overview

Overview

Overview

Overview

Distributing work I

1. Is a parallisation of an algorithm possible?
2. Case yes: distribute work evenly among processors
   However: processors might have to wait for synchronisation
   - idea: distribute work to inactive processors $\Rightarrow$ communication, time consuming
3. Many algorithms need communication (expensive (architecture dependent), high startup time possible, ...)

Distributing work I

1. Is a parallisation of an algorithm possible?
2. Case yes: distribute work evenly among processors
   However: processors might have to wait for synchronisation
   - idea: distribute work to inactive processors $\Rightarrow$ communication, time consuming
3. Many algorithms need communication (expensive (architecture dependent), high startup time possible, ...)

Distributing work I

1. Is a parallisation of an algorithm possible?
2. Case yes: distribute work evenly among processors
   However: processors might have to wait for synchronisation
   - idea: distribute work to inactive processors $\Rightarrow$ communication, time consuming
3. Many algorithms need communication (expensive (architecture dependent), high startup time possible, ...)

Distributing work I

1. Is a parallisation of an algorithm possible?
2. Case yes: distribute work evenly among processors
   However: processors might have to wait for synchronisation
   - idea: distribute work to inactive processors $\Rightarrow$ communication, time consuming
3. Many algorithms need communication (expensive (architecture dependent), high startup time possible, ...)

Distributing work II

- problems mentioned above are machine-dependent.
- desirable: distribute work evenly and thereby minimising communication
  $\Rightarrow$ problem of partitioning a graph.

What is a graph?

Definition: Graph

A graph is a tuple $(\mathcal{N}, \mathcal{E})$ with

- $\mathcal{N} = \{v_i | i = 1, \ldots, n\}$: nodes/ vertices
- $\mathcal{E} = \left\{ e_{i,j} = (i,j) \middle| \text{ there is an edge between } v_i \text{ and } v_j \right\}$: edges

- subgraph: For $\overline{\mathcal{N}} \subset \mathcal{N}$: induced subgraph $\left(\overline{\mathcal{N}}, \overline{\mathcal{E}}\right)$.
- weights: $W_{\mathcal{E}} = \left\{ w_e\left(e_{i,j}\right) \in \mathbb{N} \middle| e_{i,j} \in \mathcal{E} \right\}$: edge weights, $W_{\mathcal{N}} = \left\{ w_v\left(v_i\right) \in \mathbb{N} | v_i \in \mathcal{N} \right\}$: weights of vertices.

What is a graph?

Definition: Graph

A graph is a tuple $(\mathcal{N}, \mathcal{E})$ with

- $\mathcal{N} = \{v_i | i = 1, \ldots, n\}$: nodes/ vertices
- $\mathcal{E} = \left\{ e_{i,j} = (i,j) \middle| \text{ there is an edge between } v_i \text{ and } v_j \right\}$: edges

- subgraph: For $\overline{\mathcal{N}} \subset \mathcal{N}$: induced subgraph $\left(\overline{\mathcal{N}}, \overline{\mathcal{E}}\right)$.
- weights: $W_{\mathcal{E}} = \left\{ w_e\left(e_{i,j}\right) \in \mathbb{N} \middle| e_{i,j} \in \mathcal{E} \right\}$: edge weights, $W_{\mathcal{N}} = \left\{ w_v\left(v_i\right) \in \mathbb{N} | v_i \in \mathcal{N} \right\}$: weights of vertices.

What is a graph?

Definition: Graph

A graph is a tuple $(\mathcal{N}, \mathcal{E})$ with

- $\mathcal{N} = \{v_i | i = 1, \ldots, n\}$: nodes/ vertices
- $\mathcal{E} = \left\{ e_{i,j} = (i,j) \middle| \text{ there is an edge between } v_i \text{ and } v_j \right\}$: edges

- subgraph: For $\overline{\mathcal{N}} \subset \mathcal{N}$: induced subgraph $\left(\overline{\mathcal{N}}, \overline{\mathcal{E}}\right)$.
- weights: $W_{\mathcal{E}} = \left\{ w_e \left(e_{i,j}\right) \in \mathbb{N} \middle| e_{i,j} \in \mathcal{E} \right\}$: edge weights, $W_{\mathcal{N}} = \{ w_v (v_i) \in \mathbb{N} | v_i \in \mathcal{N} \}$: weights of vertices.

Formulation of the problem

Problem: graph bisection (of an unweighted graph)

Let $(\mathcal{N}, \mathcal{E})$ be a graph. Find $(\mathcal{N}_1, \mathcal{N}_2)$, $\mathcal{N}_1 \dot{\cup} \mathcal{N}_2 = \mathcal{N}$ with:

1. $\#\mathcal{N}_1 = \#\mathcal{N}_2$ and
2. $\#\left\{e_{i,j} \in \mathcal{E} \middle| v_i \in \mathcal{N}_1 \text{ and } v_j \in \mathcal{N}_2\right\}$ is minimal under constraint 1.

Problem: graph partitioning (general case)

Let $(\mathcal{N}, \mathcal{E})$ be a graph with weights $W_{\mathcal{N}}$ and $W_{\mathcal{E}}$, $p \in \mathbb{N}$ and $p | \sum_{v_i \in \mathcal{N}} w_v(v_i)$. Find partition $(\mathcal{N}_i | i = 1, \ldots, p)$ such that:

1. $\dot{\bigcup}_{i=1}^{p} \mathcal{N}_i = \mathcal{N}$,
2. $\sum_{v_j \in \mathcal{N}_i} w_v\left(v_j\right)$ equal for all $i = 1, \ldots, p$.
3. $\sum_{e_{i,j} \in \mathcal{E}, v_i \in \mathcal{N}_k, v_j \in \mathcal{N}_l \text{ with } l \neq k} w_e\left(e_{i,j}\right)$ minimal under 1 & 2.

Formulation of the problem

Problem: graph bisection (of an unweighted graph)

Let $(\mathcal{N}, \mathcal{E})$ be a graph. Find $(\mathcal{N}_1, \mathcal{N}_2)$, $\mathcal{N}_1 \dot{\cup} \mathcal{N}_2 = \mathcal{N}$ with:

1. $\#\mathcal{N}_1 = \#\mathcal{N}_2$ and
2. $\#\left\{e_{i,j} \in \mathcal{E} \middle| v_i \in \mathcal{N}_1 \text{ and } v_j \in \mathcal{N}_2\right\}$ is minimal under constraint 1.

Problem: graph partitioning (general case)

Let $(\mathcal{N}, \mathcal{E})$ be a graph with weights $W_{\mathcal{N}}$ and $W_{\mathcal{E}}$, $p \in \mathbb{N}$ and $p | \sum_{v_i \in \mathcal{N}} w_v(v_i)$. Find partition $(\mathcal{N}_i | i = 1, \ldots, p)$ such that:

1. $\dot{\bigcup}_{i=1}^{p} \mathcal{N}_i = \mathcal{N}$,
2. $\sum_{v_j \in \mathcal{N}_i} w_v\left(v_j\right)$ equal for all $i = 1, \ldots, p$.
3. $\sum_{e_{i,j} \in \mathcal{E}, v_i \in \mathcal{N}_k, v_j \in \mathcal{N}_l \text{ with } l \neq k} w_e\left(e_{i,j}\right)$ minimal under 1 & 2.

Remarks

- $\#\left\{e_{i,j} \in \mathcal{E} \middle| v_i \in \mathcal{N}_1 \text{ and } v_j \in \mathcal{N}_2\right\}$ or $\sum_{e_{i,j} \in \mathcal{E}, v_i \in \mathcal{N}_k, v_j \in \mathcal{N}_l \text{ with } l \neq k} w_e\left(e_{i,j}\right)$ respectively, is called cut-size.
- subset of edges separating the graph: edge separator.
- Also vertex separator possible: Find $\mathcal{N}_s \subset \mathcal{N}$ such that
  $\mathcal{N}_1 \dot{\cup} \mathcal{N}_2 \dot{\cup} \mathcal{N}_S = \mathcal{N}$,
  $\#\mathcal{N}_S$ small
  $\#\mathcal{N}_1 \approx \#\mathcal{N}_2$
  $\mathcal{N}_1$ and $\mathcal{N}_2$ are not connected.

Remarks

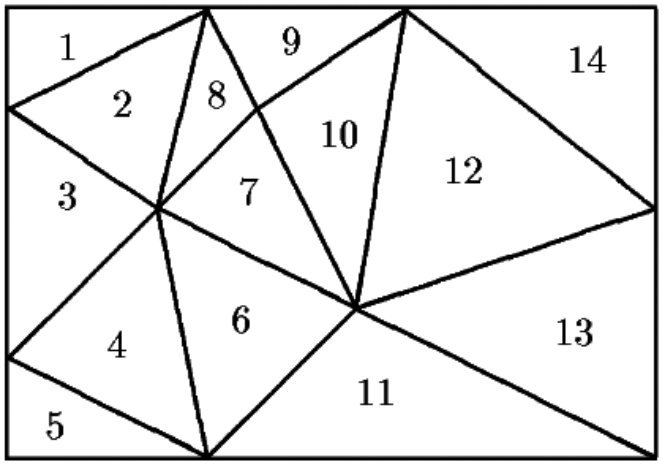
- $\#\left\{e_{i,j} \in \mathcal{E} \middle| v_i \in \mathcal{N}_1 \text{ and } v_j \in \mathcal{N}_2\right\}$ or $\sum_{e_{i,j} \in \mathcal{E}, v_i \in \mathcal{N}_k, v_j \in \mathcal{N}_l \text{ with } l \neq k} w_e\left(e_{i,j}\right)$ respectively, is called cut-size.
- subset of edges separating the graph: edge separator.
- Also vertex separator possible: Find $\mathcal{N}_s \subset \mathcal{N}$ such that
  $\mathcal{N}_1 \dot{\cup} \mathcal{N}_2 \dot{\cup} \mathcal{N}_S = \mathcal{N}$,
  $\#\mathcal{N}_S$ small
  $\#\mathcal{N}_1 \approx \#\mathcal{N}_2$
  $\mathcal{N}_1$ and $\mathcal{N}_2$ are not connected.

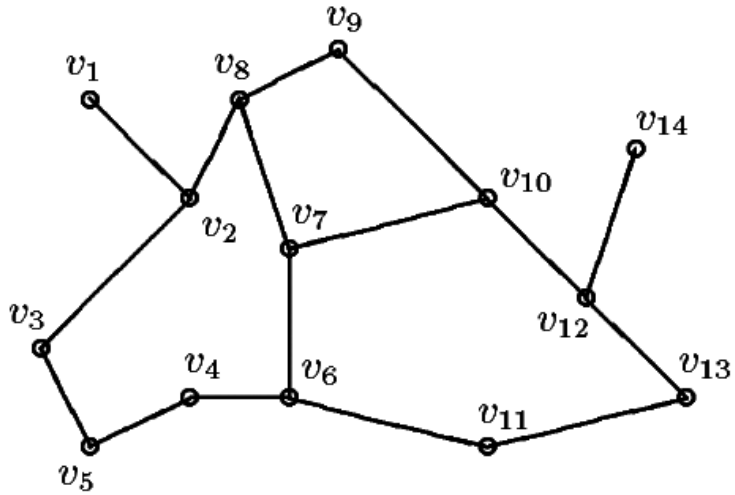
Remarks

- $\#\left\{e_{i,j} \in \mathcal{E} \middle| v_i \in \mathcal{N}_1 \text{ and } v_j \in \mathcal{N}_2\right\}$ or $\sum_{e_{i,j} \in \mathcal{E}, v_i \in \mathcal{N}_k, v_j \in \mathcal{N}_l \text{ with } l \neq k} w_e\left(e_{i,j}\right)$ respectively, is called cut-size.
- subset of edges separating the graph: edge separator.
- Also vertex separator possible: Find $\mathcal{N}_s \subset \mathcal{N}$ such that $\mathcal{N}_1 \dot{\cup} \mathcal{N}_2 \dot{\cup} \mathcal{N}_S = \mathcal{N}$,
  $\#\mathcal{N}_S$ small
  $\#\mathcal{N}_1 \approx \#\mathcal{N}_2$
  $\mathcal{N}_1$ and $\mathcal{N}_2$ are not connected.

PDEs: Finite element methods

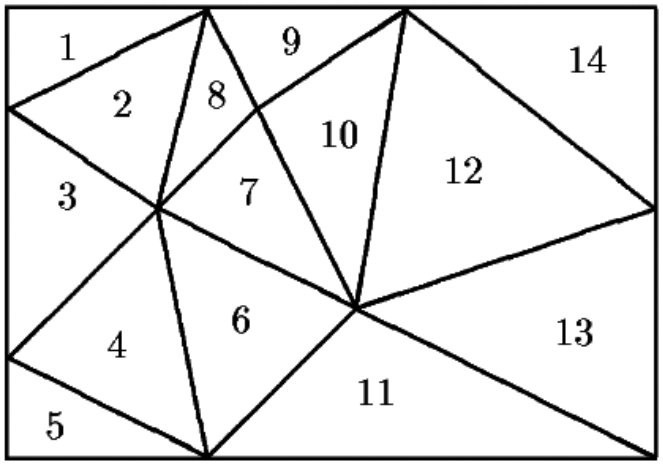
- Distribution of work: partitioning grid into subgrids
- communication: edges of dependency graph
- problem: partition dependency graph (dual graph)

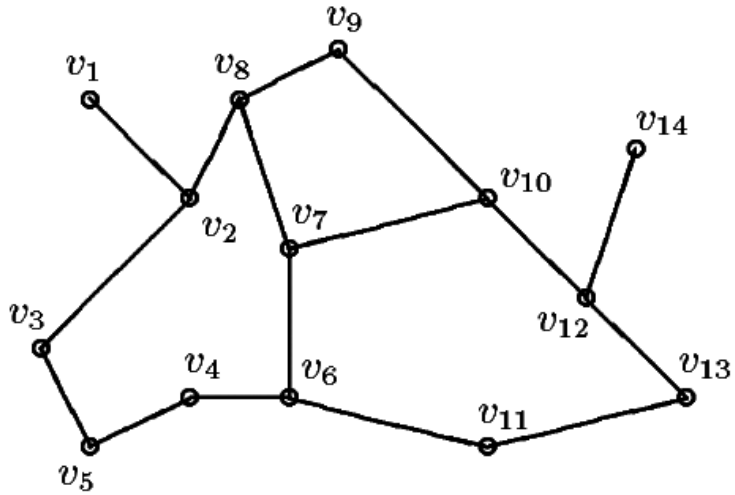




PDEs: Finite element methods

- Distribution of work: partitioning grid into subgrids
- communication: edges of dependency graph
- problem: partition dependency graph (dual graph)





Sparse Matrix-Vector Multiplication I

- for simplification: symmetric sparse matrix $A \in \mathbb{R}^{n \times n}$.
- Calculate $y = Ax$: $y_i = \sum_{j:a_{ij} \neq 0} a_{ij} x_j$.
- Distribute A row-wise and x correspondingly.
- Minimise occurence of: i-th row if A is stored on a processor, $a_{ij} \neq 0$, but j-th row is not.
- Graph partitioning problem:
  n vertices $v_1, \ldots, v_n$ for each row, edge between v_i & v_j $(i \neq j)$ if $a_{ij} \neq 0$.
  $w_v(v_i) = \#$non-zero elements in i-th row.
  $w_e\left(e_{i,j}\right) = 1$.

Sparse Matrix-Vector Multiplication I

- for simplification: symmetric sparse matrix $A \in \mathbb{R}^{n \times n}$.
- Calculate $y = Ax$: $y_i = \sum_{j:a_{ij} \neq 0} a_{ij} x_j$.
- Distribute A row-wise and x correspondingly.
- Minimise occurence of: i-th row if A is stored on a processor, $a_{ij} \neq 0$, but j-th row is not.
- Graph partitioning problem:
  n vertices $v_1, \ldots, v_n$ for each row, edge between v_i & v_j $(i \neq j)$ if $a_{ij} \neq 0$.
  $w_v(v_i) = \#$non-zero elements in i-th row.
  $w_e\left(e_{i,j}\right) = 1$.

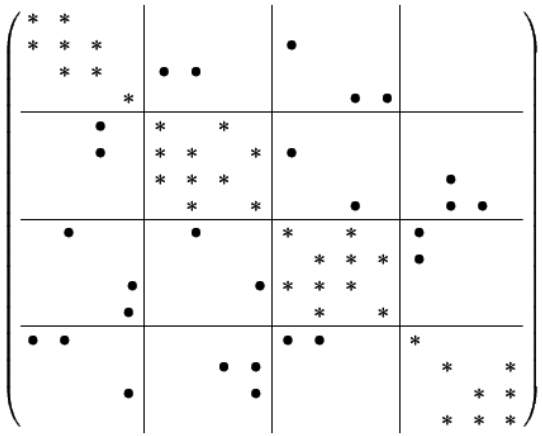
Sparse Matrix-Vector Multiplication I

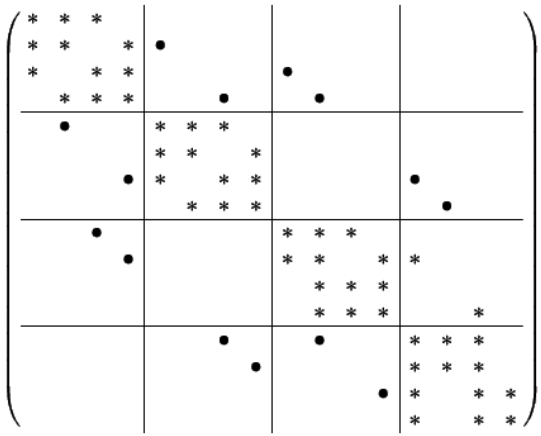
- for simplification: symmetric sparse matrix $A \in \mathbb{R}^{n \times n}$.
- Calculate $y = Ax$: $y_i = \sum_{j:a_{ij} \neq 0} a_{ij} x_j$.
- Distribute A row-wise and x correspondingly.
- Minimise occurence of: i-th row if A is stored on a processor, $a_{ij} \neq 0$, but j-th row is not.
- Graph partitioning problem:
  n vertices $v_1, \ldots, v_n$ for each row, edge between v_i & v_j ($i \neq j$) if $a_{ij} \neq 0$.
  $w_v(v_i) = \#$non-zero elements in i-th row.
  $w_e(e_{i,j}) = 1$.

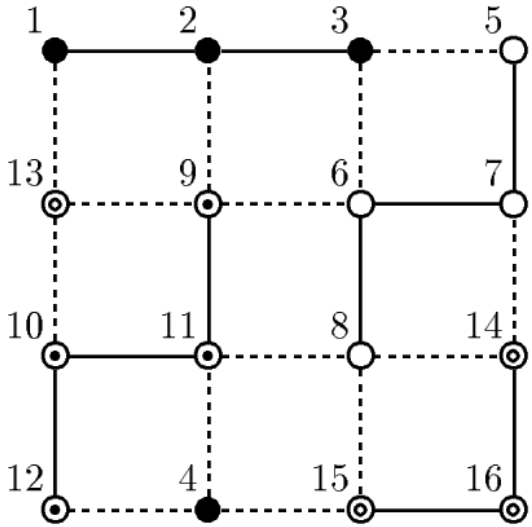
Sparse Matrix-Vector Multiplication I

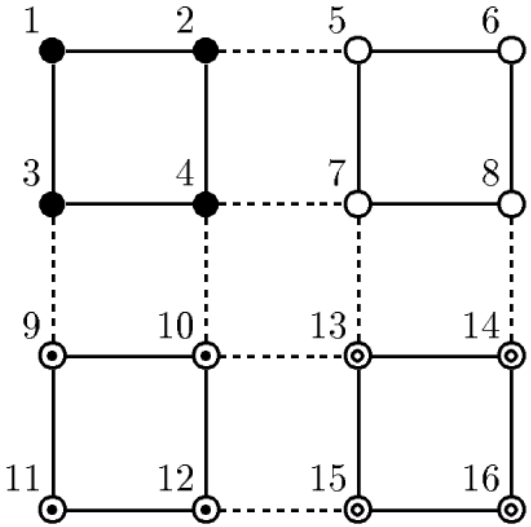
- for simplification: symmetric sparse matrix $A \in \mathbb{R}^{n \times n}$.
- Calculate $y = Ax$: $y_i = \sum_{j:a_{ij} \neq 0} a_{ij} x_j$.
- Distribute A row-wise and x correspondingly.
- Minimise occurence of: i-th row if A is stored on a processor, $a_{ij} \neq 0$, but j-th row is not.
- Graph partitioning problem:
  n vertices $v_1, \ldots, v_n$ for each row, edge between v_i & v_j ($i \neq j$) if $a_{ij} \neq 0$.
  $w_v(v_i) = \#$non-zero elements in i-th row.
  $w_e\left(e_{i,j}\right) = 1$.

Sparse Matrix-Vector Multiplication II









Other applications

- Assign components of electronic circuits to circuit boards such that number of connections between boards is minimised.
- Hypertext browsing
- Network layout
- Geographic information services
- Physical mapping of DNA

Other applications

- Assign components of electronic circuits to circuit boards such that number of connections between boards is minimised.
- Hypertext browsing
- Network layout
- Geographic information services
- Physical mapping of DNA

Other applications

- Assign components of electronic circuits to circuit boards such that number of connections between boards is minimised.
- Hypertext browsing
- Network layout
- Geographic information services
- Physical mapping of DNA

Other applications

- Assign components of electronic circuits to circuit boards such that number of connections between boards is minimised.
- Hypertext browsing
- Network layout
- Geographic information services
- Physical mapping of DNA

Other applications

- Assign components of electronic circuits to circuit boards such that number of connections between boards is minimised.
- Hypertext browsing
- Network layout
- Geographic information services
- Physical mapping of DNA

Algorithms?

- Problem: for non-trivial problems graph partitioning is NP-complete: For a graph $(\mathcal{N}, \mathcal{E})$ no algorithm is known to solve the problem in $\mathcal{O}(k^n)$ time for any $n \in \mathbb{N}$ (polynomial time), where $k = \#\mathcal{N} + \#\mathcal{E}$.
- optimal partition too expensive! $\Rightarrow$ heuristics necessary!
- execution time vs. quality
  dependent on application.
- slightly different partition size $\stackrel{?}{\Rightarrow}$ better cut-size

Algorithms?

- Problem: for non-trivial problems graph partitioning is NP-complete: For a graph $(\mathcal{N}, \mathcal{E})$ no algorithm is known to solve the problem in $\mathcal{O}(k^n)$ time for any $n \in \mathbb{N}$ (polynomial time), where $k = \#\mathcal{N} + \#\mathcal{E}$.
- optimal partition too expensive! $\Rightarrow$ heuristics necessary!
- execution time vs. quality
  dependent on application.
- slightly different partition size $\overset{?}{\Rightarrow}$ better cut-size

Algorithms?

- Problem: for non-trivial problems graph partitioning is NP-complete: For a graph $(\mathcal{N}, \mathcal{E})$ no algorithm is known to solve the problem in $\mathcal{O}(k^n)$ time for any $n \in \mathbb{N}$ (polynomial time), where $k = \#\mathcal{N} + \#\mathcal{E}$.
- optimal partition too expensive! $\Rightarrow$ heuristics necessary!
- execution time vs. quality
  dependent on application.
- slightly different partition size $\overset{?}{\Rightarrow}$ better cut-size

Algorithms?

- Problem: for non-trivial problems graph partitioning is NP-complete: For a graph $(\mathcal{N}, \mathcal{E})$ no algorithm is known to solve the problem in $\mathcal{O}(k^n)$ time for any $n \in \mathbb{N}$ (polynomial time), where $k = \#\mathcal{N} + \#\mathcal{E}$.
- optimal partition too expensive! $\Rightarrow$ heuristics necessary!
- execution time vs. quality
  dependent on application.
- slightly different partition size $\stackrel{?}{\Rightarrow}$ better cut-size

Overview

Overview

- great variety of algorithms (dependent on problem):
  - use of geometric information or only the graph itself
  - local view versus global view
  - deterministic versus random
  - application of graph-theoretic methods versus special case of another problem (e. g. optimisation problem)

Overview

- great variety of algorithms (dependent on problem):
  - use of geometric information or only the graph itself
  - local view versus global view
  - deterministic versus random
  - application of graph-theoretic methods versus special case of another problem (e. g. optimisation problem)

Overview

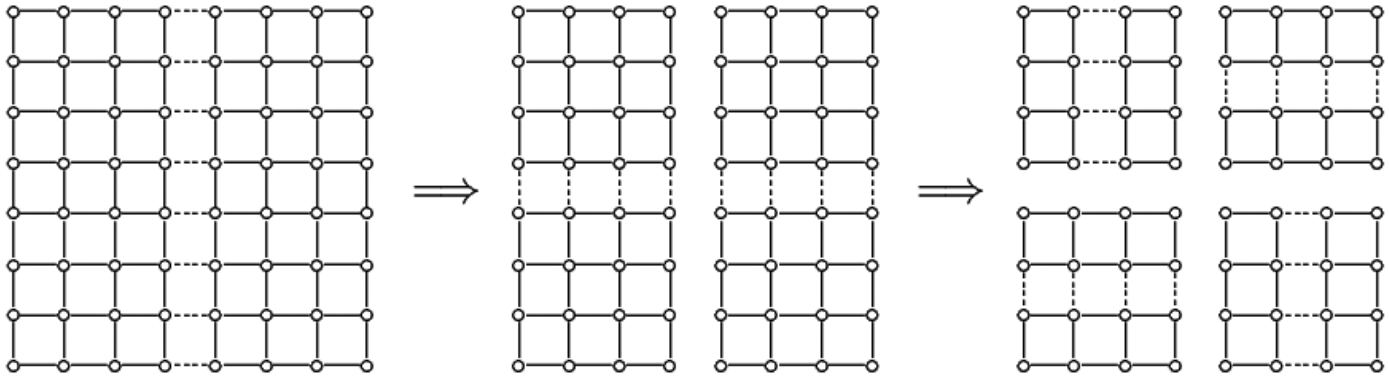
- great variety of algorithms (dependent on problem):
  - use of geometric information or only the graph itself
  - local view versus global view
  - deterministic versus random
  - application of graph-theoretic methods versus special case of another problem (e. g. optimisation problem)

Overview

- great variety of algorithms (dependent on problem):
  - use of geometric information or only the graph itself
  - local view versus global view
  - deterministic versus random
  - application of graph-theoretic methods versus special case of another problem (e. g. optimisation problem)

Recursive Bisection: Idea

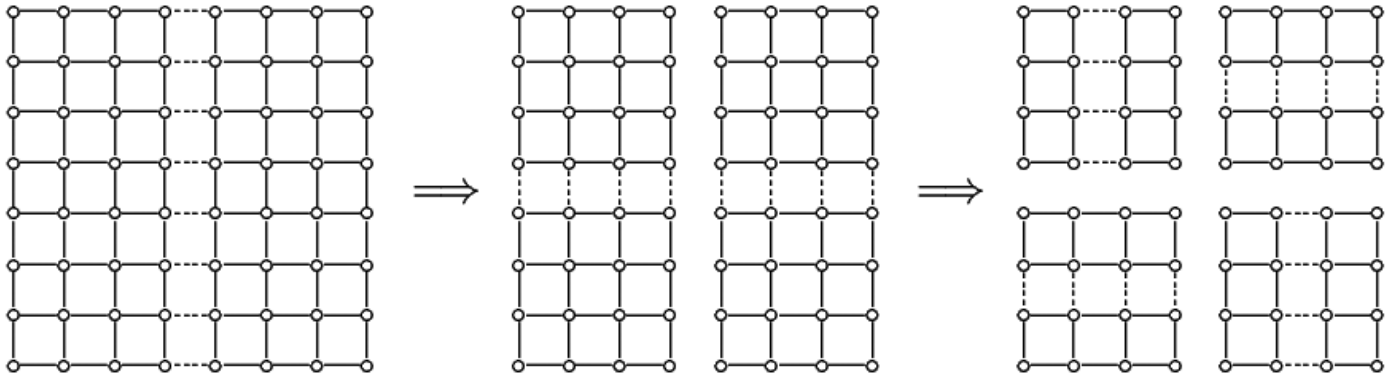
- Most algorithms: designed for graph bisection
- general case: p-partitions with $p = 2^k$ very often (dual-core, quad-core processors)
  $\Rightarrow$ recursion: bisect graph and then bisect subpartitions etc.



- Does it deliver (nearly) the same cut-size as a proper p-way partition?

Recursive Bisection: Idea

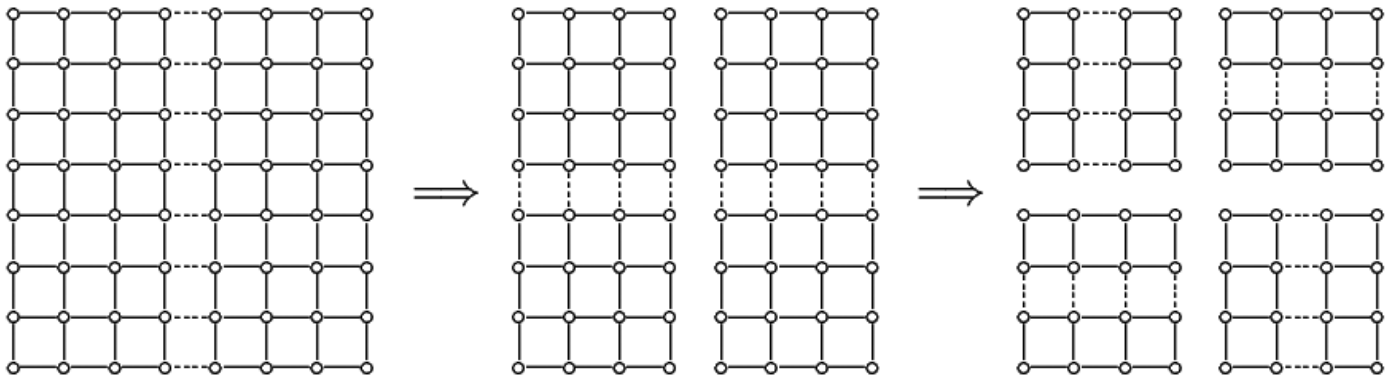
- Most algorithms: designed for graph bisection
- general case: p-partitions with $p = 2^k$ very often (dual-core, quad-core processors)
  $\Rightarrow$ recursion: bisect graph and then bisect subpartitions etc.



- Does it deliver (nearly) the same cut-size as a proper p-way partition?

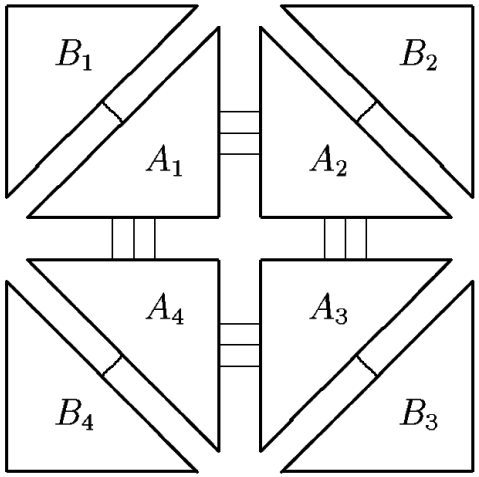
Recursive Bisection: Idea

- Most algorithms: designed for graph bisection
- general case: p-partitions with $p = 2^k$ very often (dual-core, quad-core processors)
  $\Rightarrow$ recursion: bisect graph and then bisect subpartitions etc.



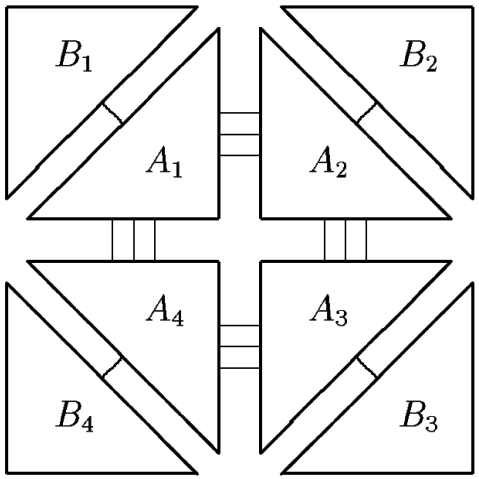
- Does it deliver (nearly) the same cut-size as a proper p-way partition?

Recursive bisection: a counter example I



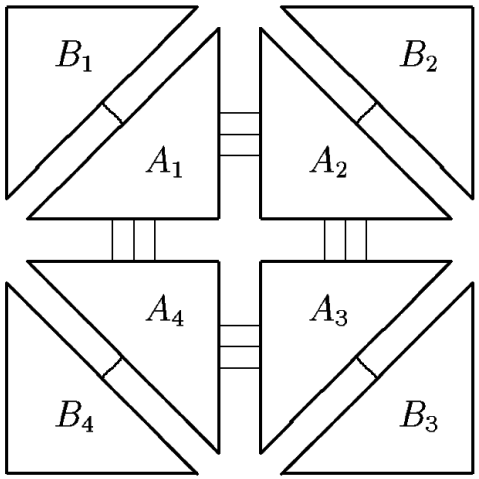
- A_i, B_i are cliques (totally connected), A_i having $\left(\frac{1}{8} + \varepsilon_i\right) n$ vertices, B_i having $\left(\frac{1}{8} - \varepsilon_i\right) n$ with:
  1. $-\frac{1}{8} + \delta \leq \varepsilon_i \leq \frac{1}{8} - \delta$, $\delta > 0$ and $\varepsilon_i \neq 0$.
  2. $\sum_i \varepsilon_i = 0$
  3. $\varepsilon_i + \varepsilon_j \neq 0$ for arbitrary i, j
  4. $\left(\frac{1}{8} \pm \varepsilon_i\right) n \in \mathbb{N}$ for all i

Recursive bisection: a counter example II



- 4-way partition decomposes the grap into $A_i \cup B_i$ ($i = 1, 2, 3, 4$) with total cut-size 12.
- Recursive bisection first decomposes into $\cup_{i=1}^{N} A_i$ and $\cup_{i=1}^{N} B_i$.
  However: In the next step one of the A_i and one of the B_i has to be cut.
  $\Rightarrow$ cut-size $= \mathcal{O}\left(n^2\right)$.

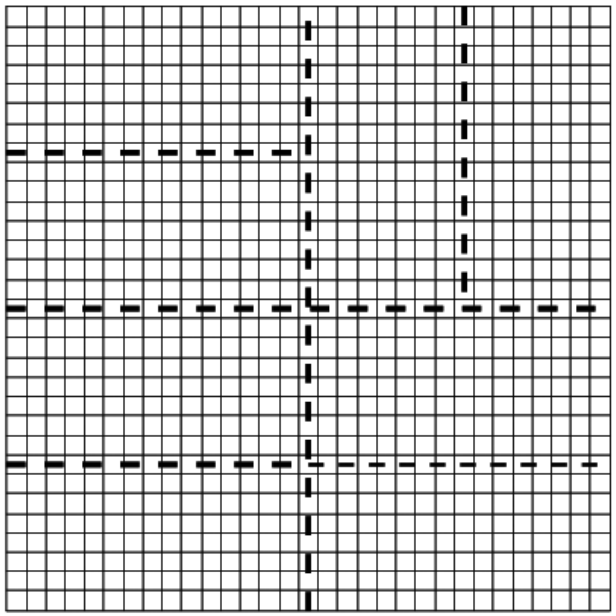
Recursive bisection: a counter example II



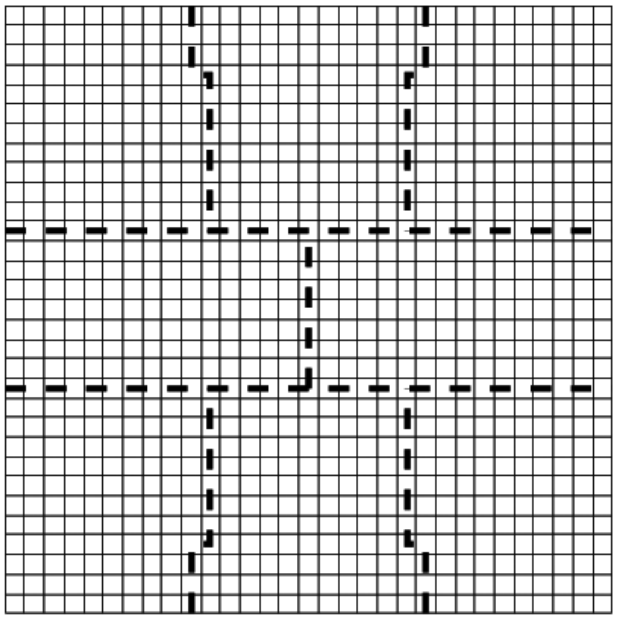
- 4-way partition decomposes the grap into $A_i \cup B_i$ $(i = 1, 2, 3, 4)$ with total cut-size 12.
- Recursive bisection first decomposes into $\cup_{i=1}^{N} A_i$ and $\cup_{i=1}^{N} B_i$.
  However: In the next step one of the A_i and one of the B_i has to be cut.
  $\Rightarrow$ cut-size $= \mathcal{O}\left(n^2\right)$.

Recursive bisection: a counter example III

- A real-life counter example:



a) 8 parts, cut-size 128



b) 8 parts, cut-size 116

Recursive bisection: good news

- For many graphs, recursive bisection is good!
  1. planar graphs: 2-dimensional graph, edges do not intersect
  2. In FEM: graphs with well-shaped simplices
- For such graphs recursive bisection produces factor $\mathcal{O}\left(\sqrt{\frac{\#\mathcal{N}}{p}}\right)$ (case 1) or $\mathcal{O}\left(\left(\frac{\#\mathcal{N}}{p}\right)^{1-1/d}\right)$ (case 2) bigger partitions
- If sizes of subpartitions are allowed to be slightly different $\Rightarrow$ logarithmic increase with factor $\mathcal{O}(\log p)$ (or $\mathcal{O}(\log p \log n)$ with an algorithm running in polynomial time)

Recursive bisection: good news

- For many graphs, recursive bisection is good!
  1. planar graphs: 2-dimensional graph, edges do not intersect
  2. In FEM: graphs with well-shaped simplices
- For such graphs recursive bisection produces factor $\mathcal{O}\left(\sqrt{\frac{\#\mathcal{N}}{p}}\right)$ (case 1) or $\mathcal{O}\left(\left(\frac{\#\mathcal{N}}{p}\right)^{1-1/d}\right)$ (case 2) bigger partitions
- If sizes of subpartitions are allowed to be slightly different $\Rightarrow$ logarithmic increase with factor $\mathcal{O}(\log p)$ (or $\mathcal{O}(\log p \log n)$ with an algorithm running in polynomial time)

Recursive bisection: good news

- For many graphs, recursive bisection is good!
  1. planar graphs: 2-dimensional graph, edges do not intersect
  2. In FEM: graphs with well-shaped simplices
- For such graphs recursive bisection produces factor $\mathcal{O}\left(\sqrt{\frac{\#\mathcal{N}}{p}}\right)$ (case 1) or $\mathcal{O}\left(\left(\frac{\#\mathcal{N}}{p}\right)^{1-1/d}\right)$ (case 2) bigger partitions
- If sizes of subpartitions are allowed to be slightly different $\Rightarrow$ logarithmic increase with factor $\mathcal{O}(\log p)$ (or $\mathcal{O}(\log p \log n)$ with an algorithm running in polynomial time)

Ideas

- graph derived from structure in d-dimensional space
  $\Rightarrow$ geometric information (mesh)
- assumption: vertices close together in euclidian norm
  $\Rightarrow$ close together in mesh
- no use of information about the edges (bad for weighted edges)
- algorithms find structure (e. g. hyperplane) dividing space in two parts.

Ideas

- graph derived from structure in d-dimensional space
  $\Rightarrow$ geometric information (mesh)
- assumption: vertices close together in euclidian norm
  $\Rightarrow$ close together in mesh
- no use of information about the edges (bad for weighted edges)
- algorithms find structure (e. g. hyperplane) dividing space in two parts.

Ideas

- graph derived from structure in d-dimensional space
  $\Rightarrow$ geometric information (mesh)
- assumption: vertices close together in euclidian norm
  $\Rightarrow$ close together in mesh
- no use of information about the edges (bad for weighted edges)
- algorithms find structure (e. g. hyperplane) dividing space in two parts.

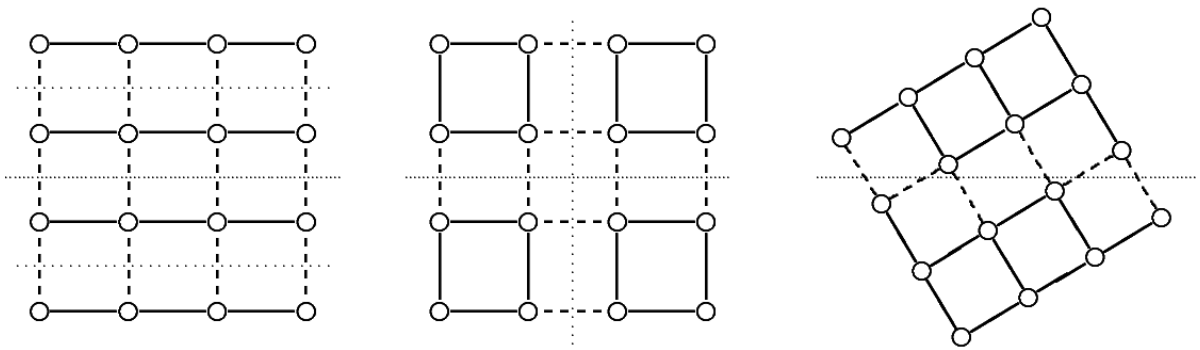
Ideas

- graph derived from structure in d-dimensional space
  $\Rightarrow$ geometric information (mesh)
- assumption: vertices close together in euclidian norm
  $\Rightarrow$ close together in mesh
- no use of information about the edges (bad for weighted edges)
- algorithms find structure (e. g. hyperplane) dividing space in two parts.

Coordniate Bisection

Algorithm: Coordniate bisection

Find hyperplane orthogonal too a chosen axis etc. that divides the points in two equal parts.

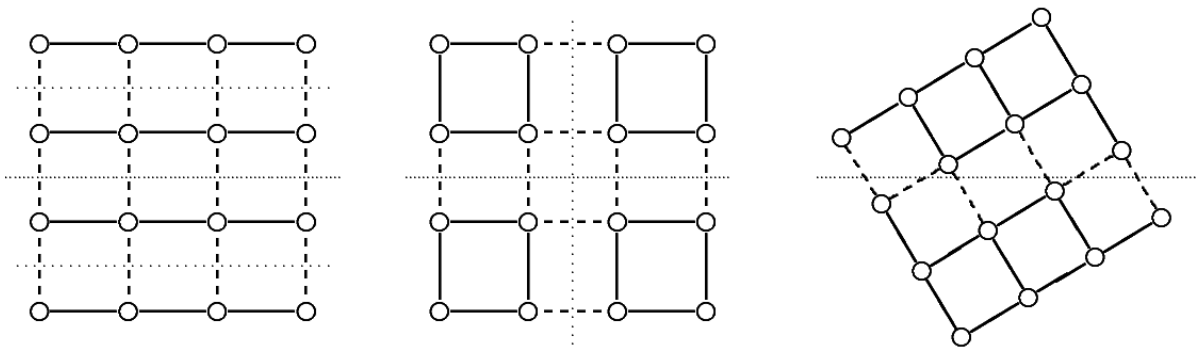


- recursive application: switch between axes $\Rightarrow$ reduction of aspect-ratio
- problem: coordinate dependent

Coordniate Bisection

Algorithm: Coordniate bisection

Find hyperplane orthogonal too a chosen axis etc. that divides the points in two equal parts.

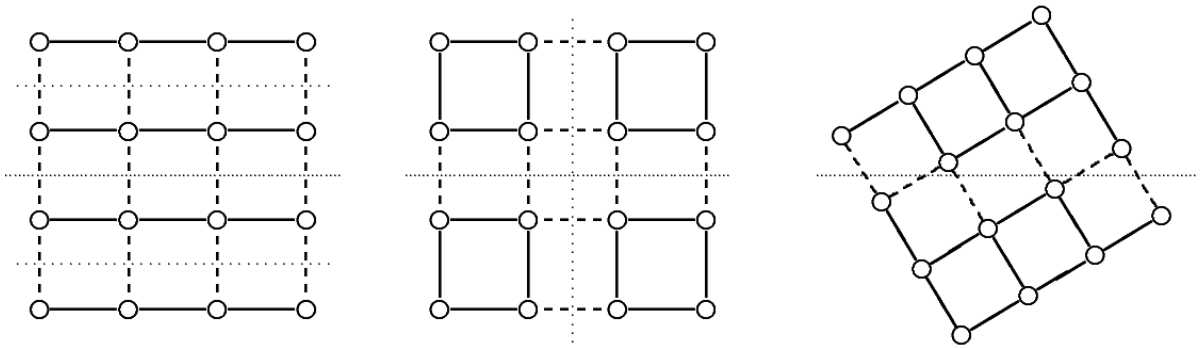


- recursive application: switch between axes $\Rightarrow$ reduction of aspect-ratio
- problem: coordinate dependent

Coordniate Bisection

Algorithm: Coordniate bisection

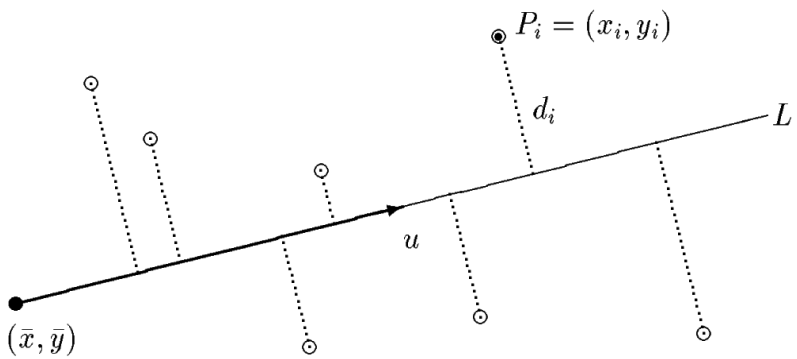
Find hyperplane orthogonal too a chosen axis etc. that divides the points in two equal parts.



- recursive application: switch between axes $\Rightarrow$ reduction of aspect-ratio
- problem: coordinate dependent

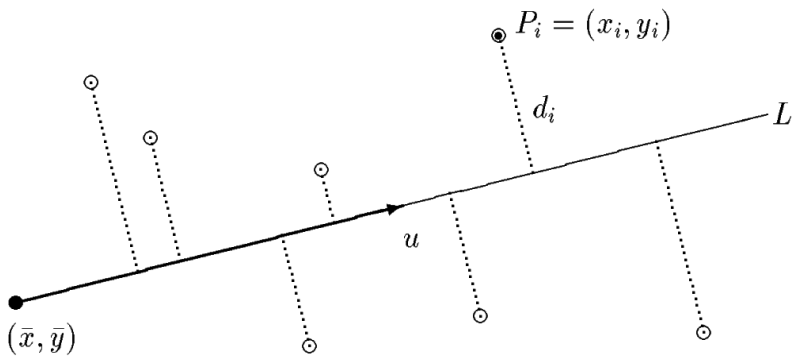
Inertial Bisection

- improvement: find line L and thereby minimise the squared distances d_i of the vertices to the line.
- $L = \left\{ \overline{P} + \alpha u \middle| \alpha \in \mathbb{R} \right\}$ with the center of mass $\overline{P} = (\overline{x}, \overline{y}) = \frac{1}{n} \sum_{i=1}^{n} (x_i, y_i)$ and $|u| = 1$.
- minimise: $\sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n} = \ldots = u^T M u$ with $M_{11} = \sum_i (x_i - \overline{x})^2$, $M_{22} = \sum_i (y_i - \overline{y})^2$ and $M_{12} = M_{21} = -\sum_i (x_i - \overline{x})(y_i - \overline{y})$.
  $\Rightarrow$ calculate eigenvector of M with minimal eigenvalue.



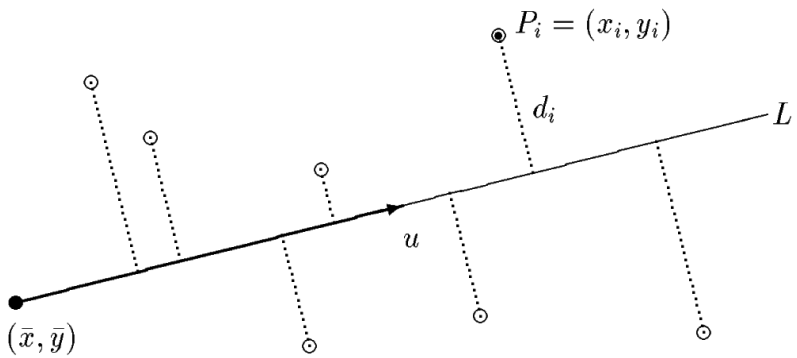
Inertial Bisection

- improvement: find line L and thereby minimise the squared distances d_i of the vertices to the line.
- $L = \left\{ \overline{P} + \alpha u \middle| \alpha \in \mathbb{R} \right\}$ with the center of mass $\overline{P} = (\overline{x}, \overline{y}) = \frac{1}{n} \sum_{i=1}^{n} (x_i, y_i)$ and $|u| = 1$.
- minimise: $\sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n} = \ldots = u^T M u$ with $M_{11} = \sum_i (x_i - \overline{x})^2$, $M_{22} = \sum_i (y_i - \overline{y})^2$ and $M_{12} = M_{21} = -\sum_i (x_i - \overline{x})(y_i - \overline{y})$.
  $\Rightarrow$ calculate eigenvector of M with minimal eigenvalue.



Inertial Bisection

- improvement: find line L and thereby minimise the squared distances d_i of the vertices to the line.
- $L = \left\{ \overline{P} + \alpha u \middle| \alpha \in \mathbb{R} \right\}$ with the center of mass $\overline{P} = (\overline{x}, \overline{y}) = \frac{1}{n} \sum_{i=1}^{n} (x_i, y_i)$ and $|u| = 1$.
- minimise: $\sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n} = \ldots = u^T M u$ with $M_{11} = \sum_i (x_i - \overline{x})^2$, $M_{22} = \sum_i (y_i - \overline{y})^2$ and $M_{12} = M_{21} = -\sum_i (x_i - \overline{x})(y_i - \overline{y})$.
  $\Rightarrow$ calculate eigenvector of M with minimal eigenvalue.



Overview

Ideas

- problem: often geometric information is not available or not meaningful. $\Rightarrow$ only consider connectivity info of the graph!
- Features of spectral bisection:
  - no use of geometric information
  - operates on mathematical object that represents the graph
  - uses floating point operations
  - view on the graph as a whole (global view)

Ideas

- problem: often geometric information is not available or not meaningful. $\Rightarrow$ only consider connectivity info of the graph!
- Features of spectral bisection:
  - no use of geometric information
  - operates on mathematical object that represents the graph
  - uses floating point operations
  - view on the graph as a whole (global view)

Ideas

- problem: often geometric information is not available or not meaningful. $\Rightarrow$ only consider connectivity info of the graph!
- Features of spectral bisection:
  - no use of geometric information
  - operates on mathematical object that represents the graph
  - uses floating point operations
  - view on the graph as a whole (global view)

Ideas

- problem: often geometric information is not available or not meaningful. $\Rightarrow$ only consider connectivity info of the graph!
- Features of spectral bisection:
  - no use of geometric information
  - operates on mathematical object that represents the graph
  - uses floating point operations
  - view on the graph as a whole (global view)

Ideas

- problem: often geometric information is not available or not meaningful. $\Rightarrow$ only consider connectivity info of the graph!
- Features of spectral bisection:
  - no use of geometric information
  - operates on mathematical object that represents the graph
  - uses floating point operations
  - view on the graph as a whole (global view)

Definition

Definition: Laplacian matrix $\mathcal{L}$

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, $n = \#\mathcal{N}$. The Laplacian $\mathcal{L}(\mathcal{G}) = (l_{ij}) \in \mathrm{Sym}_n(\mathbb{R})$ is defined as

$$l_{ij} = \begin{cases} -1 & \text{if } i \neq j \text{ and } e_{i,j} \in \mathcal{E} \\ \deg(v_i) & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

with $\deg(v_i) =$ number of adjacent vertices. The same matrix with zero diagonal: adjacency matrix $\mathcal{A} = \mathcal{A}(\mathcal{G})$

- $\mathcal{L}$ is unique except for permutation of rows and columns (orthogonal trafo)

Definition

Definition: Laplacian matrix $\mathcal{L}$

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, $n = \#\mathcal{N}$. The Laplacian $\mathcal{L}(\mathcal{G}) = (l_{ij}) \in \mathrm{Sym}_n(\mathbb{R})$ is defined as

$$l_{ij} = \begin{cases} -1 & \text{if } i \neq j \text{ and } e_{i,j} \in \mathcal{E} \\ \deg(v_i) & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

with $\deg(v_i) =$ number of adjacent vertices. The same matrix with zero diagonal: adjacency matrix $\mathcal{A} = \mathcal{A}(\mathcal{G})$

- $\mathcal{L}$ is unique except for permutation of rows and columns (orthogonal trafo)

Properties I

Theorem: properties of $\mathcal{L}$

- $\mathcal{L}(\mathcal{G})$ is real, symmetric with eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \lambda_n$ real and orthogonal choosable eigenvectors u_i.
- $\lambda_j \geq 0$, $\mathcal{L}(\mathcal{G})$ is positive semidefinite.
- for $e := \begin{pmatrix} 1 & 1 & \ldots & 1 \end{pmatrix}^T$ we have $\mathcal{L}e = 0$, so $\lambda_1 = 0$ with associated eigenvector e.
- algebraic (= geometric) multiplicity of zero eigenvalue is equal to the number of connected components of $\mathcal{G}$. In particular:

$$\lambda_2 \neq 0 \Leftrightarrow \mathcal{G} \text{ is connected}$$

- λ_2 is called algebraic connectivity of $\mathcal{G}$. $u = u_2$ is called Fiedler vector.

Properties I

Theorem: properties of $\mathcal{L}$

- $\mathcal{L}(\mathcal{G})$ is real, symmetric with eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \lambda_n$ real and orthogonal choosable eigenvectors u_i.
- $\lambda_j \geq 0$, $\mathcal{L}(\mathcal{G})$ is positive semidefinite.
- for $e := \begin{pmatrix} 1 & 1 & \ldots & 1 \end{pmatrix}^T$ we have $\mathcal{L}e = 0$, so $\lambda_1 = 0$ with associated eigenvector e.
- algebraic (= geometric) multiplicity of zero eigenvalue is equal to the number of connected components of $\mathcal{G}$. In particular:

$$\lambda_2 \neq 0 \Leftrightarrow \mathcal{G} \text{ is connected}$$

- λ_2 is called algebraic connectivity of $\mathcal{G}$. $u = u_2$ is called Fiedler vector.

Properties II

- (small) motivation for the pending algorithm:

Theorem (Miroslav Fiedler)

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ (with $\mathcal{N} = \{v_1, v_2, \ldots, v_n\}$) be a connected graph, u be its Fiedler vector. For any $r \geq 0$ we define $\mathcal{N}_1 = \{v_i \in \mathcal{N} | u_i \geq -r\}$. Then the subgraph induced by $\mathcal{N}_1$ is connected. The same holds for $r \leq 0$ and $\mathcal{N}_2 = \{v_i \in \mathcal{N} | u_i \leq -r\}$.

- independent of length and sign of the Fiedler vector.
- Using the Fiedler vector: sensible division of the graph.

Properties II

- (small) motivation for the pending algorithm:

Theorem (Miroslav Fiedler)

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ (with $\mathcal{N} = \{v_1, v_2, \ldots, v_n\}$) be a connected graph, u be its Fiedler vector. For any $r \geq 0$ we define $\mathcal{N}_1 = \{v_i \in \mathcal{N} | u_i \geq -r\}$. Then the subgraph induced by $\mathcal{N}_1$ is connected. The same holds for $r \leq 0$ and $\mathcal{N}_2 = \{v_i \in \mathcal{N} | u_i \leq -r\}$.

- independent of length and sign of the Fiedler vector.
- Using the Fiedler vector: sensible division of the graph.

Properties II

- (small) motivation for the pending algorithm:

Theorem (Miroslav Fiedler)

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ (with $\mathcal{N} = \{v_1, v_2, \ldots, v_n\}$) be a connected graph, u be its Fiedler vector. For any $r \geq 0$ we define $\mathcal{N}_1 = \{v_i \in \mathcal{N} | u_i \geq -r\}$. Then the subgraph induced by $\mathcal{N}_1$ is connected. The same holds for $r \leq 0$ and $\mathcal{N}_2 = \{v_i \in \mathcal{N} | u_i \leq -r\}$.

- independent of length and sign of the Fiedler vector.
- Using the Fiedler vector: sensible division of the graph.

The algorithm

Algorithm: Spectral Bisection

Given a connected Graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$.

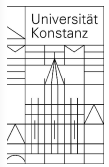
- Number the vertices and calculate $\mathcal{L} = \mathcal{L}(\mathcal{G})$.
- Calculate the second smallest eigenvalue λ_2 and corresponding Fiedler vector u.
- Calculate median m_u of components of u.
- Choose $\mathcal{N}_1 = \{v_i \in \mathcal{N} | u_i < m_u\}$ and $\mathcal{N}_1 = \{v_i \in \mathcal{N} | u_i > m_u\}$ and distribute elements v_i with $u_i = m_u$ so that the partition is balanced.

Formulation as a discrete optimisation problem

- Heuristics: *discrete optimisation problem*, physical models, neural net, ...
- graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ (with $n = \#\mathcal{N}$) to be partitioned into equal sized $\mathcal{N}_1, \mathcal{N}_2$.
- Index vector $x \in \{\pm 1\}^n$ with $x_i = \begin{cases} 1 & \text{if } v_i \in \mathcal{N}_1 \\ -1 & \text{if } v_i \in \mathcal{N}_2 \end{cases}$
- $f(x) := \frac{1}{4} \sum_{(i,j)\in\mathcal{E}} (x_i - x_j)^2$ denotes number of cut edges.
- some calculation: $f(x) = \frac{1}{4} x^T \mathcal{L} x$.

Discrete optimisation problem

Minimise $f(x) = \frac{1}{4} x^T \mathcal{L} x$ under the constraints $x \in \{\pm 1\}^n$ and $x^T e = 0$ (equal sized).



Formulation as a discrete optimisation problem

- Heuristics: *discrete optimisation problem*, physical models, neural net, ...
- graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ (with $n = \#\mathcal{N}$) to be partitioned into equal sized $\mathcal{N}_1, \mathcal{N}_2$.
- Index vector $x \in \{\pm 1\}^n$ with $x_i = \begin{cases} 1 & \text{if } v_i \in \mathcal{N}_1 \\ -1 & \text{if } v_i \in \mathcal{N}_2 \end{cases}$
- $f(x) := \frac{1}{4} \sum_{(i,j)\in\mathcal{E}} (x_i - x_j)^2$ denotes number of cut edges.
- some calculation: $f(x) = \frac{1}{4} x^T \mathcal{L} x$.

Discrete optimisation problem

Minimise $f(x) = \frac{1}{4} x^T \mathcal{L} x$ under the constraints $x \in \{\pm 1\}^n$ and $x^T e = 0$ (equal sized).

Formulation as a discrete optimisation problem

- Heuristics: *discrete optimisation problem*, physical models, neural net, ...
- graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ (with $n = \#\mathcal{N}$) to be partitioned into equal sized $\mathcal{N}_1, \mathcal{N}_2$.
- Index vector $x \in \{\pm 1\}^n$ with $x_i = \begin{cases} 1 & \text{if } v_i \in \mathcal{N}_1 \\ -1 & \text{if } v_i \in \mathcal{N}_2 \end{cases}$
- $f(x) := \frac{1}{4} \sum_{(i,j)\in\mathcal{E}} (x_i - x_j)^2$ denotes number of cut edges.
- some calculation: $f(x) = \frac{1}{4} x^T \mathcal{L} x$.

Discrete optimisation problem

Minimise $f(x) = \frac{1}{4} x^T \mathcal{L} x$ under the constraints $x \in \{\pm 1\}^n$ and $x^T e = 0$ (equal sized).

Formulation as a discrete optimisation problem

- Heuristics: *discrete optimisation problem*, physical models, neural net, ...
- graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ (with $n = \#\mathcal{N}$) to be partitioned into equal sized $\mathcal{N}_1, \mathcal{N}_2$.
- Index vector $x \in \{\pm 1\}^n$ with $x_i = \begin{cases} 1 & \text{if } v_i \in \mathcal{N}_1 \\ -1 & \text{if } v_i \in \mathcal{N}_2 \end{cases}$
- $f(x) := \frac{1}{4} \sum_{(i,j)\in\mathcal{E}} (x_i - x_j)^2$ denotes number of cut edges.
- some calculation: $f(x) = \frac{1}{4} x^T \mathcal{L} x$.

Discrete optimisation problem

Minimise $f(x) = \frac{1}{4} x^T \mathcal{L} x$ under the constraints $x \in \{\pm 1\}^n$ and $x^T e = 0$ (equal sized).

Formulation as a discrete optimisation problem

- Heuristics: *discrete optimisation problem*, physical models, neural net, ...
- graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ (with $n = \#\mathcal{N}$) to be partitioned into equal sized $\mathcal{N}_1, \mathcal{N}_2$.
- Index vector $x \in \{\pm 1\}^n$ with $x_i = \begin{cases} 1 & \text{if } v_i \in \mathcal{N}_1 \\ -1 & \text{if } v_i \in \mathcal{N}_2 \end{cases}$
- $f(x) := \frac{1}{4} \sum_{(i,j)\in\mathcal{E}} (x_i - x_j)^2$ denotes number of cut edges.
- some calculation: $f(x) = \frac{1}{4} x^T \mathcal{L} x$.

Discrete optimisation problem

Minimise $f(x) = \frac{1}{4} x^T \mathcal{L} x$ under the constraints $x \in \{\pm 1\}^n$ and $x^T e = 0$ (equal sized).

Formulation as a discrete optimisation problem

- Heuristics: *discrete optimisation problem*, physical models, neural net, ...
- graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ (with $n = \#\mathcal{N}$) to be partitioned into equal sized $\mathcal{N}_1, \mathcal{N}_2$.
- Index vector $x \in \{\pm 1\}^n$ with $x_i = \begin{cases} 1 & \text{if } v_i \in \mathcal{N}_1 \\ -1 & \text{if } v_i \in \mathcal{N}_2 \end{cases}$
- $f(x) := \frac{1}{4} \sum_{(i,j)\in\mathcal{E}} (x_i - x_j)^2$ denotes number of cut edges.
- some calculation: $f(x) = \frac{1}{4} x^T \mathcal{L} x$.

Discrete optimisation problem

Minimise $f(x) = \frac{1}{4} x^T \mathcal{L} x$ under the constraints $x \in \{\pm 1\}^n$ and $x^T e = 0$ (equal sized).

Continuous optimisation problem

- Still not exactly solvable quickly (NP-complete!)
- idea: not discrete values but continuous values, relax constraint!

Continuous optimisation problem

Minimise $f(z) = \frac{1}{4} z^T \mathcal{L} z$ under the constraints $z^T z = n$ and $z^T e = 0$.

- every discrete solution obeys upper restrictions
  $\Rightarrow$ lower bound for discrete problem,
  hopefully continuous solution good approximation for discrete case

Continuous optimisation problem

- Still not exactly solvable quickly (NP-complete!)
- idea: not discrete values but continuous values, relax constraint!

Continuous optimisation problem

Minimise $f(z) = \frac{1}{4} z^T \mathcal{L} z$ under the constraints $z^T z = n$ and $z^T e = 0$.

- every discrete solution obeys upper restrictions
  $\Rightarrow$ lower bound for discrete problem,
  hopefully continuous solution good approximation for discrete case

Continuous optimisation problem

- Still not exactly solvable quickly (NP-complete!)
- idea: not discrete values but continuous values, relax constraint!

Continuous optimisation problem

Minimise $f(z) = \frac{1}{4} z^T \mathcal{L} z$ under the constraints $z^T z = n$ and $z^T e = 0$.

- every discrete solution obeys upper restrictions
  $\Rightarrow$ lower bound for discrete problem,
  hopefully continuous solution good approximation for discrete case

Solution of continuous problem

- We have an orthonormal basis of eigenvectors $u_1, \ldots, u_n$ with $u_1 = \frac{1}{\sqrt{n}} e$.
- write $z = \sum_{i=1}^{n} \alpha_i u_i$.
- first constraint $\Rightarrow \sum_{i=1}^{n} \alpha_i^2 = n$
- second constraint $\Rightarrow \alpha_1 = 0$
- because in the new basis $\mathcal{L}$ has the form

$$\widetilde{\mathcal{L}} = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \lambda_n \end{pmatrix}$$

, we have $z = \sqrt{n} u_2$ (unique if $\lambda_3 > \lambda_2$) and $f(z) = \frac{1}{4} \lambda_2 n$.

Solution of continuous problem

- We have an orthonormal basis of eigenvectors $u_1, \ldots, u_n$ with $u_1 = \frac{1}{\sqrt{n}} e$.
- write $z = \sum_{i=1}^{n} \alpha_i u_i$.
- first constraint $\Rightarrow \sum_{i=1}^{n} \alpha_i^2 = n$
- second constraint $\Rightarrow \alpha_1 = 0$
- because in the new basis $\mathcal{L}$ has the form

$$\widetilde{\mathcal{L}} = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \lambda_n \end{pmatrix}$$

, we have $z = \sqrt{n} u_2$ (unique if $\lambda_3 > \lambda_2$) and $f(z) = \frac{1}{4} \lambda_2 n$.

Solution of continuous problem

- We have an orthonormal basis of eigenvectors $u_1, \ldots, u_n$ with $u_1 = \frac{1}{\sqrt{n}} e$.
- write $z = \sum_{i=1}^{n} \alpha_i u_i$.
- first constraint $\Rightarrow \sum_{i=1}^{n} \alpha_i^2 = n$
- second constraint $\Rightarrow \alpha_1 = 0$
- because in the new basis $\mathcal{L}$ has the form

$$\widetilde{\mathcal{L}} = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \lambda_n \end{pmatrix}$$

, we have $z = \sqrt{n} u_2$ (unique if $\lambda_3 > \lambda_2$) and $f(z) = \frac{1}{4} \lambda_2 n$.

Solution of continuous problem

- We have an orthonormal basis of eigenvectors $u_1, \ldots, u_n$ with $u_1 = \frac{1}{\sqrt{n}} e$.
- write $z = \sum_{i=1}^{n} \alpha_i u_i$.
- first constraint $\Rightarrow \sum_{i=1}^{n} \alpha_i^2 = n$
- second constraint $\Rightarrow \alpha_1 = 0$
- because in the new basis $\mathcal{L}$ has the form

$$\widetilde{\mathcal{L}} = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \lambda_n \end{pmatrix}$$

, we have $z = \sqrt{n} u_2$ (unique if $\lambda_3 > \lambda_2$) and $f(z) = \frac{1}{4} \lambda_2 n$.

Solution of continuous problem

- We have an orthonormal basis of eigenvectors $u_1, \ldots, u_n$ with $u_1 = \frac{1}{\sqrt{n}} e$.
- write $z = \sum_{i=1}^{n} \alpha_i u_i$.
- first constraint $\Rightarrow \sum_{i=1}^{n} \alpha_i^2 = n$
- second constraint $\Rightarrow \alpha_1 = 0$
- because in the new basis $\mathcal{L}$ has the form

$$\widetilde{\mathcal{L}} = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \lambda_n \end{pmatrix}$$

, we have $z = \sqrt{n} u_2$ (unique if $\lambda_3 > \lambda_2$) and $f(z) = \frac{1}{4} \lambda_2 n$.

Mapping on discrete solution

- How to map continuous solution on discrete solution?
- first method: $x_i = \text{sign}(z_i)$. For $\text{sign}(z_i) = 0$ always choose one partition
  $\Rightarrow$ no balanced partitions in general
- secon method: emphasis on balance: calculate median $\overline{z}$ of the z_i and choose

$$x_i = \begin{cases} -1 & \text{if } z_i < \overline{z} \\ +1 & \text{if } z_i > \overline{z} \end{cases}$$

  Distribute $z_i = \overline{z}$ such that the balance is preserved.
- Chan, Ciarlet and Szeto: for any α we have $x = \text{argmin}_{p \in \{\pm 1\}^n, p^T e = 0} |p - z|_\alpha$.
- No high accuracy for $u = u_2$ necessary!

Mapping on discrete solution

- How to map continuous solution on discrete solution?
- first method: $x_i = \text{sign}(z_i)$. For $\text{sign}(z_i) = 0$ always choose one partition
  $\Rightarrow$ no balanced partitions in general
- secon method: emphasis on balance: calculate median $\overline{z}$ of the z_i and choose

$$x_i = \begin{cases} -1 & \text{if } z_i < \overline{z} \\ +1 & \text{if } z_i > \overline{z} \end{cases}$$

  Distribute $z_i = \overline{z}$ such that the balance is preserved.
- Chan, Ciarlet and Szeto: for any α we have $x = \text{argmin}_{p \in \{\pm 1\}^n, p^T e = 0} |p - z|_\alpha$.
- No high accuracy for $u = u_2$ necessary!

Mapping on discrete solution

- How to map continuous solution on discrete solution?
- first method: $x_i = \text{sign}(z_i)$. For $\text{sign}(z_i) = 0$ always choose one partition
  $\Rightarrow$ no balanced partitions in general
- secon method: emphasis on balance: calculate median $\overline{z}$ of the z_i and choose

$$x_i = \begin{cases} -1 & \text{if } z_i < \overline{z} \\ +1 & \text{if } z_i > \overline{z} \end{cases}$$

  Distribute $z_i = \overline{z}$ such that the balance is preserved.
- Chan, Ciarlet and Szeto: for any α we have
  $x = \text{argmin}_{p \in \{\pm 1\}^n, p^T e = 0} |p - z|_\alpha$.
- No high accuracy for $u = u_2$ necessary!

Spectral bisection of weighted graphs I

- graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ with *weighted edges* $W_{\mathcal{E}}$.
  Define weighted Laplacian $\mathcal{L} = (l_{ij})$ through

$$l_{ij} = \begin{cases} -w_e\left(e_{ij}\right) & \text{if } i \neq j \text{ and } e_{i,j} \in \mathcal{E} \\ \sum_{k=1}^{n} w_e\left(e_{i,k}\right) & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

  $f(x) = \frac{1}{4} x^T \mathcal{L} x$ still denotes weight of the cut edges,
  $x^T e = 0$, $\mathcal{L} e = 0$ etc.

Spectral bisection of weighted graphs II

- *weighted vertices:* weights $W_{\mathcal{N}}$, define $w := \left(w\left(v_i\right)\right)_{i=1}^{n}$ and $V = \operatorname{diag}\left(w\right)$. New problem:

Discrete problem

Minimise $f(x) = \frac{1}{4} x^T \mathcal{L} x$ subject to $x \in \{\pm 1\}^n$ and $x^T V e = 0$.

- Because $x^T V x = \sum_i w_v\left(v_i\right) x_i^2 = \sum_i w_v\left(v_i\right) = e^T V e$ continuous problem:

Continuous problem

Minimise $f(z) = \frac{1}{4} z^T \mathcal{L} z$ subject to $z^T V z = e^T V e$ and $z^T V e = 0$.

- solution: eigenvector corresponding to second smallest eigenvalue of the generalised eigenproblem $\mathcal{L} z = \lambda V z$. Because $V > 0$ equivalent to $\mathcal{B} y = \lambda y$ with $y = V^{1/2} z$ and $\mathcal{B} = V^{-1/2} \mathcal{L} V^{-1/2}$. $\mathcal{B}$ has the same properties as $\mathcal{L}$.

Spectral bisection of weighted graphs II

- *weighted vertices:* weights $W_{\mathcal{N}}$, define $w := \left(w\left(v_i\right)\right)_{i=1}^{n}$ and $V = \operatorname{diag}(w)$. New problem:

Discrete problem

Minimise $f(x) = \frac{1}{4} x^T \mathcal{L} x$ subject to $x \in \{\pm 1\}^n$ and $x^T V e = 0$.

- Because $x^T V x = \sum_i w_v\left(v_i\right) x_i^2 = \sum_i w_v\left(v_i\right) = e^T V e$ continuous problem:

Continuous problem

Minimise $f(z) = \frac{1}{4} z^T \mathcal{L} z$ subject to $z^T V z = e^T V e$ and $z^T V e = 0$.

- solution: eigenvector corresponding to second smallest eigenvalue of the generalised eigenproblem $\mathcal{L} z = \lambda V z$. Because $V > 0$ equivalent to $\mathcal{B} y = \lambda y$ with $y = V^{1/2} z$ and $\mathcal{B} = V^{-1/2} \mathcal{L} V^{-1/2}$. $\mathcal{B}$ has the same properties as $\mathcal{L}$.

Spectral bisection of weighted graphs II

- *weighted vertices:* weights $W_{\mathcal{N}}$, define $w := \left(w\left(v_i\right)\right)_{i=1}^{n}$ and $V = \operatorname{diag}(w)$. New problem:

Discrete problem

Minimise $f(x) = \frac{1}{4} x^T \mathcal{L} x$ subject to $x \in \{\pm 1\}^n$ and $x^T V e = 0$.

- Because $x^T V x = \sum_i w_v\left(v_i\right) x_i^2 = \sum_i w_v\left(v_i\right) = e^T V e$ continuous problem:

Continuous problem

Minimise $f(z) = \frac{1}{4} z^T \mathcal{L} z$ subject to $z^T V z = e^T V e$ and $z^T V e = 0$.

- solution: eigenvector corresponding to second smallest eigenvalue of the generalised eigenproblem $\mathcal{L} z = \lambda V z$. Because $V > 0$ equivalent to $\mathcal{B} y = \lambda y$ with $y = V^{1/2} z$ and $\mathcal{B} = V^{-1/2} \mathcal{L} V^{-1/2}$. $\mathcal{B}$ has the same properties as $\mathcal{L}$.

Spectral quadri- and octasection

- Already noticed: advantageous to divide a graph into more than two parts at once.
- Algorithm to divide graph in 4 or 8 parts at once using 2 or 3 eigenvectors: spectral quadrisection (or octasection):
  - Manhattan metric: weight of edge is multiplied with the number of bits that differ in the binary representation of the graph.
  - quadrisection: partitions $\mathcal{N}_0, \ldots, \mathcal{N}_3$, $x, y \in \{\pm 1\}^n$ with

    $$x_i = -1, y_i = -1 \Leftrightarrow v_i \in \mathcal{N}_0,\ x_i = -1, y_i = +1 \Leftrightarrow v_i \in \mathcal{N}_1, \ldots$$

  - ... Minimise $f(x, y) = \frac{1}{4}\left(x^T \mathcal{L} x + y^T \mathcal{L} y\right)$ under constraints $e^T x = e^T y = x^T y = 0$, $x, y \in \{\pm 1\}^n$.
  - Continuous problem with $z^T z = w^T w = n$ can be solved using linear combination of $\sqrt{n}u_2$ and $\sqrt{n}u_3$.

Spectral quadri- and octasection

- Already noticed: advantageous to divide a graph into more than two parts at once.
- Algorithm to divide graph in 4 or 8 parts at once using 2 or 3 eigenvectors: spectral quadrisection (or octasection):
  - Manhattan metric: weight of edge is multiplied with the number of bits that differ in the binary representation of the graph.
  - quadrisection: partitions $\mathcal{N}_0, \ldots, \mathcal{N}_3$, $x, y \in \{\pm 1\}^n$ with

    $$x_i = -1, y_i = -1 \Leftrightarrow v_i \in \mathcal{N}_0,\ x_i = -1, y_i = +1 \Leftrightarrow v_i \in \mathcal{N}_1, \ldots$$

  - ... Minimise $f(x, y) = \frac{1}{4}\left(x^T \mathcal{L} x + y^T \mathcal{L} y\right)$ under constraints $e^T x = e^T y = x^T y = 0$, $x, y \in \{\pm 1\}^n$.
  - Continuous problem with $z^T z = w^T w = n$ can be solved using linear combination of $\sqrt{n} u_2$ and $\sqrt{n} u_3$.

Spectral quadri- and octasection

- Already noticed: advantageous to divide a graph into more than two parts at once.
- Algorithm to divide graph in 4 or 8 parts at once using 2 or 3 eigenvectors: spectral quadrisection (or octasection):
  - Manhattan metric: weight of edge is multiplied with the number of bits that differ in the binary representation of the graph.
  - quadrisection: partitions $\mathcal{N}_0, \ldots, \mathcal{N}_3$, $x, y \in \{\pm 1\}^n$ with

    $$x_i = -1, y_i = -1 \Leftrightarrow v_i \in \mathcal{N}_0,\ x_i = -1, y_i = +1 \Leftrightarrow v_i \in \mathcal{N}_1, \ldots$$

  - ... Minimise $f(x, y) = \frac{1}{4}\left(x^T \mathcal{L} x + y^T \mathcal{L} y\right)$ under constraints $e^T x = e^T y = x^T y = 0$, $x, y \in \{\pm 1\}^n$.
  - Continuous problem with $z^T z = w^T w = n$ can be solved using linear combination of $\sqrt{n}u_2$ and $\sqrt{n}u_3$.

Spectral quadri- and octasection

- Already noticed: advantageous to divide a graph into more than two parts at once.
- Algorithm to divide graph in 4 or 8 parts at once using 2 or 3 eigenvectors: spectral quadrisection (or octasection):
  - Manhattan metric: weight of edge is multiplied with the number of bits that differ in the binary representation of the graph.
  - quadrisection: partitions $\mathcal{N}_0, \ldots, \mathcal{N}_3$, $x, y \in \{\pm 1\}^n$ with

    $$x_i = -1, y_i = -1 \Leftrightarrow v_i \in \mathcal{N}_0,\ x_i = -1, y_i = +1 \Leftrightarrow v_i \in \mathcal{N}_1, \ldots$$

  - ... Minimise $f(x, y) = \frac{1}{4}\left(x^T \mathcal{L} x + y^T \mathcal{L} y\right)$ under constraints $e^T x = e^T y = x^T y = 0$, $x, y \in \{\pm 1\}^n$.
  - Continuous problem with $z^T z = w^T w = n$ can be solved using linear combination of $\sqrt{n} u_2$ and $\sqrt{n} u_3$.

Spectral quadri- and octasection

- Already noticed: advantageous to divide a graph into more than two parts at once.
- Algorithm to divide graph in 4 or 8 parts at once using 2 or 3 eigenvectors: spectral quadrisection (or octasection):
  - Manhattan metric: weight of edge is multiplied with the number of bits that differ in the binary representation of the graph.
  - quadrisection: partitions $\mathcal{N}_0, \ldots, \mathcal{N}_3$, $x, y \in \{\pm 1\}^n$ with

    $$x_i = -1, y_i = -1 \Leftrightarrow v_i \in \mathcal{N}_0,\ x_i = -1, y_i = +1 \Leftrightarrow v_i \in \mathcal{N}_1, \ldots$$

  - ... Minimise $f(x, y) = \frac{1}{4}\left(x^T \mathcal{L} x + y^T \mathcal{L} y\right)$ under constraints $e^T x = e^T y = x^T y = 0$, $x, y \in \{\pm 1\}^n$.
  - Continuous problem with $z^T z = w^T w = n$ can be solved using linear combination of $\sqrt{n}u_2$ and $\sqrt{n}u_3$.

Spectral quadri- and octasection

- Already noticed: advantageous to divide a graph into more than two parts at once.
- Algorithm to divide graph in 4 or 8 parts at once using 2 or 3 eigenvectors: spectral quadrisection (or octasection):
  - Manhattan metric: weight of edge is multiplied with the number of bits that differ in the binary representation of the graph.
  - quadrisection: partitions $\mathcal{N}_0, \ldots, \mathcal{N}_3$, $x, y \in \{\pm 1\}^n$ with

    $$x_i = -1, y_i = -1 \Leftrightarrow v_i \in \mathcal{N}_0,\ x_i = -1, y_i = +1 \Leftrightarrow v_i \in \mathcal{N}_1, \ldots$$

  - ... Minimise $f(x, y) = \frac{1}{4}\left(x^T \mathcal{L} x + y^T \mathcal{L} y\right)$ under constraints $e^T x = e^T y = x^T y = 0$, $x, y \in \{\pm 1\}^n$.
  - Continuous problem with $z^T z = w^T w = n$ can be solved using linear combination of $\sqrt{n}u_2$ and $\sqrt{n}u_3$.

Ideas

- One of the first algorithms, originally used to optimise placement of el. circuits on circuit boards
- improves given partition:
  1. possibility: choose random partition and improve it using K/L, do this several times, take the best result.
  2. possibility: use result found by other algorithms and improve it.
- K/L has a local view (exchanges neighbouring nodes)
  $\Rightarrow$ complements algorithms with a global view (like spectral partitioning)

Ideas

- One of the first algorithms, originally used to optimise placement of el. circuits on circuit boards
- improves given partition:
  1. possibility: choose random partition and improve it using K/L, do this several times, take the best result.
  2. possibility: use result found by other algorithms and improve it.
- K/L has a local view (exchanges neighbouring nodes) $\Rightarrow$ complements algorithms with a global view (like spectral partitioning)

Ideas

- One of the first algorithms, originally used to optimise placement of el. circuits on circuit boards
- improves given partition:
  1. possibility: choose random partition and improve it using K/L, do this several times, take the best result.
  2. possibility: use result found by other algorithms and improve it.
- K/L has a local view (exchanges neighbouring nodes)
  $\Rightarrow$ complements algorithms with a global view (like spectral partitioning)

Some notations

- $(\mathcal{N}, \mathcal{E}, W_{\mathcal{E}})$ graph, given subpartitions $\mathcal{N}_A \dot{\cup} \mathcal{N}_B = \mathcal{N}$.

Definition: diff-value

$v \in \mathcal{N}_A$: $\operatorname{diff}(v, \mathcal{N}_A, \mathcal{N}_B) := \sum_{b \in \mathcal{N}_B} w_e(e_{v,b}) - \sum_{a \in \mathcal{N}_A} w_e(e_{v,a})$

- amount the cut-size decreases if v is moved to $\mathcal{N}_B$.

Definition: gain-value

$a \in \mathcal{N}_A, b \in \mathcal{N}_B$: $\operatorname{gain}(a, b, \mathcal{N}_A, \mathcal{N}_B) := \operatorname{diff}(a) + \operatorname{diff}(b) - 2w_e(e_{a,b})$

- amount the cut-size decreases if a and b are swapped.

Some notations

- $(\mathcal{N}, \mathcal{E}, W_{\mathcal{E}})$ graph, given subpartitions $\mathcal{N}_A \dot{\cup} \mathcal{N}_B = \mathcal{N}$.

Definition: diff-value

$v \in \mathcal{N}_A:\ \mathrm{diff}(v, \mathcal{N}_A, \mathcal{N}_B) := \sum_{b \in \mathcal{N}_B} w_e(e_{v,b}) - \sum_{a \in \mathcal{N}_A} w_e(e_{v,a})$

- amount the cut-size decreases if v is moved to $\mathcal{N}_B$.

Definition: gain-value

$a \in \mathcal{N}_A, b \in \mathcal{N}_B:\ \mathrm{gain}(a, b, \mathcal{N}_A, \mathcal{N}_B) := \mathrm{diff}(a) + \mathrm{diff}(b) - 2w_e(e_{a,b})$

- amount the cut-size decreases if a and b are swapped.

Some notations

- $(\mathcal{N}, \mathcal{E}, W_{\mathcal{E}})$ graph, given subpartitions $\mathcal{N}_A \dot{\cup} \mathcal{N}_B = \mathcal{N}$.

Definition: diff-value

$v \in \mathcal{N}_A :\ \mathrm{diff}(v, \mathcal{N}_A, \mathcal{N}_B) := \sum_{b \in \mathcal{N}_B} w_e(e_{v,b}) - \sum_{a \in \mathcal{N}_A} w_e(e_{v,a})$

- amount the cut-size decreases if v is moved to $\mathcal{N}_B$.

Definition: gain-value

$a \in \mathcal{N}_A, b \in \mathcal{N}_B :\ \mathrm{gain}(a, b, \mathcal{N}_A, \mathcal{N}_B) := \mathrm{diff}(a) + \mathrm{diff}(b) - 2w_e(e_{a,b})$

- amount the cut-size decreases if a and b are swapped.

The algorithm

Algorithm: Kernigan-Lin (K/L)

- repeat:
  - Compute the diff-values of all vertices, unmark all vertices, Let $k_0 :=$ cut-size.
  - For $i = 1$ to min $(\#\mathcal{N}_A, \#\mathbb{N}_B)$ do:
    - Among all unmarked vertices, find the pair (a_i, b_i) with gain $(a_i, b_i) =$ ismax (< 0 possible) and mark a_i and b_i.
    - For all neighbours v of a_i and b_i do:

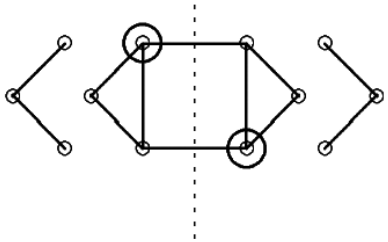
$$\text{diff}(v) := diff(v) + \begin{cases} 2\left[w_e\left(e_{v,a_i}\right) - w_e\left(e_{v,b_i}\right)\right] & \text{for } v \in \mathcal{N}_A \\ 2\left[w_e\left(e_{v,b_i}\right) - w_e\left(e_{v,a_i}\right)\right] & \text{for } v \in \mathcal{N}_B \end{cases}$$

    - $k_i = k_{i-1} -$ gain (a_i, b_i) (cut-size if $a_1, \ldots, a_i$ and $b_1, \ldots, b_i$ had been swapped)
  - Find j such that $k_j = \min_i k_i$.
  - Definie $\mathcal{N}_A := \mathcal{N}_A \setminus \{a_1, \ldots, a_i\} \cup \{b_1, \ldots, b_i\}$, $\mathcal{N}_B := \mathcal{N}_B \setminus \{b_1, \ldots, b_i\} \cup \{a_1, \ldots, a_i\}$.

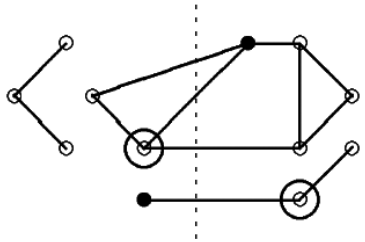
until no further cut-size improvement is achieved.

K/L at work

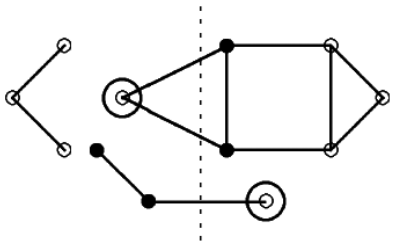
- K/L does not stop as soon as there the gain is negative!
  $\Rightarrow$ ability to climb out of local minima:



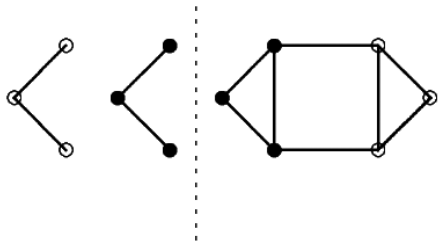
Original Graph, $k_0 = 2$



First swap, $k_1 = 4$



Second swap, $k_2 = 3$

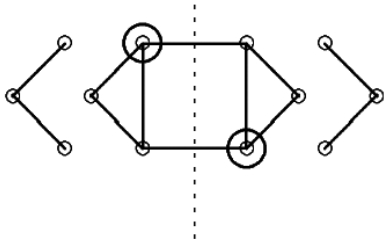


Result, $k_4 = 0$

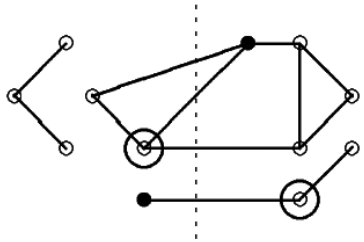
- several improvements and variations possible!

K/L at work

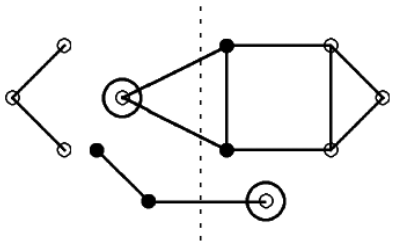
- K/L does not stop as soon as there the gain is negative!
  $\Rightarrow$ ability to climb out of local minima:



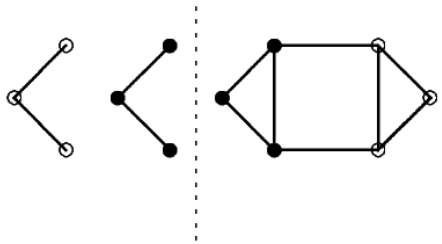
Original Graph, $k_0 = 2$



First swap, $k_1 = 4$



Second swap, $k_2 = 3$

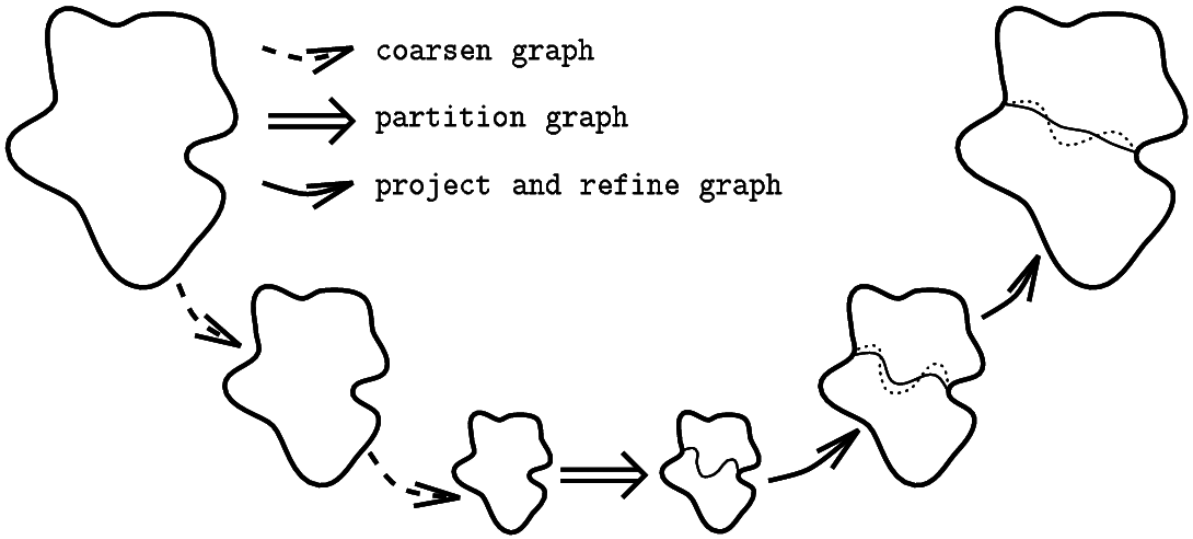


Result, $k_4 = 0$

- several improvements and variations possible!

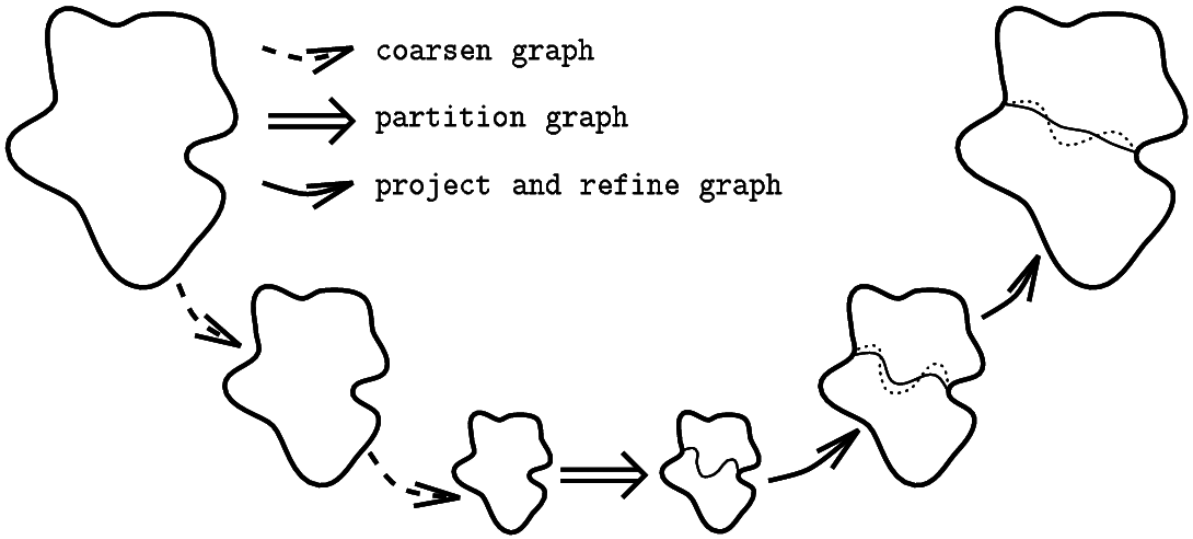
Algorithms not discussed

- more sophisticated geometric partitioning
- Greedy and graphgrowing algorithms
- Multilevel Spectral Bisection
- Multilevel Partitioning



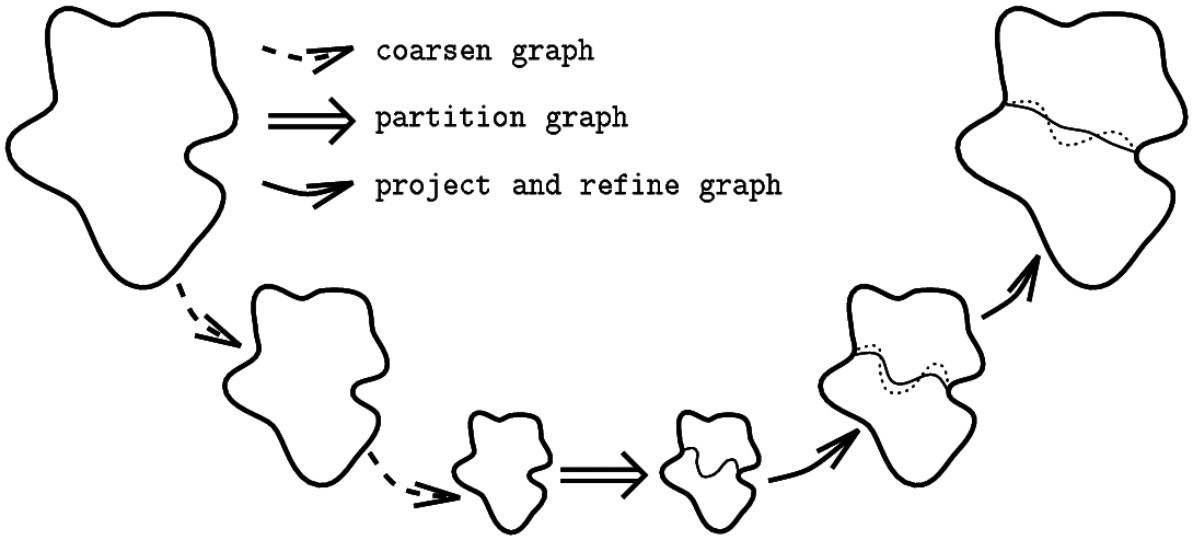
Algorithms not discussed

- more sophisticated geometric partitioning
- Greedy and graphgrowing algorithms
- Multilevel Spectral Bisection
- Multilevel Partitioning



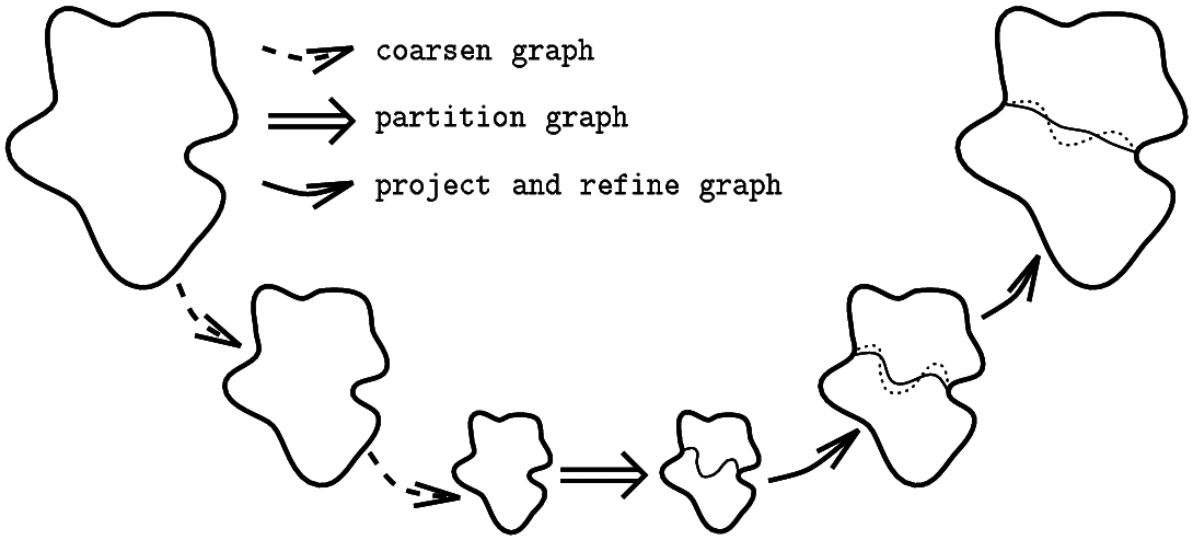
Algorithms not discussed

- more sophisticated geometric partitioning
- Greedy and graphgrowing algorithms
- Multilevel Spectral Bisection
- Multilevel Partitioning



Algorithms not discussed

- more sophisticated geometric partitioning
- Greedy and graphgrowing algorithms
- Multilevel Spectral Bisection
- Multilevel Partitioning



What did we learn?

- distributing work in parallel programs can be often formulated as a graph partitioning problem
- graph partitioning is NP-complete $\Rightarrow$ Heuristics necessary
- Several algorithms portrayed:
  - geometric approaches
  - spectral bisection
  - K/L-algorithm

What did we learn?

- distributing work in parallel programs can be often formulated as a graph partitioning problem
- graph partitioning is NP-complete $\Rightarrow$ Heuristics necessary
- Several algorithms portrayed:
  - geometric approaches
  - spectral bisection
  - K/L-algorithm

What did we learn?

- distributing work in parallel programs can be often formulated as a graph partitioning problem
- graph partitioning is NP-complete $\Rightarrow$ Heuristics necessary
- Several algorithms portrayed:
  - geometric approaches
  - spectral bisection
  - K/L-algorithm

What did we learn?

- distributing work in parallel programs can be often formulated as a graph partitioning problem
- graph partitioning is NP-complete $\Rightarrow$ Heuristics necessary
- Several algorithms portrayed:
  - geometric approaches
  - spectral bisection
  - K/L-algorithm

What did we learn?

- distributing work in parallel programs can be often formulated as a graph partitioning problem
- graph partitioning is NP-complete $\Rightarrow$ Heuristics necessary
- Several algorithms portrayed:
  - geometric approaches
  - spectral bisection
  - K/L-algorithm