Chapter 8

The Mapping Problem

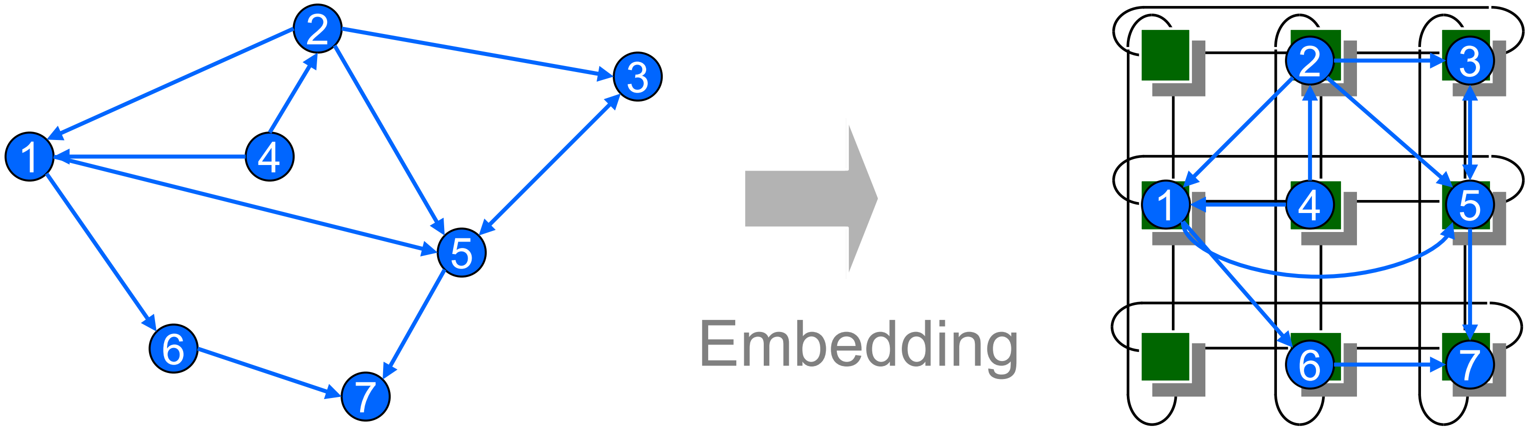
The Mapping or Embedding Problem

- We assume the allocation at program level has already taken place and a partition of the processor network has been allocated to each program.
- Now we have to determine, which process is placed onto which processor.
- If the program is given as a communication graph (TIG), then we have to find a **mapping** of the TIG to the processor connection graph (PCG).
- In an ideal case both TIG and PCG match (graph isomorphism).

Graph Embedding

Placement of threads to processors =
Embedding of thread interaction graph into processor connection graph

Threads are placed so that messages between them get only short delay (short distance).



Parallel program

Parallel machine

8.1 Definitions

- Given two graphs $G = (V_G,E_G)$ and $H = (V_H,E_H)$ with unit edge weight.
- An **embedding** of G into H is a mapping of nodes of G (*guest graph*) to the nodes of H (*host graph*) together with a mapping of the edges of G to paths of H.
- Formally: $\theta = (p_V, p_E)$ with

$$\pi_V : V_G \rightarrow V_H$$

$$\pi_E : E_G \rightarrow \text{Set of paths in } H$$

such that

$$\forall e = (i, j) \in E_G : \quad \pi_E(e) = ((\pi_V(i), v_1), (v_1, v_2), \ldots, (v_{p-1}, \pi_V(j)))$$

- $\theta = (p_V, p_E)$ *is called* **injective** (one-to-one-embedding), if p_V *is* injective.
- Otherwise $\theta = (p_V, p_E)$ *is called* **contractive** (many-to-one-embedding).

More Definitions

- The number of guest nodes that are mapped to a host node $v \in V_H$ is called **load factor** of *v: lf(v).*
- The maximum of load factors over all nodes V_H is called **load factor of the embedding**:

$$lf(\theta) := \max_{v \in V_H} \{lf(v)\}$$

Injective embeddings have a load factor of $lf=1$.

- Edges in *G* ($e \in E_G$) are mapped to paths in *H*. The **edge set of such a path is called** $E(\pi_E(e))$, the **node set** $V(\pi_E(e))$, the set of internal nodes (i.e. without start and end node) is denoted as $VI(\pi_E(e))$.
- The length, i.e. the number of edges of a path to which an edge $e \in E_G$ *is mapped*, is called **dilation** of edge *dil(e).*
- The dilation of the embedding θ is the maximum of all edge dilations:

$$dil(\theta) := \max_{e \in E_G} \{dil(e)\}$$

More Definitions

- With an embedding, multiple paths $\pi_E(e)$, $e \in E_G$ can be routed across the same link $e' \in E_H$. The number of these paths is called **edge congestion** of e'*:*

$$econg(e') := \left|\left\{e \in E_G \middle| e' \in E(\pi_E(e))\right\}\right|$$

- Analogously, we define the **vertex congestion** as the number of paths that have this vertex as an inner node:

$$vcong(v') := \left|\left\{e \in E_G \middle| v' \in VI(\pi_E(e))\right\}\right|$$

- Correspondingly, we define the **edge and vertex congestion** of the embedding as:

$$econg(\theta) := \max_{e' \in E_G} \{econg(e')\}$$

$$vcong(\theta) := \max_{v' \in V_H} \{vcong(v')\}$$

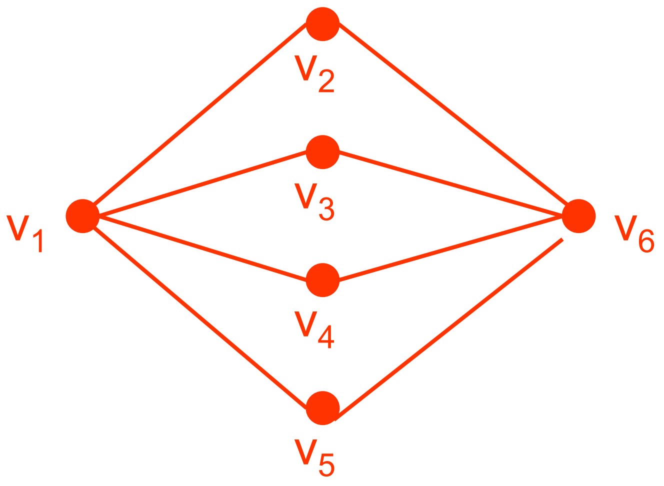
More Definitions

- The **expansion** of an embedding is the ratio of the node numbers of host and guest graph:

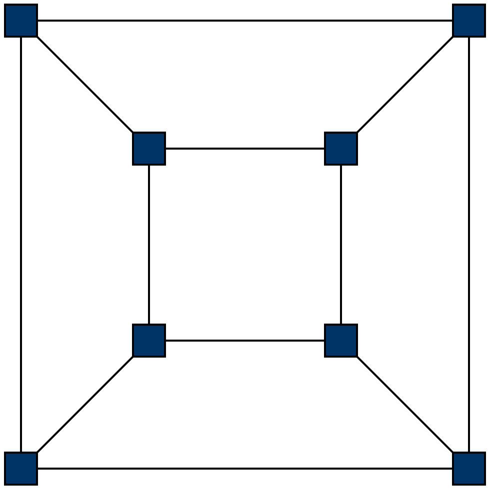
$$exp\,ansion(\theta) := \frac{|V_H|}{|V_G|}$$

- An embedding with a high expansion allows for better dispersion of the paths and therefore leads to smaller edge and vertex congestions.
- The **cardinality** of an embedding *card*(θ) is the number of edges $e \in E_G$, that are one-to-one mapped to edges $e' \in E_H$.

Example for an embedding



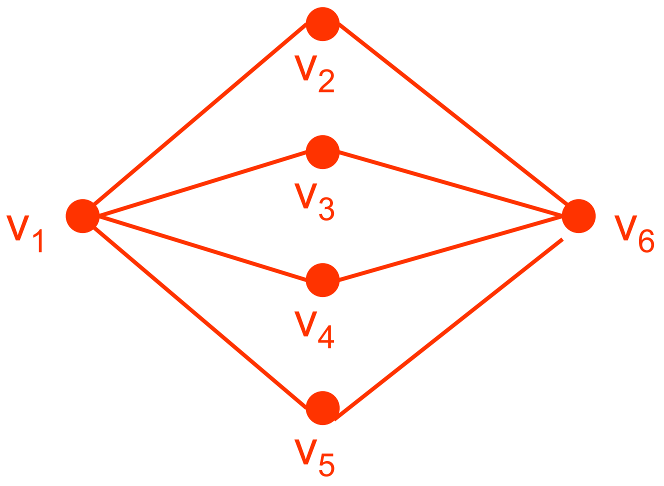
Guest graph G (program)



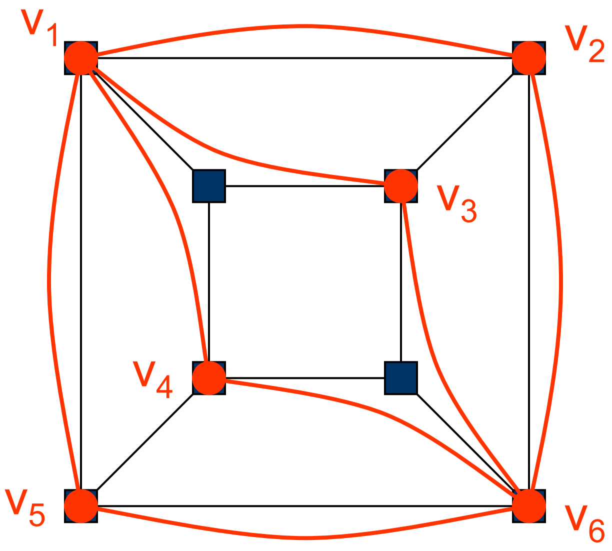
Host graph H (machine)
(3D hypercube)

dil(θ)=?, econg(θ)=?, vcong(θ)=?, expansion(θ)=? and card(θ)=?

Example for an embedding



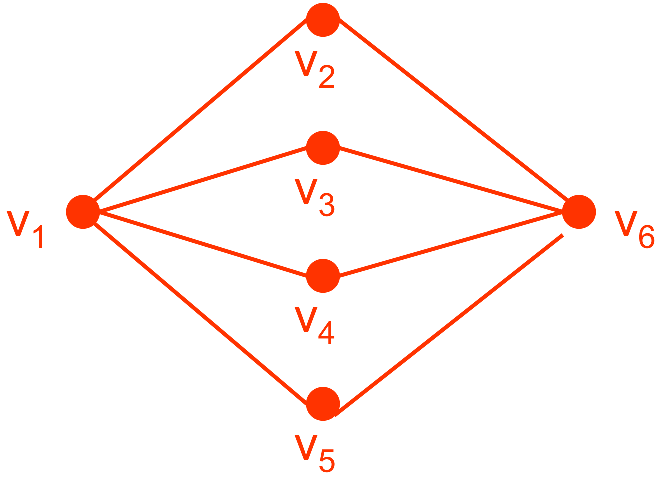
Guest graph G (program)



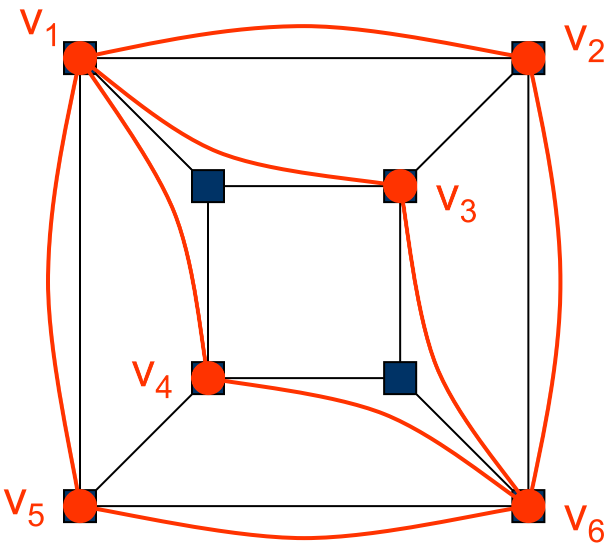
Host graph H (machine)
(3D hypercube)

dil(θ)=?, econg(θ)=?, vcong(θ)=?, expansion(θ)=? and card(θ)=?

Example for an embedding



Guest graph G (program)



Host graph H (machine)
(3D hypercube)

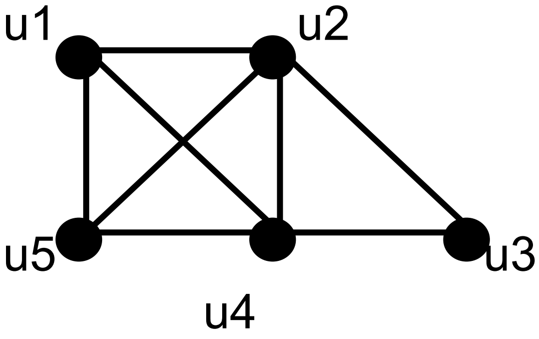
Example for an embedding θ with dil(θ)=2, econg(θ)=2, vcong(θ)=2, expansion(θ)=4/3 and card(θ)=4

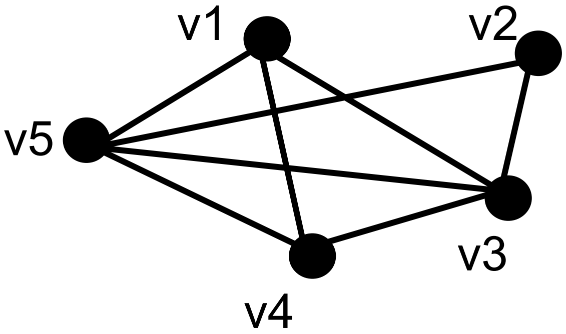
Graph isomorphism

- An embedding $\theta = (\pi_V, \pi_E)$ is called an **isomorphism**, if π_V as well as π_E are bijective and the following holds:

$$\forall (u,v) \in E : \pi_E(u,v) = (\pi_V(u), \pi_V(v))$$

- Are the following graphs **isomorphous**?





- Remark: The graph isomorphism problem (checking, whether two graphs are isomorphous) is in class **NP**. It is still open, whether it is NP-complete.

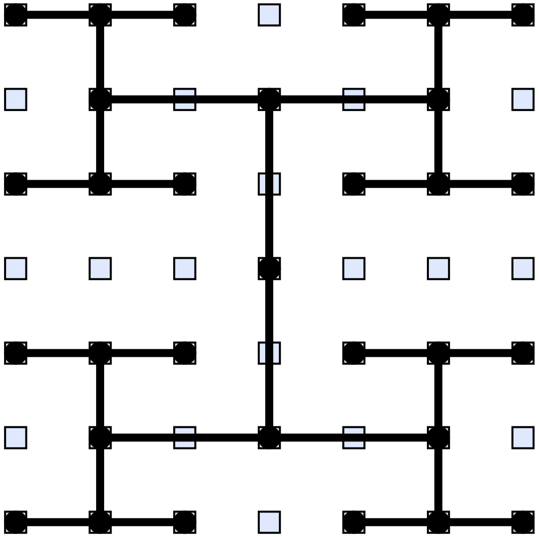
8.2 Injective Embedding

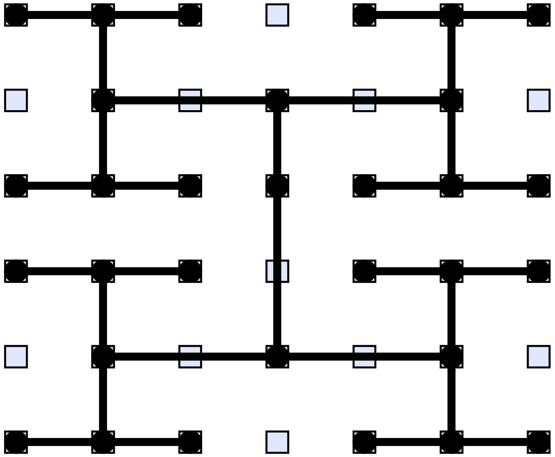
8.2.1 Regular Graphs

- The graph embedding problem is still subject of research.
- Especially for regular graphs (e.g. trees, meshes, hypercubes as guest and host graph) a multitude of solutions is known.
- The goal is usually to find an embedding with minimal **dilation** and/or minimal **expansion**.
- In some cases optimal embeddings are known, sometimes lower bounds can be specified.
- In the following, some results for 2D-meshes as host graphs will be represented.

Binary tree as guest graph

- The usual mapping is the so-called H-embedding (down left).
- The dilation increases linearly with the height *h* of the tree to be embedded.
- Vertex congestion=0, edge congestion=1, cardinality= 2^h+2^{h-1}.
- The expansion goes to 2 for $h \to \infty$.
- A slight improvement is possible by squashing it along one dimension (bottom right):





2D-mesh as guest graph

- With regard to the rectangular mesh *b* x *h* to be embedded, the quadratic mesh of side length *s* is called

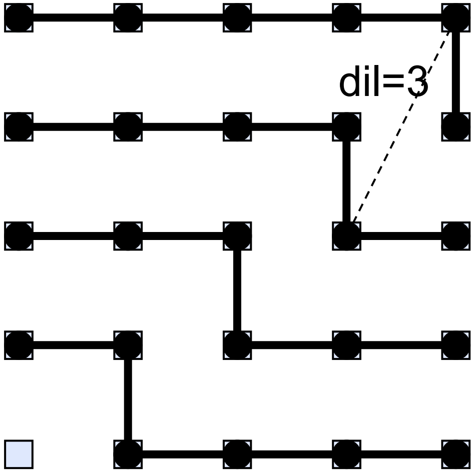
  ideal, if $s = \left\lceil \sqrt{b \cdot h} \right\rceil$

  almost ideal, if $s = \left\lceil \sqrt{b \cdot h} \right\rceil + 1$

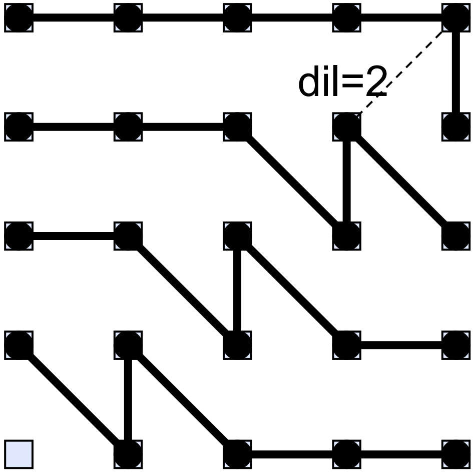
- Each rectangular mesh can be embedded in an almost ideal quadratic mesh with *dil* $\leq$ 3.
- Embeddings with *dil* =2 are possible, but only with higher expansion.
- In the following, we assume w.l.o.g. $b \leq h$. Let ρ := *h/b* define the height-breadth-ratio of the rectangle to be embedded.

2D-mesh as guest graph: step embedding

- The step embedding starts with the first line of the source graph and turns right, as soon as it hits the border or nodes already used.
- The side length of the required square is $s = \lceil (b+h)/2 \rceil$
- For the expansion holds: $\text{expansion}(\theta) \leq (1+\rho)^2 / 4\rho$
- The dilation is *dil*=3.
- With a small modification we achieve that more edges are dilated but the dilation does not exceed 2.



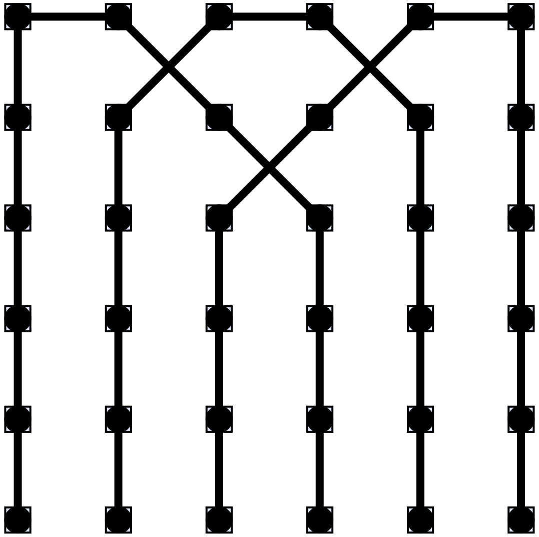
Step embedding



Modified step embedding

2D-mesh as guest graph: folding

- The rectangle is folded along the longer side in serpentine lines into the host graph. At the folding points, a smart layout ensures that a dilation of 2 is not exceeded.
- The folding needs a side length of $s = b\left\lceil \sqrt{\rho} \right\rceil$ and achieves a dilation of dil =2 at an expansion of $\text{expansion}(\theta) = \left\lceil \sqrt{\rho} \right\rceil^2 / \rho$



Further examples

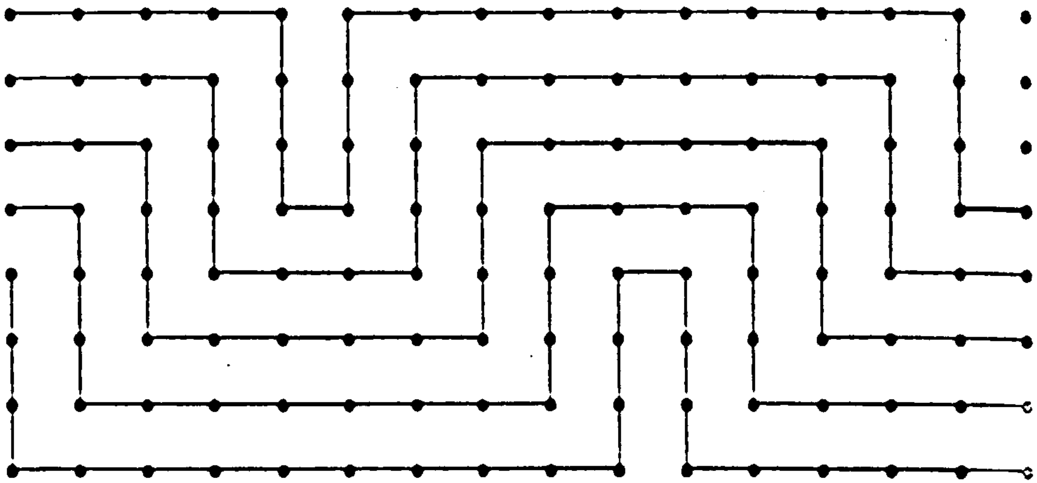


FIG. 4. Embedding of a 5 × 25 mesh into a 8 × 16 mesh by mo[illegible]ed

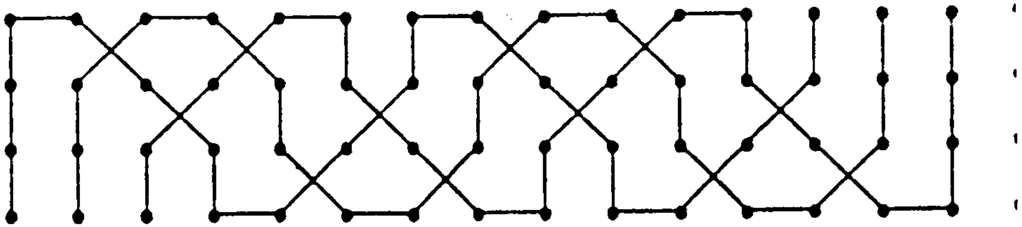


FIG. 5. Embedding of a 3 × 20 mesh into a 4 × 16 mesh by folding.

8.2.2 Irregular Graphs

- If only one of the two graphs, guest or host graph, is irregular, i.e. it does not belong to a graph family, then the embedding problem is in most cases NP-complete.
- To find a solution, we have to apply heuristic algorithms, e.g. dedicated heuristics or general approaches like Simulated Annealing, Genetic Algorithms, Taboo-search,…

8.3 Contractive embedding

- If more threads are to be mapped than processors are available, then we have to contract (group, cluster) some threads that will be mapped to the same processor.
- Those threads should be grouped that intensively communicate with each other.
- The allocation should also try to achieve that all processors receive roughly the same workload (e.g. #threads, #machine instructions).
- Constraints concerning memory capacity can be considered.

8.3.1 Contraction and contraction graph

- Let be G=(V, E) a graph and k a positive integer number.
- A **k-partitioning P** of a graph is a non-void, exhaustive set of pair wise disjoint subsets of V:
- $P = \{V_1, V_2, \ldots, V_k\}, V_i \subseteq V$ with

$$\bigcup_{i=1}^{k} V_i = V$$ „exhaustive“

$$i \neq k \Rightarrow V_i \cap V_k = \varnothing$$ „pair wise disjoint“

- A partitioning decomposes the graph into subgraphs $G_i = (V_i, E_i)$:

$$E_i = \{(u,v) \in E \mid u,v \in V_i\}$$

The edge sets E_i are called **internal edges** of the respective partition.

- Edges that connect vertices of different partitions are called **cutting edges**: $$cut(P) = E - \bigcup_{i=1}^{k} E_i = \{(u,v) \in E \mid u \in V_i, v \in V_j \text{ with } i \neq j\}$$

Contraction graph

- Let *P* = {V_1, V_2,..., V_k} be a k-partitioning of a graph *G=(V,E).*
- The corresponding **contraction graph** $G_P=(V',E')$ consists of one vertex for each partition and edges for any two partitions, if these partitions are connected by edges in *G*:

$$V' = \{v'_1, v'_2, \ldots, v'_k\}$$

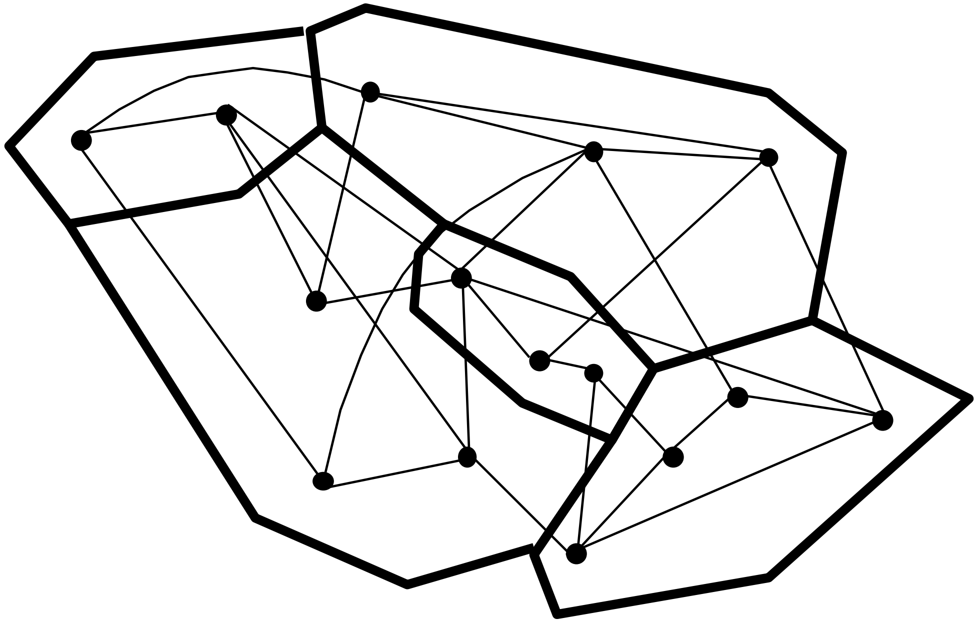
$$(v'_p, v'_q) \in E' \Leftrightarrow \exists v_p \in V_p, v_q \in V_q : (v_p, v_q) \in E, \text{ where } V_p, V_q \in P$$

- Vertex weights of the contraction graph are built by summing up over the vertices contracted:

$$\mu(v'_p) = \sum_{v \in V_p} \mu(v)$$

- Edge weights of the contraction graph are given by the weights of the cutting edges: $\gamma(v'_p, v'_q) = \sum_{u \in V_p, v \in V_q} \gamma(u, v)$

Example

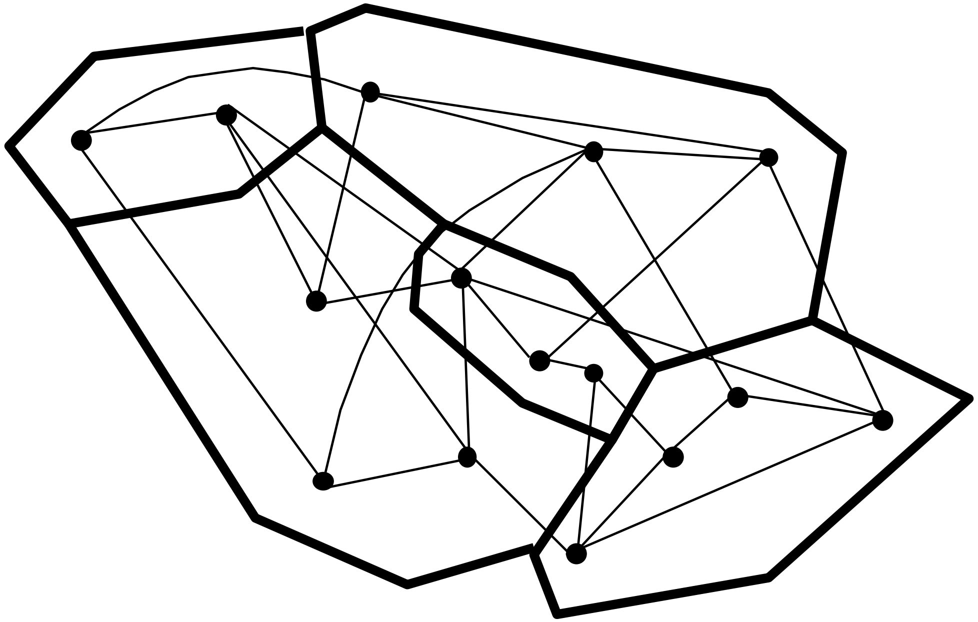


Graph with
5-partitioning

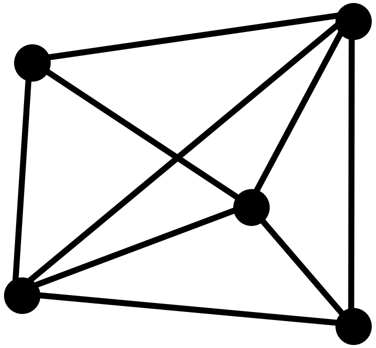
Corresponding
contraction graph ?

Example

Graph with 5-partitioning



Corresponding contraction graph



Contractive allocation

A contractive allocation may consist of two steps :

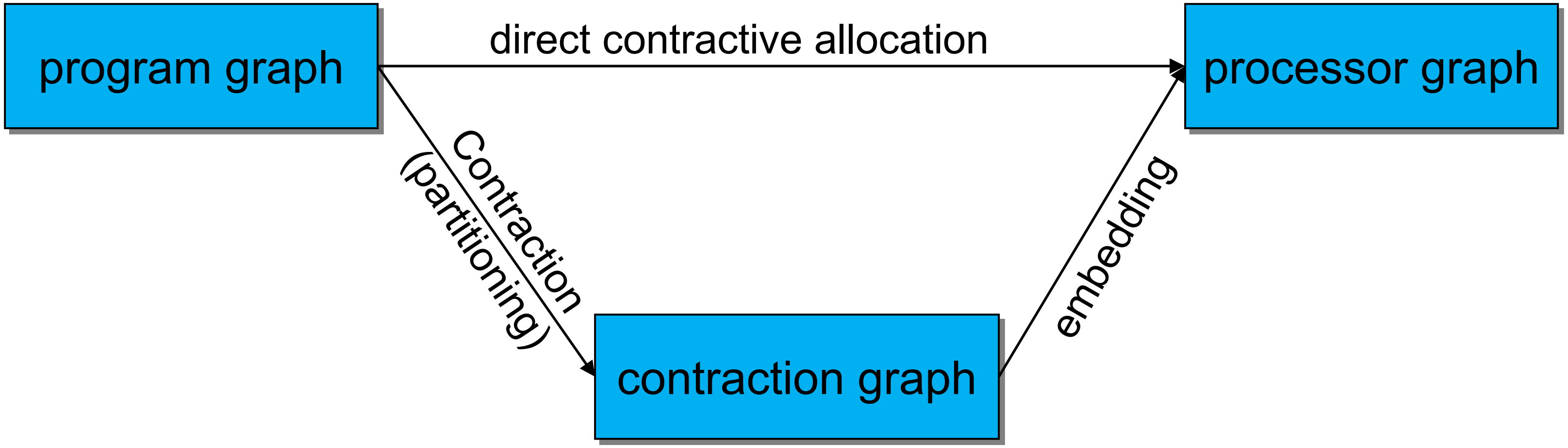
1. Solving the contraction problem (*k*-partitioning), where *k* denotes the number of available processors. The goal is a balanced partitioning with minimal cut costs (edge weights).
2. Embedding the contraction graph into the processor graph, i.e. injective allocation of the thread groups to the processors. The goal is an embedding e.g. with minimal dilation (measure for the maximum latency of inter-thread communication).

Both optimization problems are in general NP-complete.

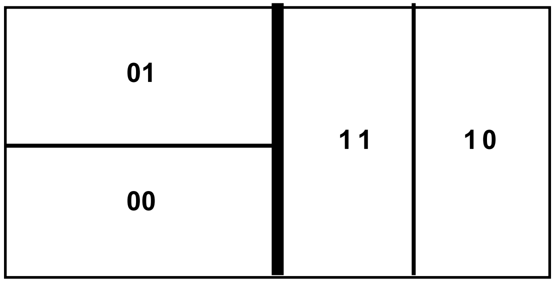
Therefore, we have to resort to heuristic approaches.

Alternative approach

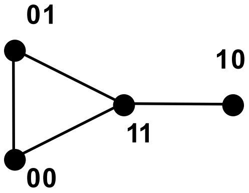
- Instead of proceeding indirectly and first conduct a partitioning, we could directly calculate contractive allocation.
- The indirect approach holds the danger of some loss of optimality since the topology of the processor graphs is not considered in the partitioning.



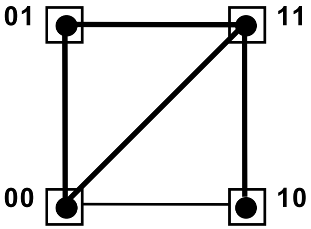
Example



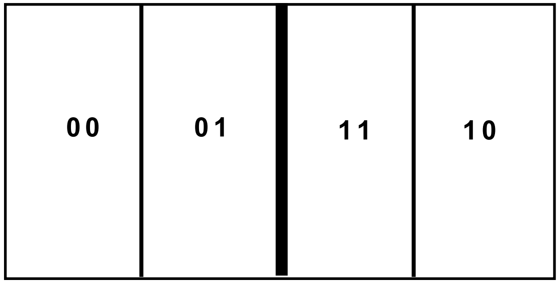
partitioning of problem graph



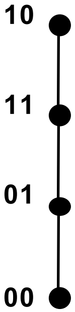
contraction graph



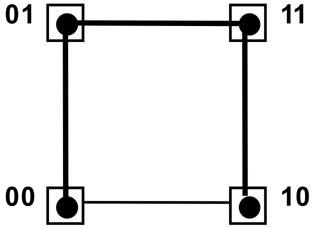
embedding



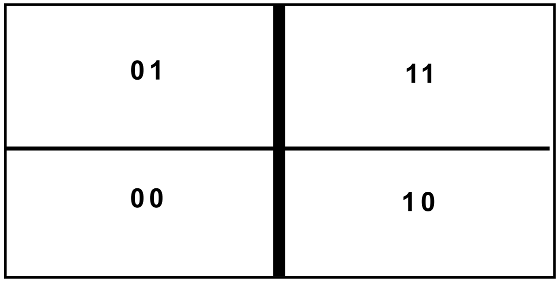
partitioning of problem graph



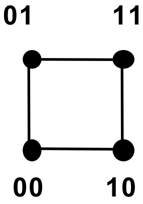
contraction graph



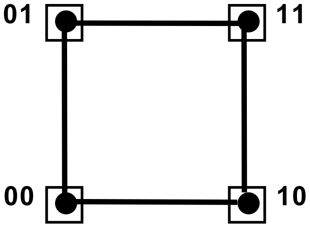
embedding



partitioning of problem graph



contraction graph



embedding

8.3.2 Graph partitioning

- The balanced *k*-partitioning of a graph with minimal cut costs is NP-complete.
- Only for special problem instances there are efficient algorithms:
  - The **bipartitioning** of a graph with minimal cut costs can be performed using flow algorithms (Max-Flow-Min-Cut-Algorithm) in O(n m^2)
    (with n = number of nodes and m = number of edges).
  - In addition, there are some heuristic approaches that perform a bipartitioning **recursively**. By doing so, the *k*-partitioning problem can be solved for $k = 2^i$ efficiently (but not optimally).

Kernighan-Lin-Algorithm

- A frequently employed algorithm for bipartitioning goes back to Kernighan-Lin (1970):

- Starting point is a feasible (=balanced) initial partitioning. By pair wise exchange based on the method of probing paths, we try to improve the cut costs

- Given: Graph with 2*m* vertices and edge weights.
- Goal: two equally sized partitions *X* and *Y* with minimal cut costs:

$$\Gamma(X,Y) := \sum_{u \in X, v \in Y} \gamma(u,v) \rightarrow min$$

Kernighan-Lin-Algorithm

- Let be $v \in X$
- The internal costs *I(v)* of a vertex are the weights of all edges between *v* and other vertices of the same partition:

$$I(v) = \Gamma(v, X) = \sum_{u \in X} \gamma(v, u)$$

- The external costs *E(v)* of a vertex are the weights of all edges that connect v with a vertex of the other partition (cutting edges):

$$E(v) = \Gamma(v, Y) = \sum_{u \in Y} \gamma(v, u)$$

- As difference costs of *v* we define *D(v) = E(v)-I(v).*

Kernighan-Lin-Algorithm

Lemma:

- Let be u$\in X$, $v \in Y$. If u and v are exchanged, we obtain a gain $g = D(u) + D(v) - 2\,\gamma(u,v)$

- Proof:

  Let be *z* the cut costs minus all edges incident with *u* or *v*.

  Then the whole cut costs are

  $$\Gamma(X,Y) = z + E(u) + E(v) - \gamma(u,v).$$

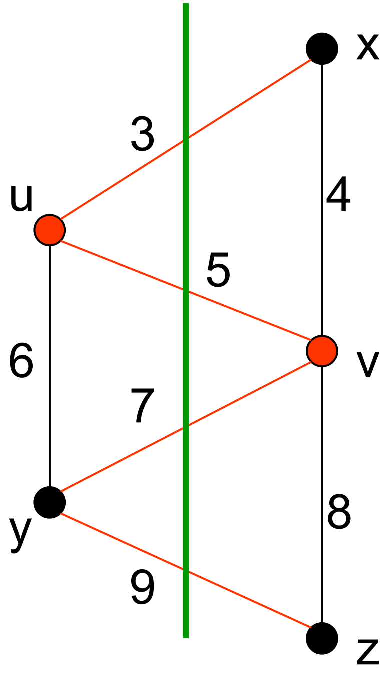
  After exchange of u and v we obtain

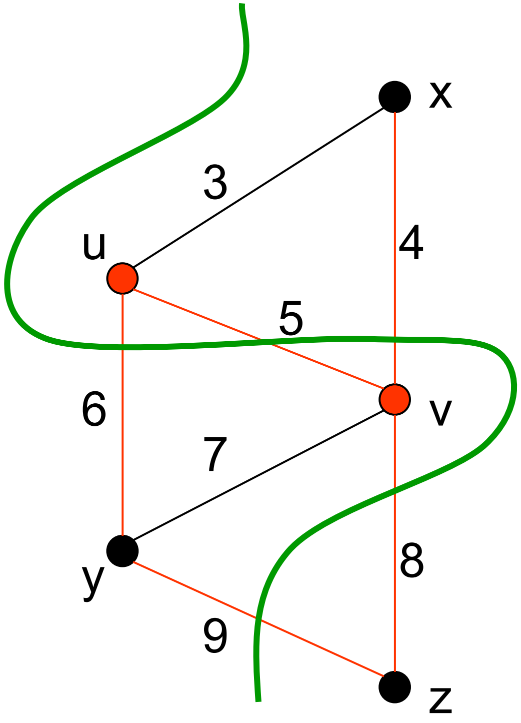
  $$\Gamma(X',Y') = z + I(u) + I(v) + \gamma(u,v).$$

  Taking the difference yields

  $$\begin{aligned} g = \Delta\Gamma &= E(u) + E(v) - \gamma(u,v) \\ &\quad -(I(u) + I(v) + \gamma(u,v)) \\ &= D(u) + D(v) - 2\,\gamma(u,v) \end{aligned}$$

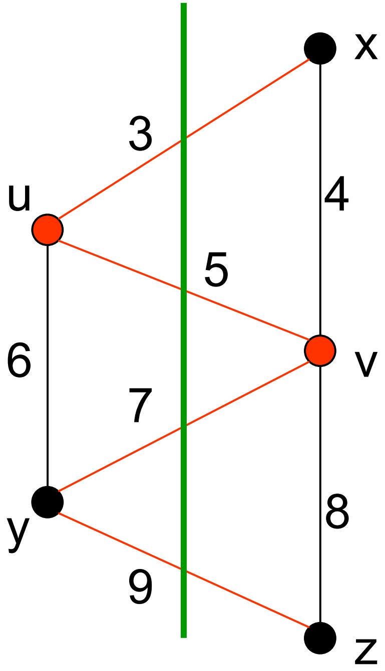
Example





Do we want to exchange u and v?

Example



Cut costs Γ=

γ(u,v) =

E(u) =

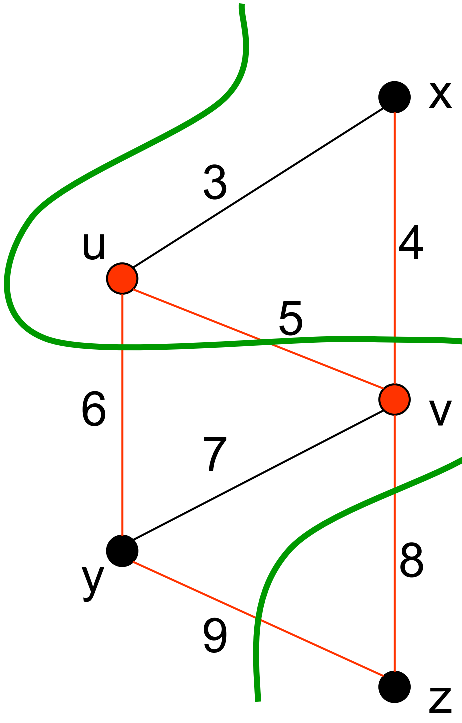
I(u) =

D(u) =

E(v) =

I(v) =

D(v) =

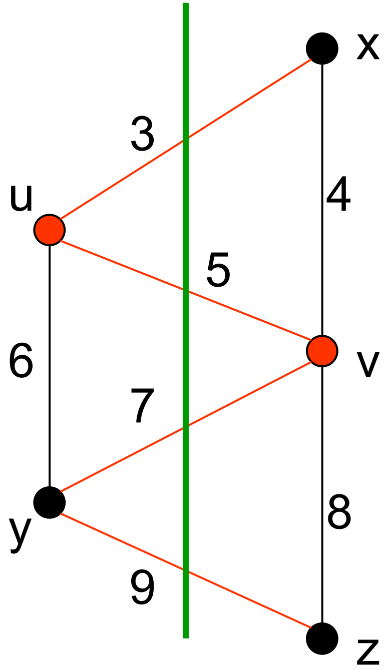


Cut costs Γ =

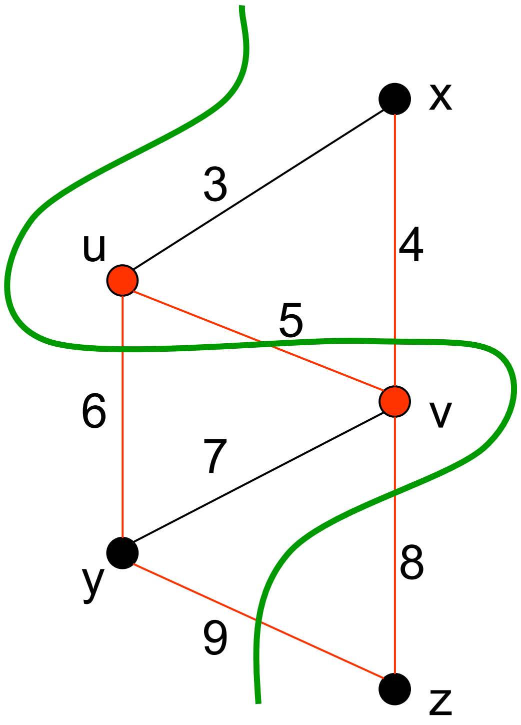
g = D(u)+D(v)-2 γ(u,v)

g =

Example



Cut costs Γ= 3+5+7+9 = 24
γ(u,v) = 5
E(u) = 3+5 = 8
I(u) = 6
D(u) = 2
E(v) = 5+7 = 12
I(v) = 4+8 = 12
D(v) = 0



Cut costs Γ = 4+5+6+8+9 = 32
g = D(u)+D(v)-2 γ(u,v)
g = 2+0-10 = -8

Kernighan-Lin-Algorithm 1

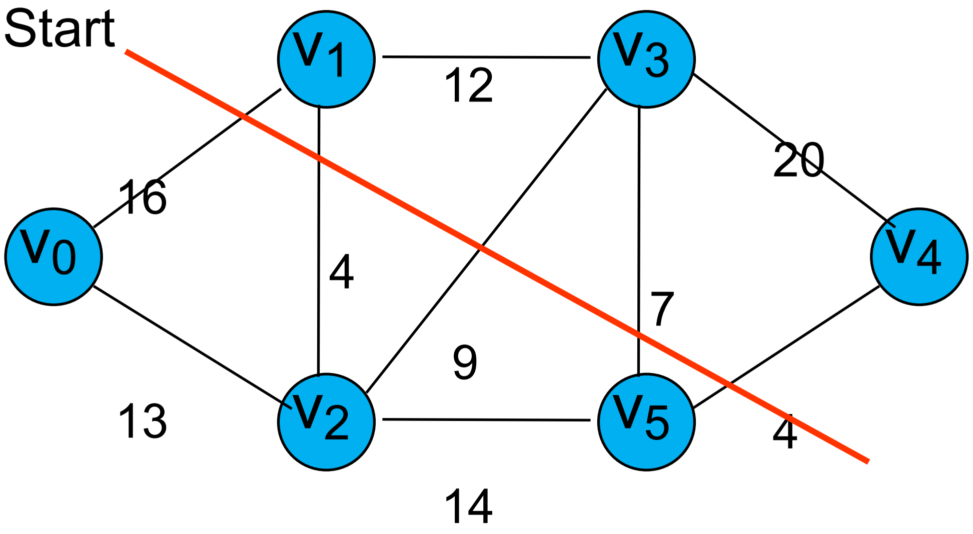
```
mincut(Xin,Yin,Xout,Yout)
X ← Xin;   Y ← Yin                                Init. balanced bipartitioning
G ← ∞                                             Initialize gain
while G > 0 do                                    Search as long as we achieve
  X'← X                                           an improvement
  Y'← Y                                           Auxiliary variable to store
  for i ← 1 to m do                               temporary node movements
    for all v∈V, v unmarked do
      calculate D[v] resp. X',Y'                  Calculation of difference costs
    end for
    for all (u,v) unmarked, u∈X',v∈Y'do           Calculation of gain when
      g[u,v] ← D[u] + D[v]- 2 γ[u,v]              exchanging u and v
    end for
    g[i] ←max{g[u,v]}                             Storing  and marking of that pair
    (u*[i],v*[i])←(u*,v*) with max{g[u,v]}        the exchange of which maximizes
    mark u*,v*                                    the gain.
    X' ← X' -{u*[i]}∪ {v*[i]}                     Temporary exchange of marked pair
    Y' ← Y' -{v*[i]}∪ {u*[i]}
  end for
```

Kernighan-Lin-Algorithm 2

20	G[k] ← $\sum_{j=1}^{k} g[j]$ (k =0,...,n)	Series of m exchange pairs is found. (X'=Y, Y'=X) G[k] indicates the gain accumulated over the first k exchange steps
21	k* ← min k with max {G[k]}	With k* a subsequence optimal with regard to the initial partitioning
22	G ← G[k*]	has been found
23	**if** G > 0	Exchange steps are performed only if
24	**then**	it pays off
25	X←X-{u*[1],..,u*[k*]}∪{v*[1],.,v*[k*]}	Factual exchange
26	Y←Y-{v*[1],..,v*[k*]}∪{u*[1],.,u*[k*]}	
27	**end** then	
28	**end** while	Further exchange steps do not lead to any improvement (G=0),
29	Xout ← X	Algorithm stops.
30	Yout ← Y	
31	**end** mincut	

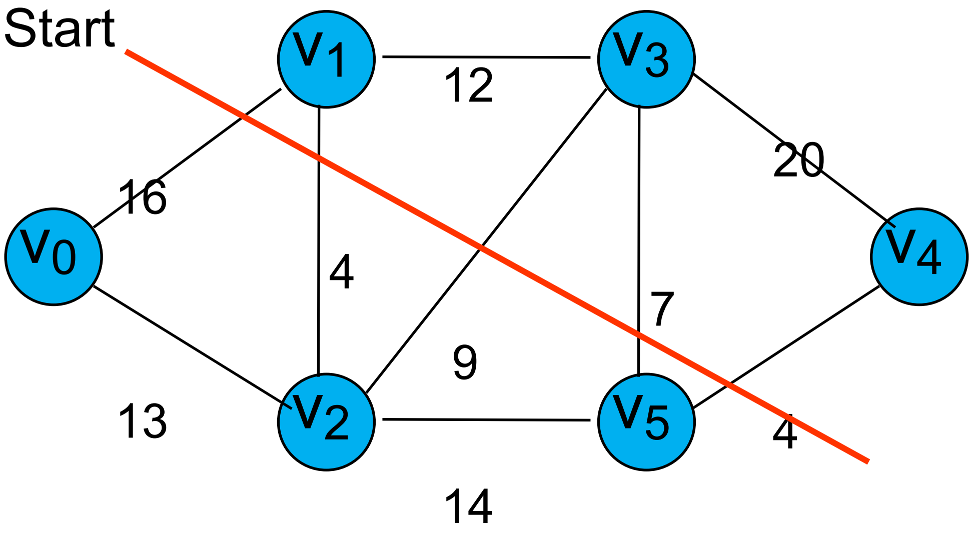
Example

- With the given initial partitioning (red) the algorithm finds the optimal solution (green) after 2 executions of the while-loop.



Example

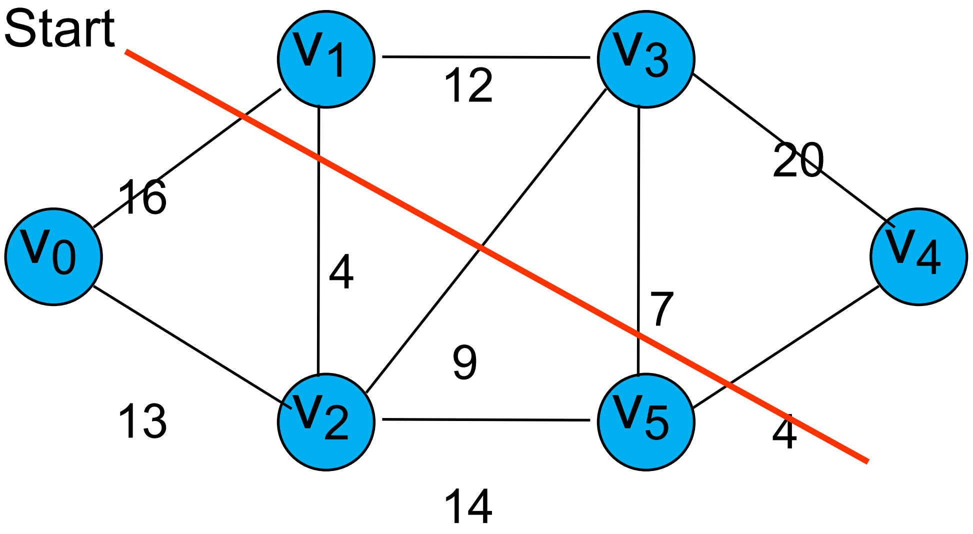
- With the given initial partitioning (red) the algorithm finds the optimal solution (green) after 2 executions of the while-loop.



	E	I	D
V0			
V1			
V2			
V3			
V4			
V5			

Example

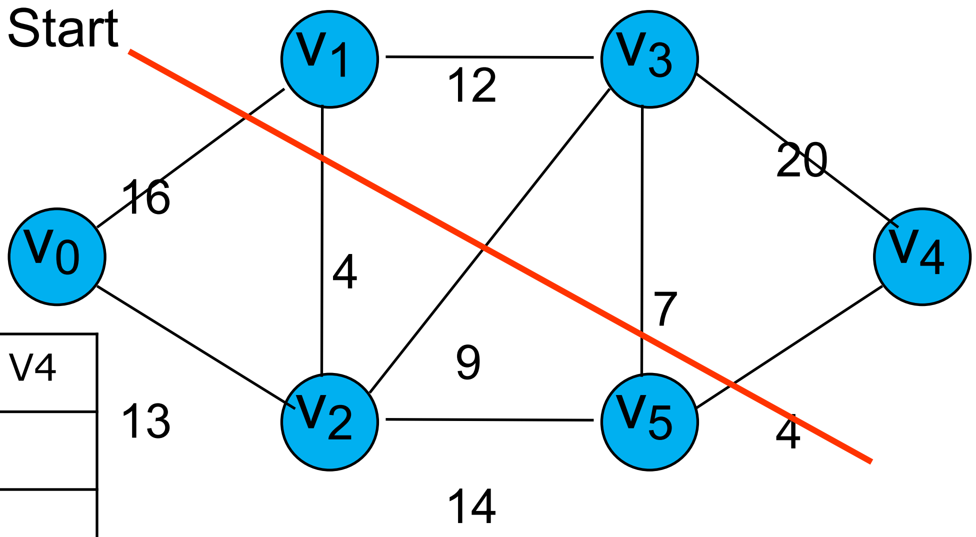
- With the given initial partitioning (red) the algorithm finds the optimal solution (green) after 2 executions of the while-loop.



	E	I	D
V0	16	13	3
V1	20	12	8
V2	13	27	-14
V3	16	32	-16
V4	4	20	-16
V5	11	14	-3

Example

- With the given initial partitioning (red) the algorithm finds the optimal solution (green) after 2 executions of the while-loop.

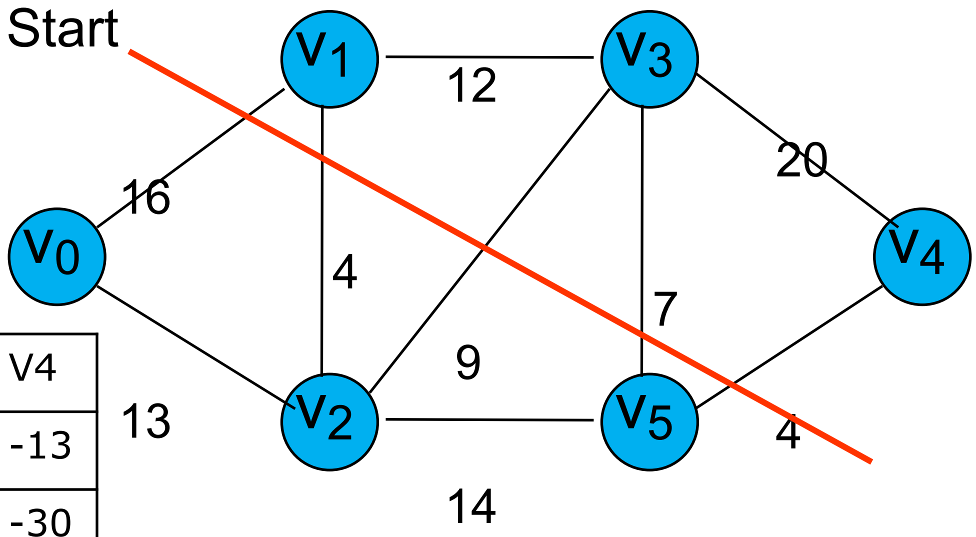


	V1	V3	V4
V0			
V2			
V5			

	E	I	D
V0	16	13	3
V1	20	12	8
V2	13	27	-14
V3	16	32	-16
V4	4	20	-16
V5	11	14	-3

Example

- With the given initial partitioning (red) the algorithm finds the optimal solution (green) after 2 executions of the while-loop.

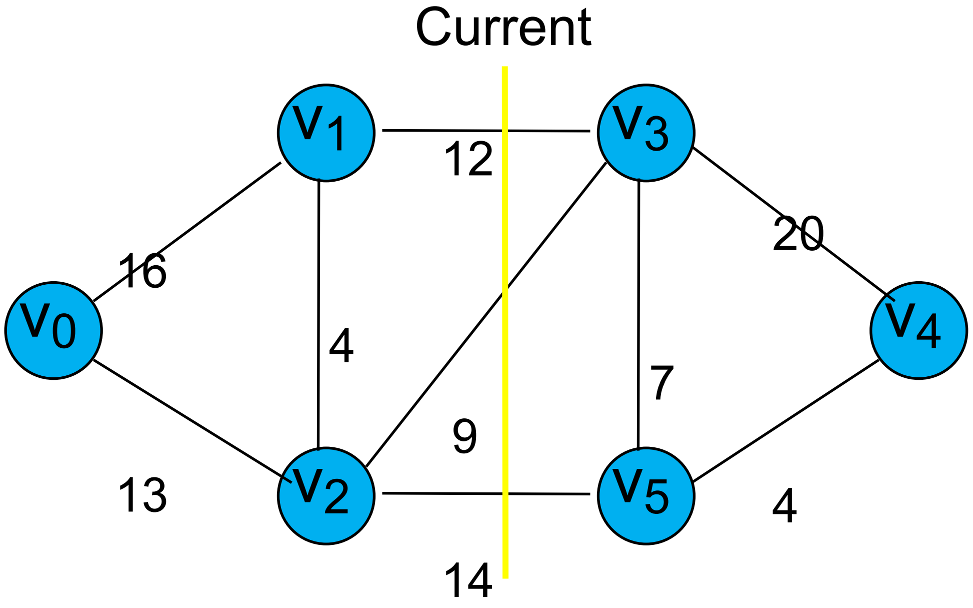


	V1	V3	V4
V0	-21	-13	-13
V2	-14	-48	-30
V5	5	-33	-27

	E	I	D
V0	16	13	3
V1	20	12	8
V2	13	27	-14
V3	16	32	-16
V4	4	20	-16
V5	11	14	-3

Example

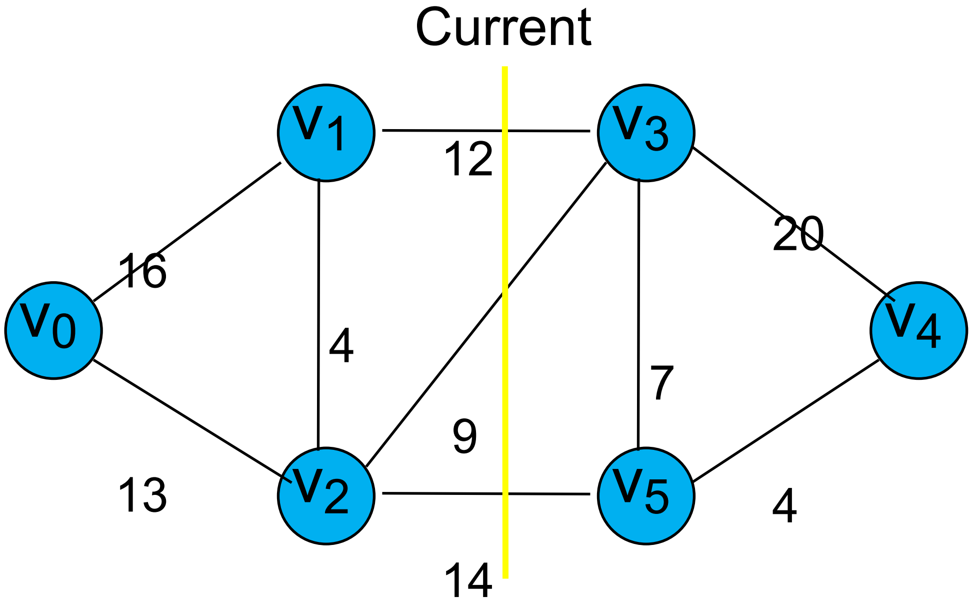
- With the given initial partitioning (red) the algorithm finds the optimal solution (green) after 2 executions of the while-loop.



	E	I	D
V0			
V1			
V2			
V3			
V4			
V5			

Example

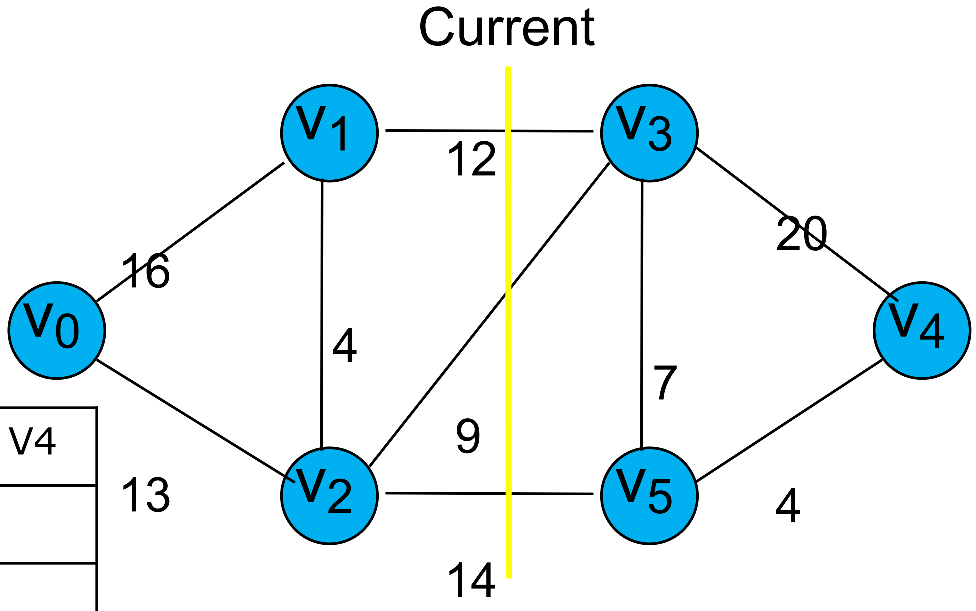
- With the given initial partitioning (red) the algorithm finds the optimal solution (green) after 2 executions of the while-loop.



	E	I	D
V0	0	29	-29
V1	12	20	-8
V2	23	17	6
V3	21	27	-6
V4	0	24	-24
V5	14	11	3

Example

- With the given initial partitioning (red) the algorithm finds the optimal solution (green) after 2 executions of the while-loop.

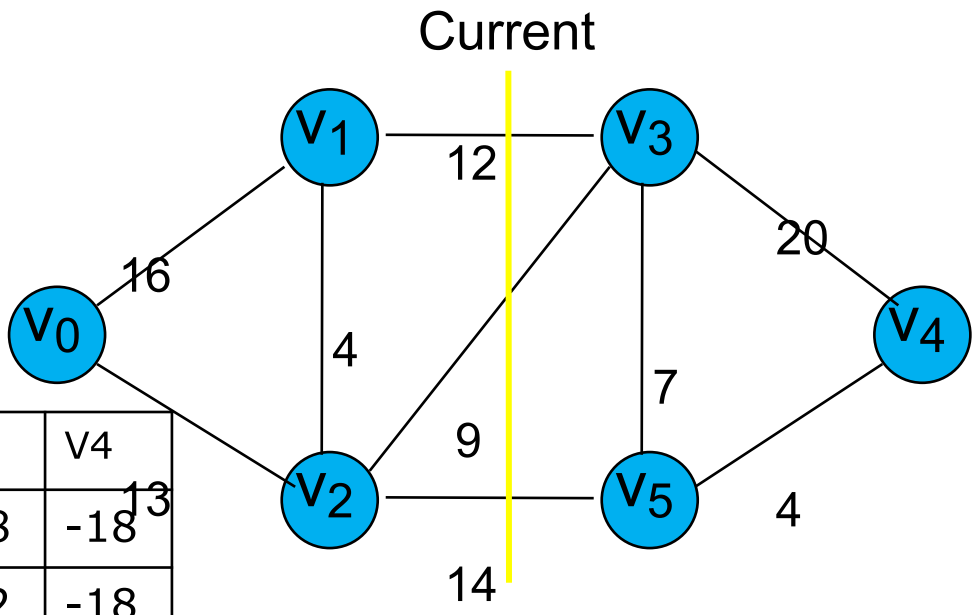


	V1	V3	V4
V0			
V2			
V5			

	E	I	D
V0	0	29	-29
V1	12	20	-8
V2	23	17	6
V3	21	27	-6
V4	0	24	-24
V5	14	11	3

Example

- With the given initial partitioning (red) the algorithm finds the optimal solution (green) after 2 executions of the while-loop.

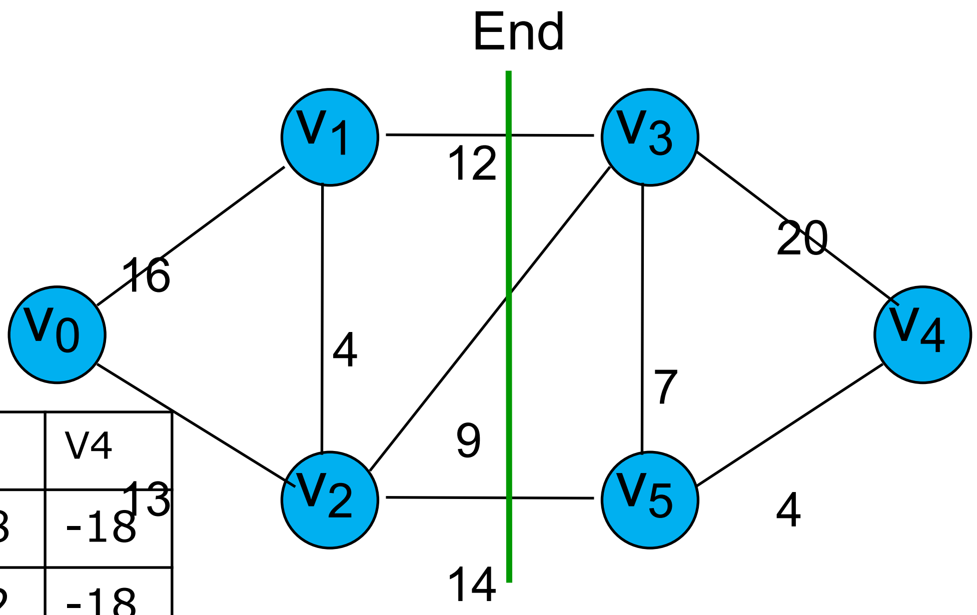


	V1	V3	V4
V0	-35	-38	-18
V2	-53	-32	-18
V5	-26	-5	-19

	E	I	D
V0	0	29	-29
V1	12	20	-8
V2	23	17	6
V3	21	27	-6
V4	0	24	-24
V5	14	11	3

Example

- With the given initial partitioning (red) the algorithm finds the optimal solution (green) after 2 executions of the while-loop.

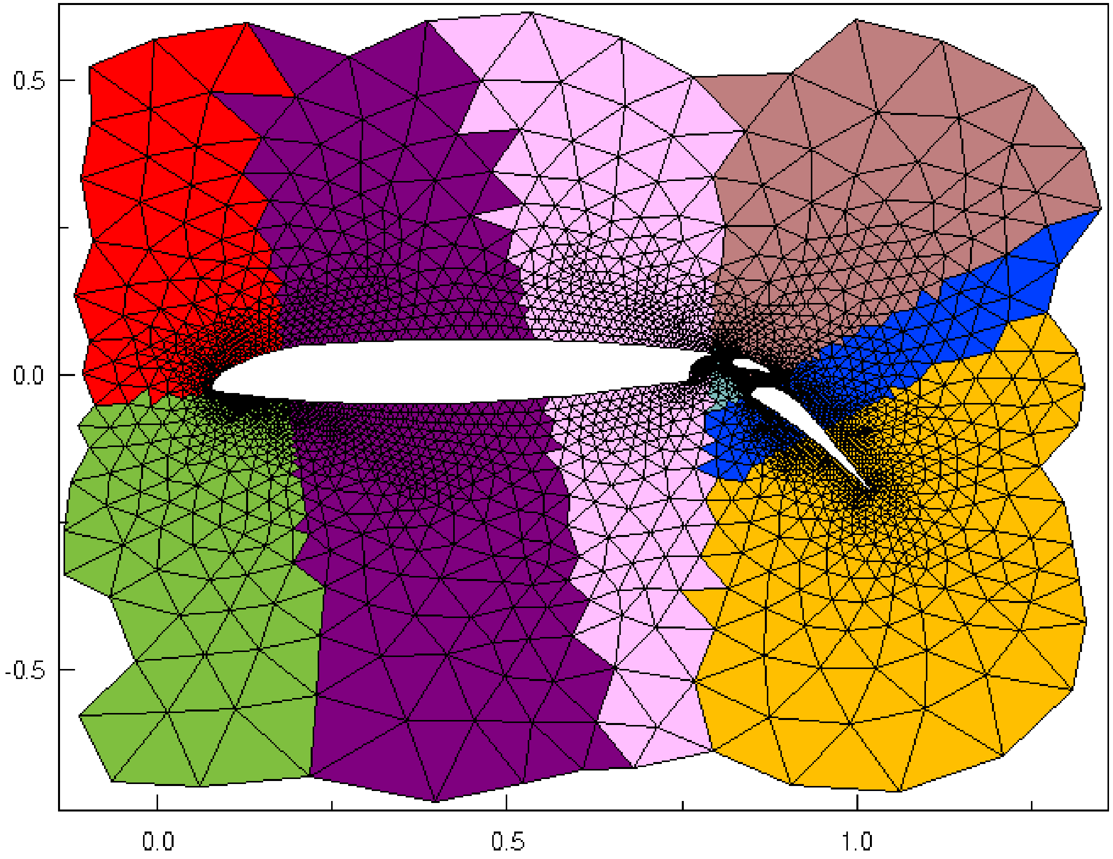


	V1	V3	V4
V0	-35	-38	-18
V2	-53	-32	-18
V5	-26	-5	-19

	E	I	D
V0	0	29	-29
V1	12	20	-8
V2	23	17	6
V3	21	27	-6
V4	0	24	-24
V5	14	11	3

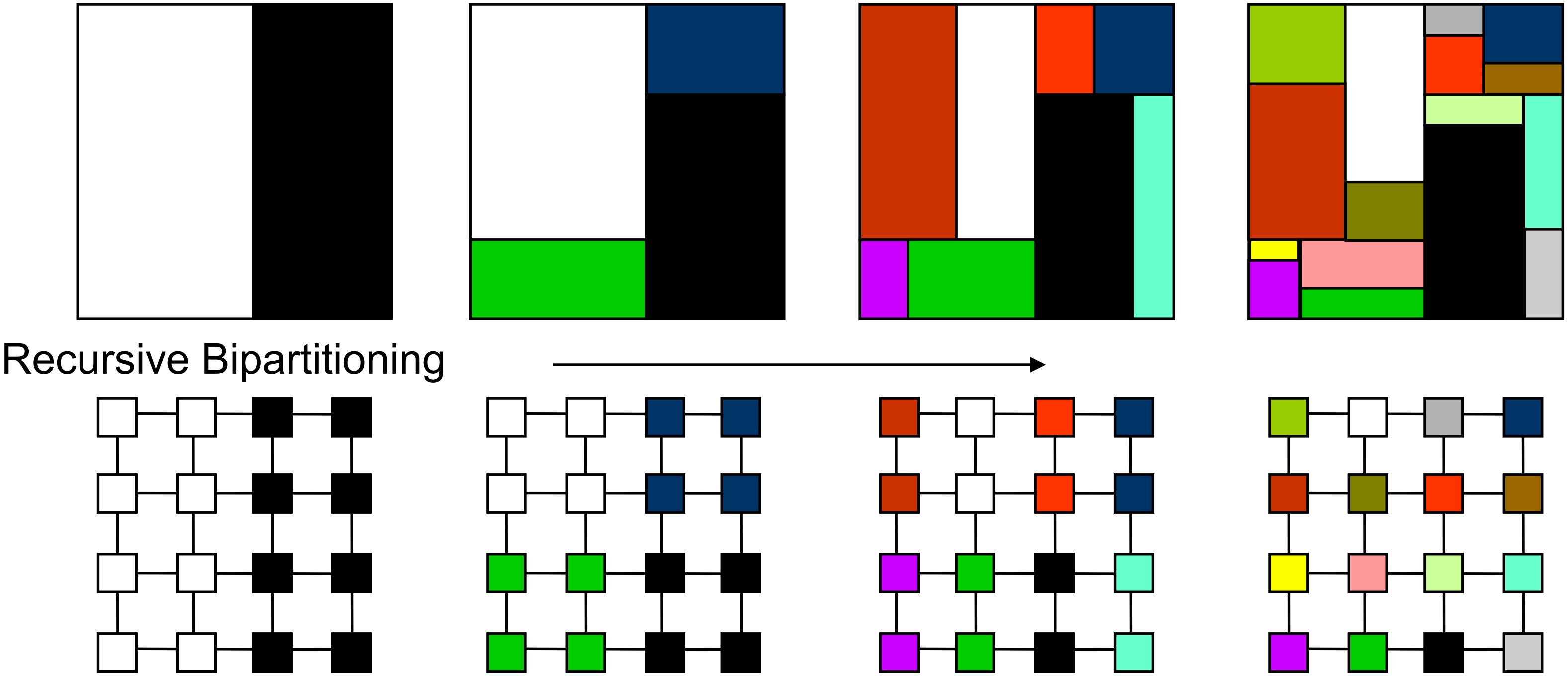
Partitioning in Practice

- In practice, the graphs to be embedded are often Finite-Element-Graphs or Finite-Volume-Graphs that result from a triangulation with non-uniform density.
- Often, there is no explicit embedding (mapping), but the partitions are assigned to the processors in a random way.



Recursive Bisectioning

- The following example shows that even if the geometric structure of the graph matches the topology of the processor network, longer communication paths will develop.
- The white and the black partition are adjacent, but are mapped to rather distant nodes.



8.4 Selforganizing Maps

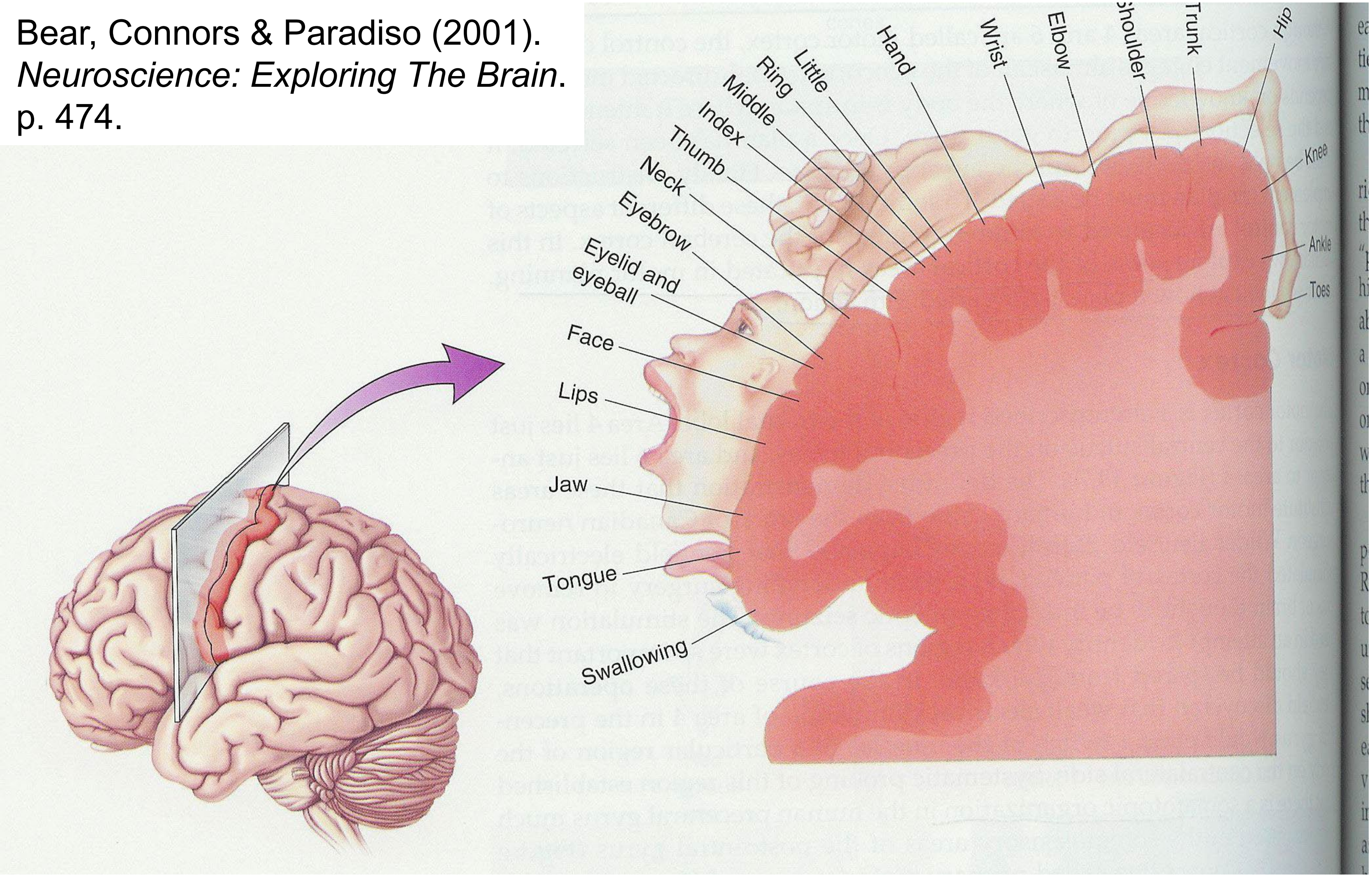
- The communication graph (TIG) of a program has to be mapped to the processor graph such that neighborhood relations are preserved.
- What we need is a **topology preserving mapping.**
- A bijective mapping π between two topological spaces A and B with metrics d_A and d_B is called **topology preserving** or **Homeomorphismus**, iff:

$$d_A(x,y) = d_B(\pi(x),\pi(y)) \quad \forall x,y \in A$$

- A topology preserving mapping can be obtained approximately by **selforganizing maps**, an application of **Kohonen networks.**

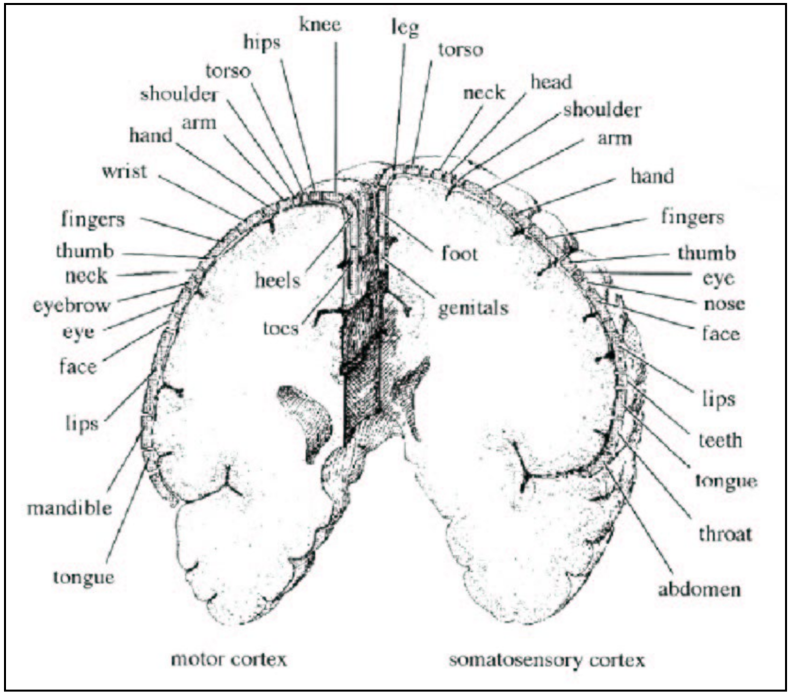
Self organization in the human brain

Bear, Connors & Paradiso (2001). *Neuroscience: Exploring The Brain*. p. 474.

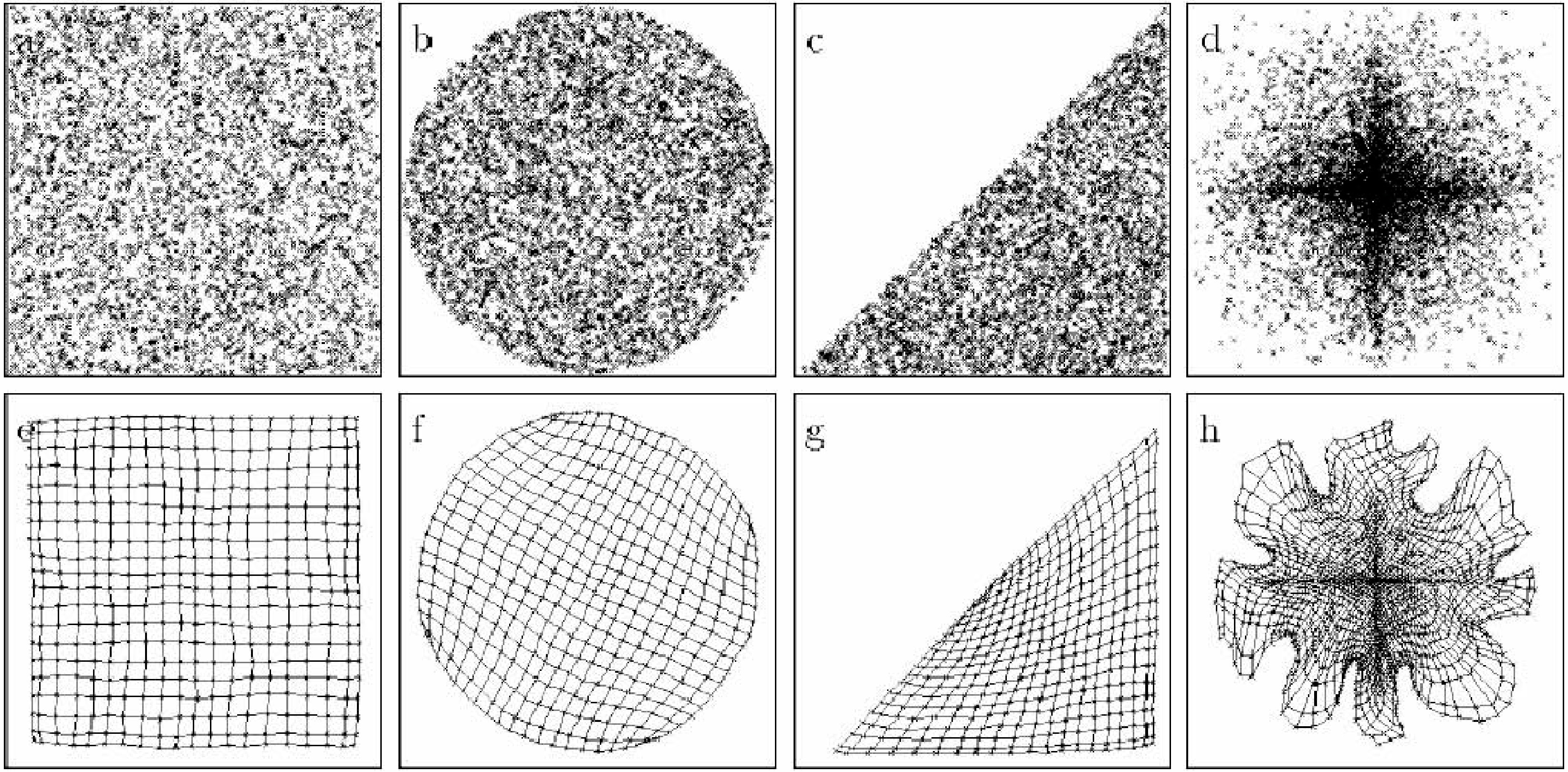


Kohonen Networks

- Self-organizing Maps (SOM)
- Developed by T. Kohonen (Helsinki)
- Mathematical model to explain the selforganization of brain cells
- Adjacent brain cells are responsible for stimulus processing of adjacent areas of the skin.

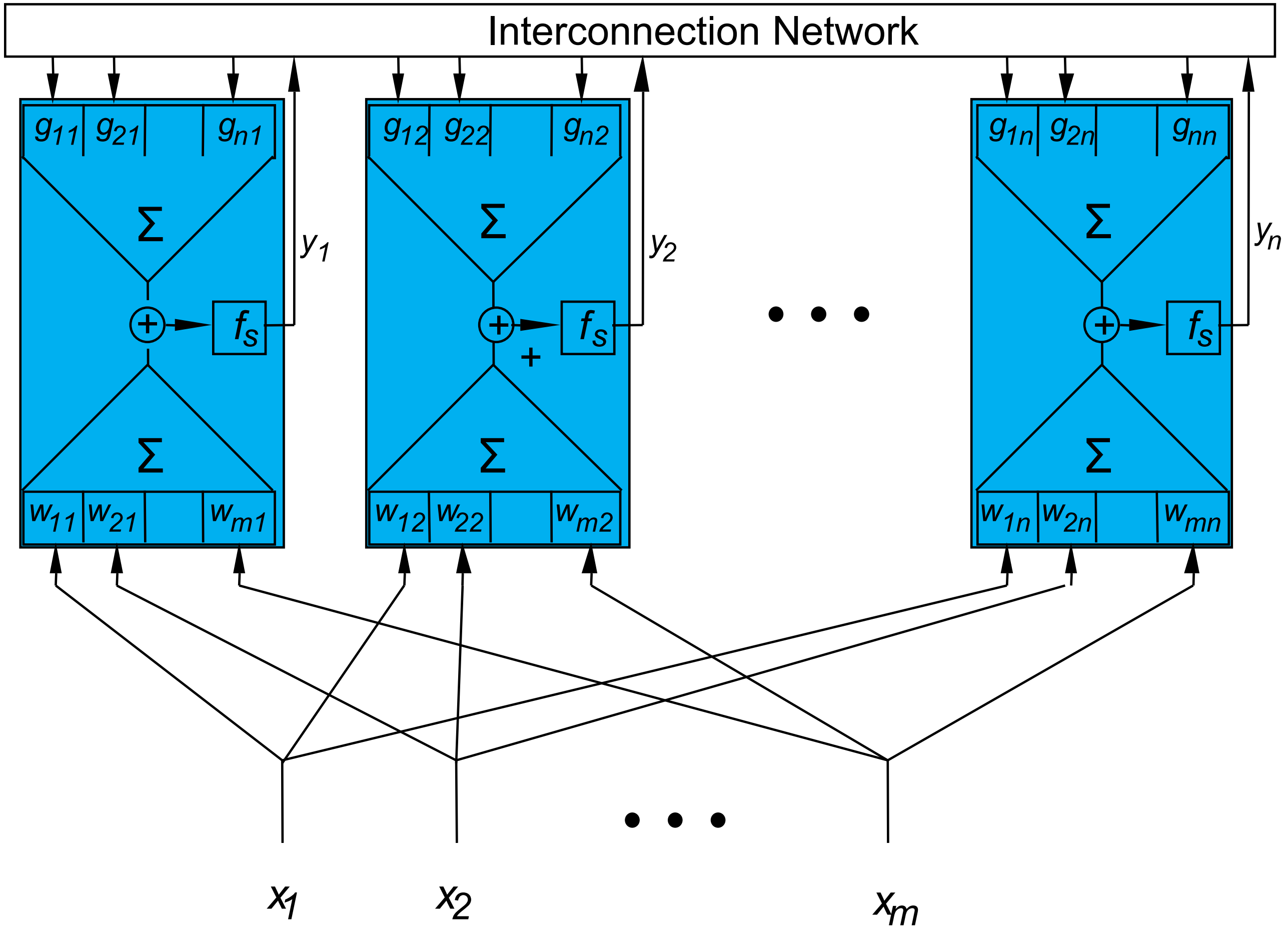


Examples for Self-Organizing Maps



The grid mimics the initial distribution

Structure of a Kohonen Network


Interconnection Network
g_{11} g_{21} g_{n1}
g_{12} g_{22} g_{n2}
g_{1n} g_{2n} g_{nn}
Σ
f_s
+
y_1
y_2
y_n
w_{11} w_{21} w_{m1}
w_{12} w_{22} w_{m2}
w_{1n} w_{2n} w_{mn}
x_1
x_2
x_m

Structure of a Kohonen Network

- A **Kohonen Network** is a layer of *n* neurons connected by some network.
- Between neurons a distance function *d(i,k)* is defined.
- Each neuron is attached to each of the *m* inputs that are associated with a specific weight w_{ij} .
- If we have a signal $\boldsymbol{x}$ at the input, each neuron *i* calculates the weighted sum

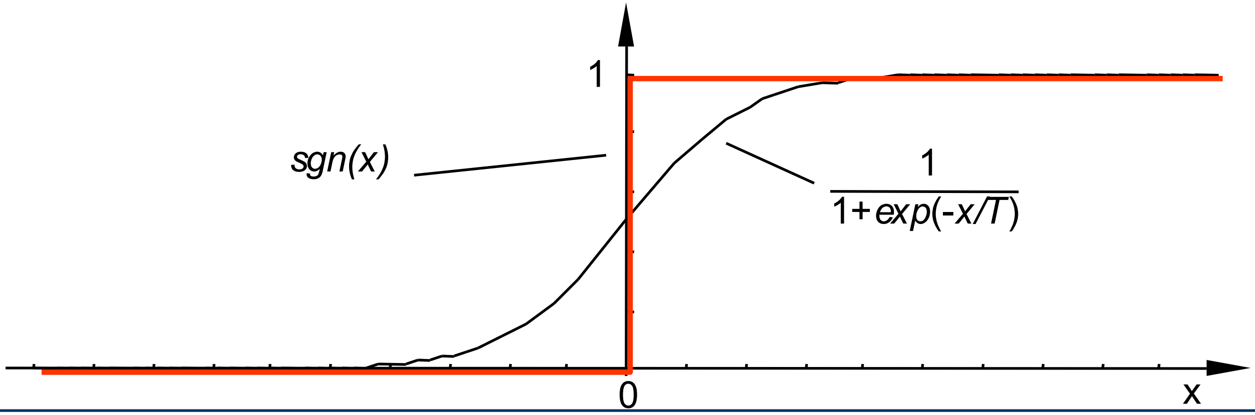
$$\sum_{j=1}^{m} w_{ij} x_j$$

Structure of a Kohonen Network

- The output signal of a neuron is given by:

$$y_i = f_s\left(\sum_{j=1}^{m} w_{ij} x_j + \sum_{k=1}^{n} g_{ki} y_k\right)$$

- The weights g_{ki} are such that they are stimulating for small distances and inhibitory for larger distances.
- f_s is a **sigmoid switching function**, that approaches 1 for x→∞ and 0 for x→-∞.



Kohonen‘s Approximation

- Kohonen describes the behavior of the network **approximately**:
- The neuron *i**, whose weighted input signal is maximum, is called **center of excitation**.

$$\sum_{j=1}^{m} w_{ji*} x_j = \max_{i=1}^{n} \sum_{j=1}^{m} w_{ji} x_j$$

- If we normalize the weights, we can also write:

$$\left\| \vec{w}_{i*} - \vec{x} \right\| = \min_{i=1}^{n} \left\| \vec{w}_i - \vec{x} \right\|$$

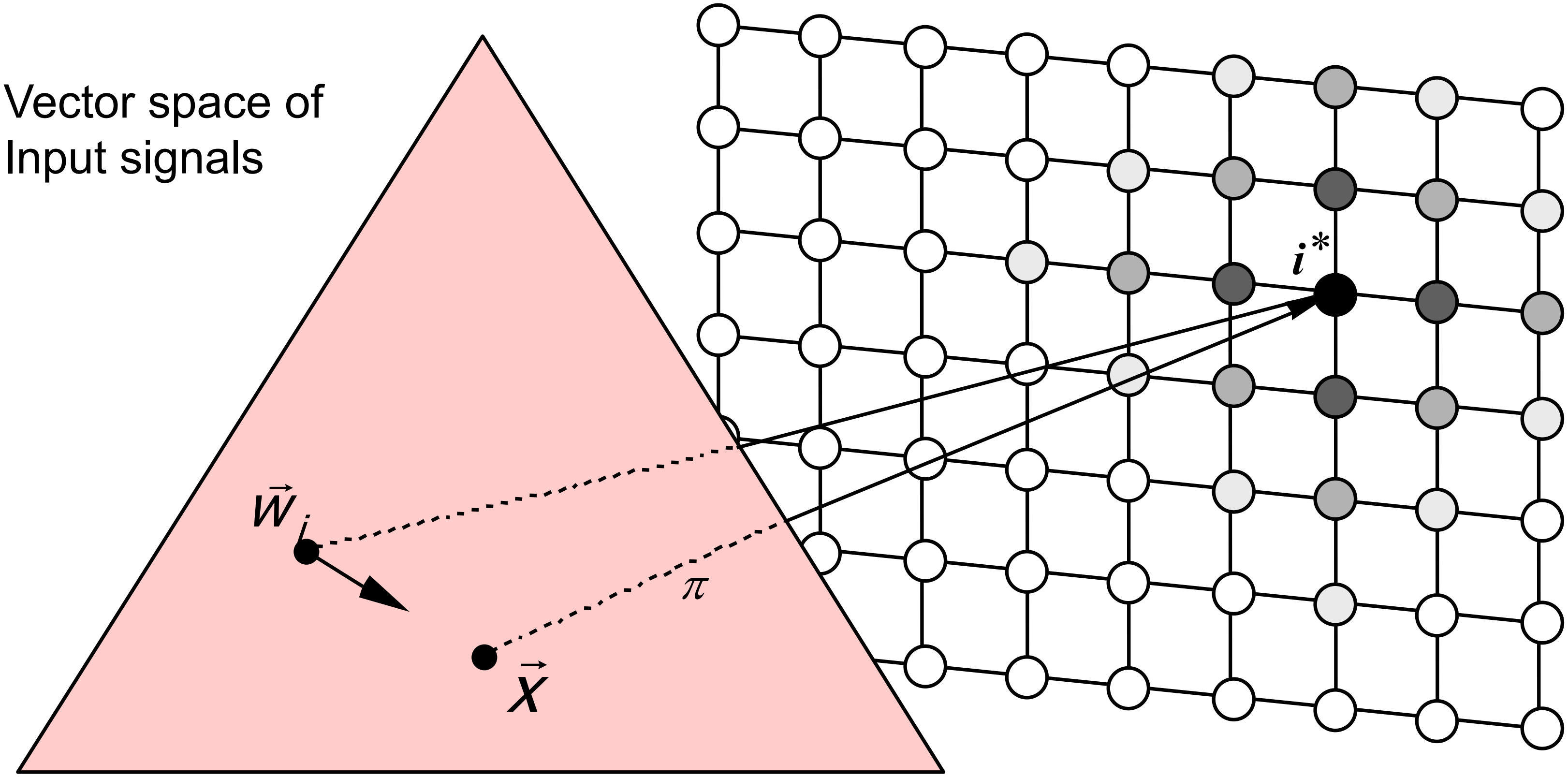
with

$$\vec{w}_i = \left(w_{1i}, w_{2i}, \ldots, w_{mi}\right)^T \text{ and } \vec{x} = \left(x_1, x_2, \ldots, x_m\right)^T$$

- Depending on the weights, a mapping is defined that assigns each input signal *x* a place (neuron) i*:

$$\pi_w : \vec{x} \mapsto i* = \pi_w\left(\vec{x}\right)$$

Excitation and Adaptation



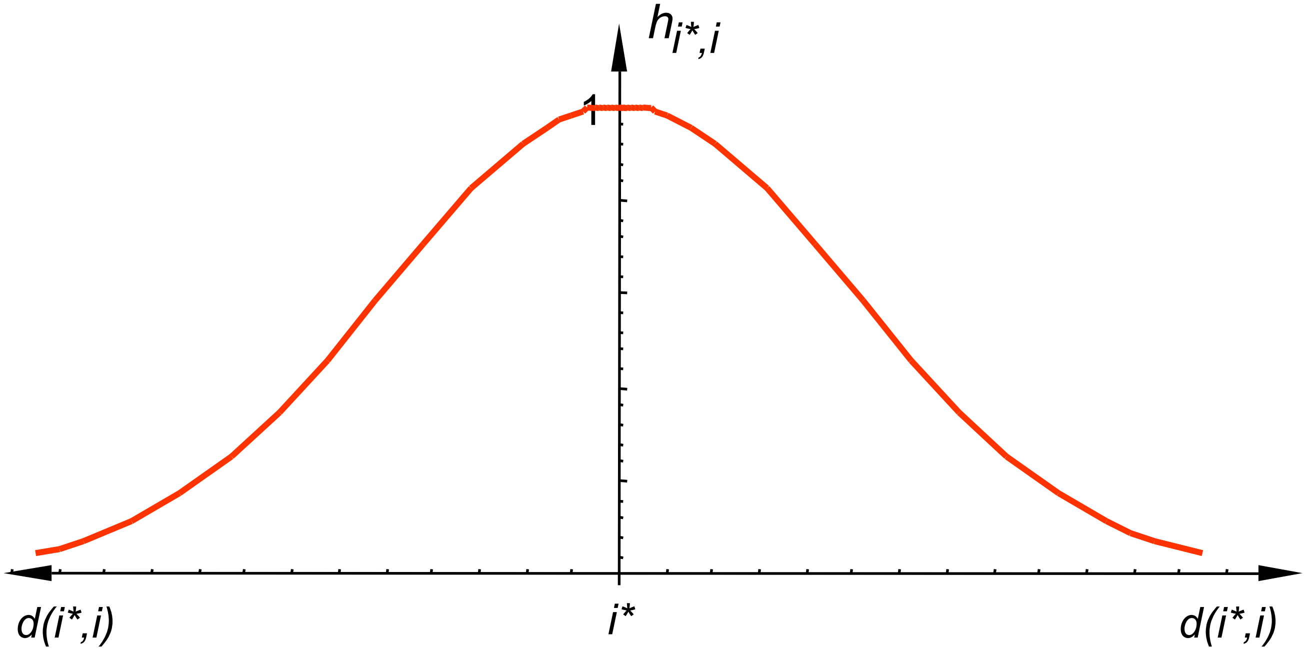
Neural network

Learning rule: $\vec{w}_i^{new} = \vec{w}_i^{old} + \varepsilon \cdot h_{i*,i} \cdot \left(\vec{x} - \vec{w}_i^{old}\right)$

Excitation in neighborhood

- In the neighborhood of the center, the excitation decreases with the distance.
- As function usually the bell-shaped Gaussian density curve is used:

$$h_{i^*,i} := exp\left(-\frac{d(i^*,i)^2}{2\sigma^2}\right)$$



Kohonen Algorithm

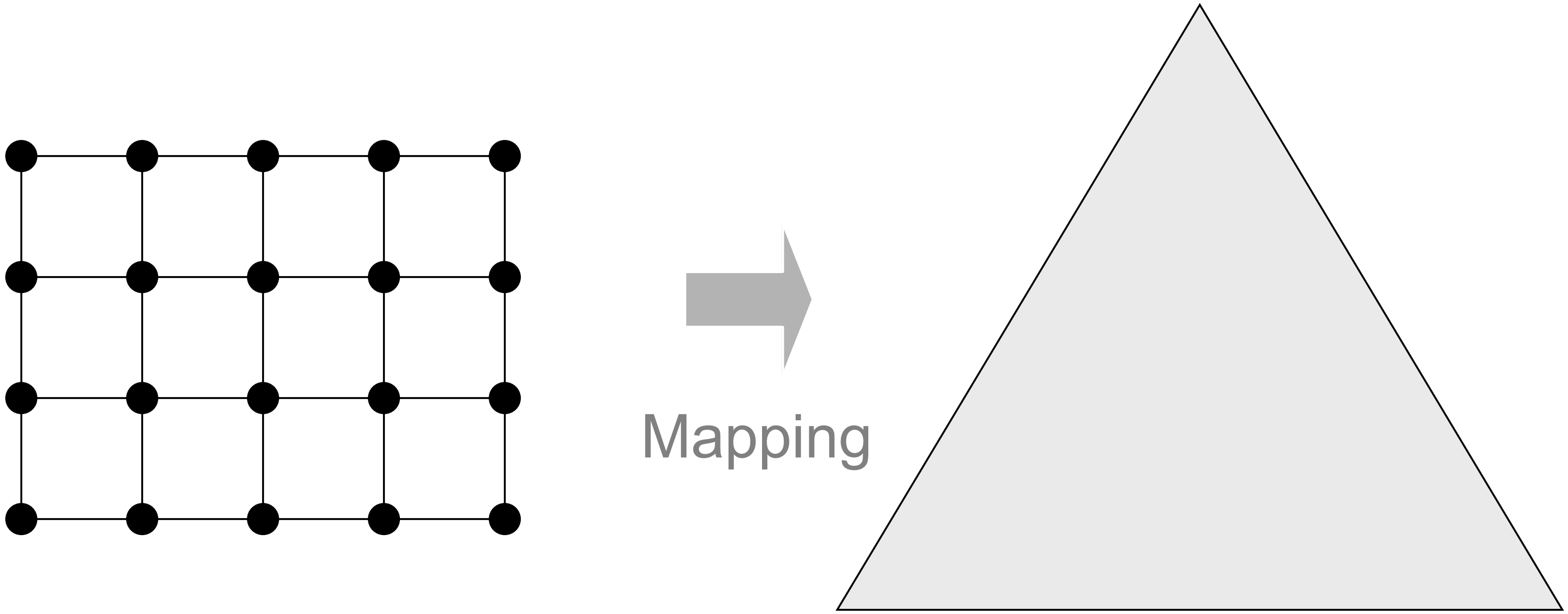
- In a loop the input signals (stimuli) x are generated following some probability distribution p(x) which stimulates the network that adapts its weights accordingly.

0	Kohonen-Algorithm	
1	initialize w_{ji}	Random selection of initial weights
2	**while** (not stopping_condition) **do**	Main loop
3	select $\vec{x} \in X$ according to $p(\vec{x})$	Stimulus selection
4	determine i* with $\lVert \vec{w}_{i*} - \vec{x} \rVert = \min_{i=1}^{n} \lVert \vec{w}_i - \vec{x} \rVert$	Center of excitation
5	$w_{ji} = w_{ji} + \varepsilon \cdot h_{i*,i} \cdot \left(x_j - w_{ji}\right) \quad \forall i,j$	Adaptation of weights
6	**end** while	
7	**end**	

Application to Mapping Problem

- The processors and their interconnection links are the Kohonen network.
- The program graph to be mapped generates the input signals.
- Both graphs are being represented in the same geometric area.
- In each step a node of the program graph is offered an input signal
- This way the processor graph unfolds in the program graph.
- (Principally also the opposite direction is possible.)

Kohonen Networks



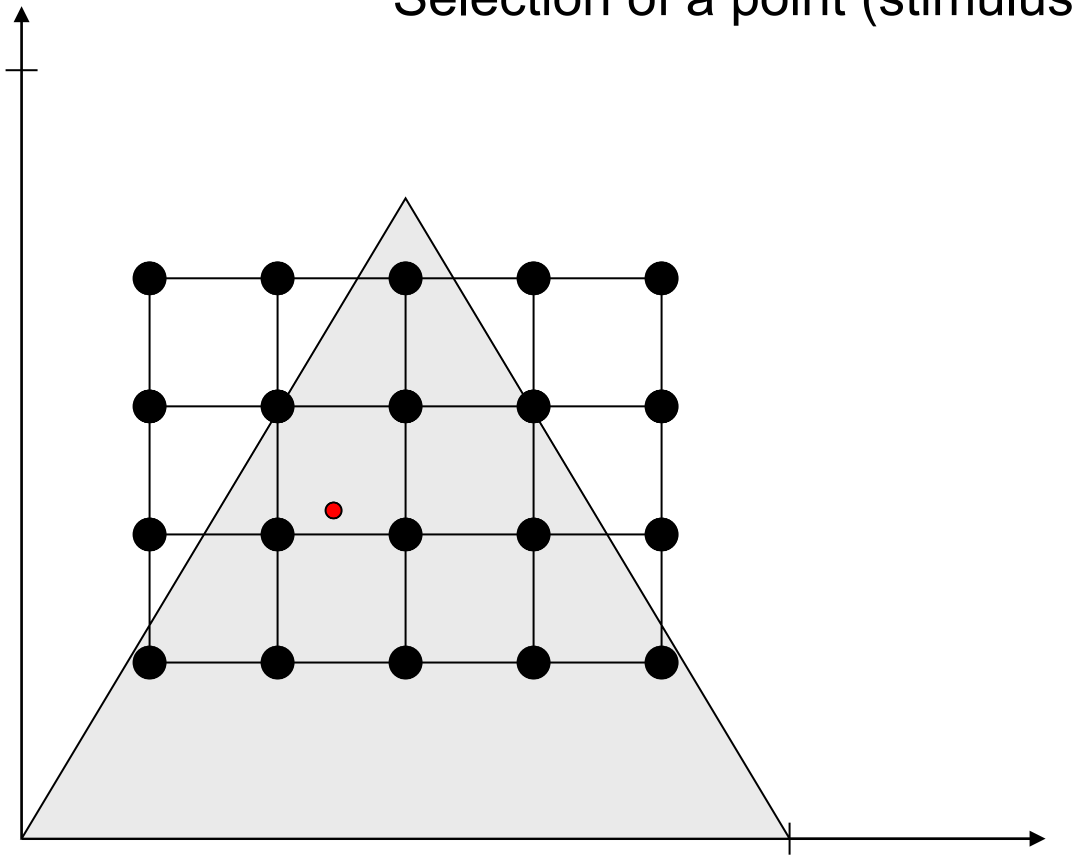
Mapping

Processor graph

Program graph

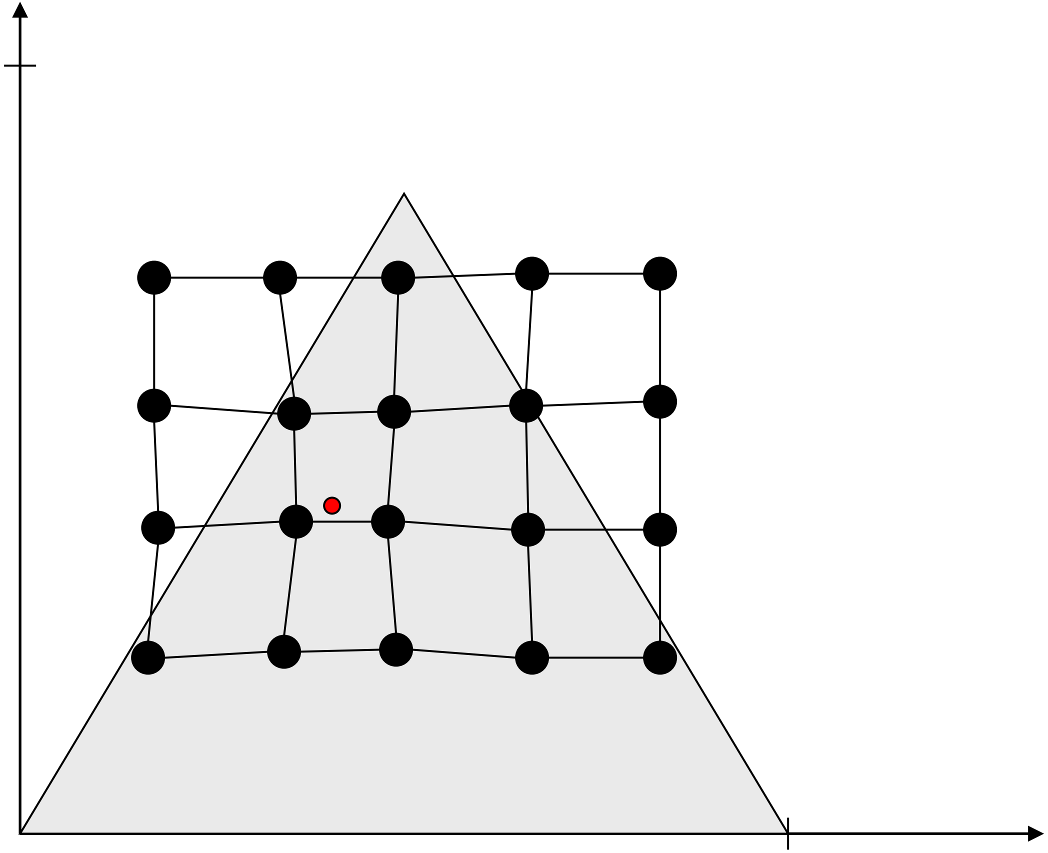
Kohonen Networks

Selection of a point (stimulus)....



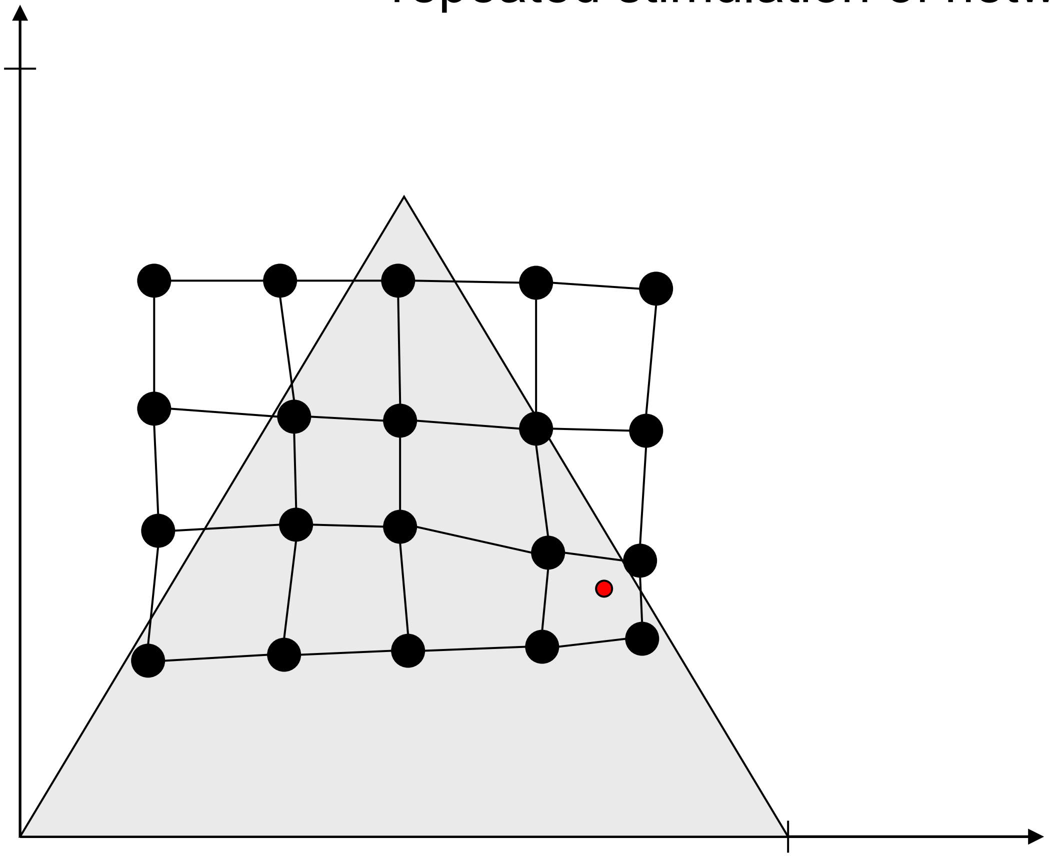
Kohonen Networks

reaction of network....



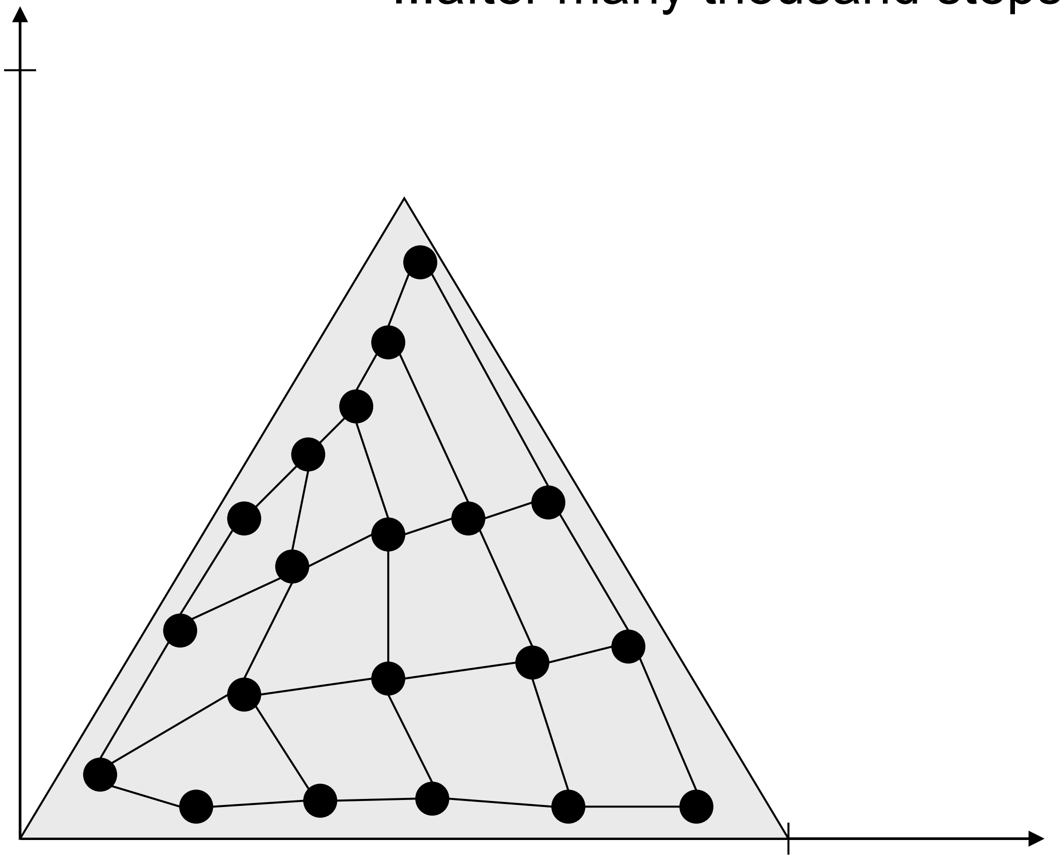
Kohonen Networks

repeated stimulation of network....

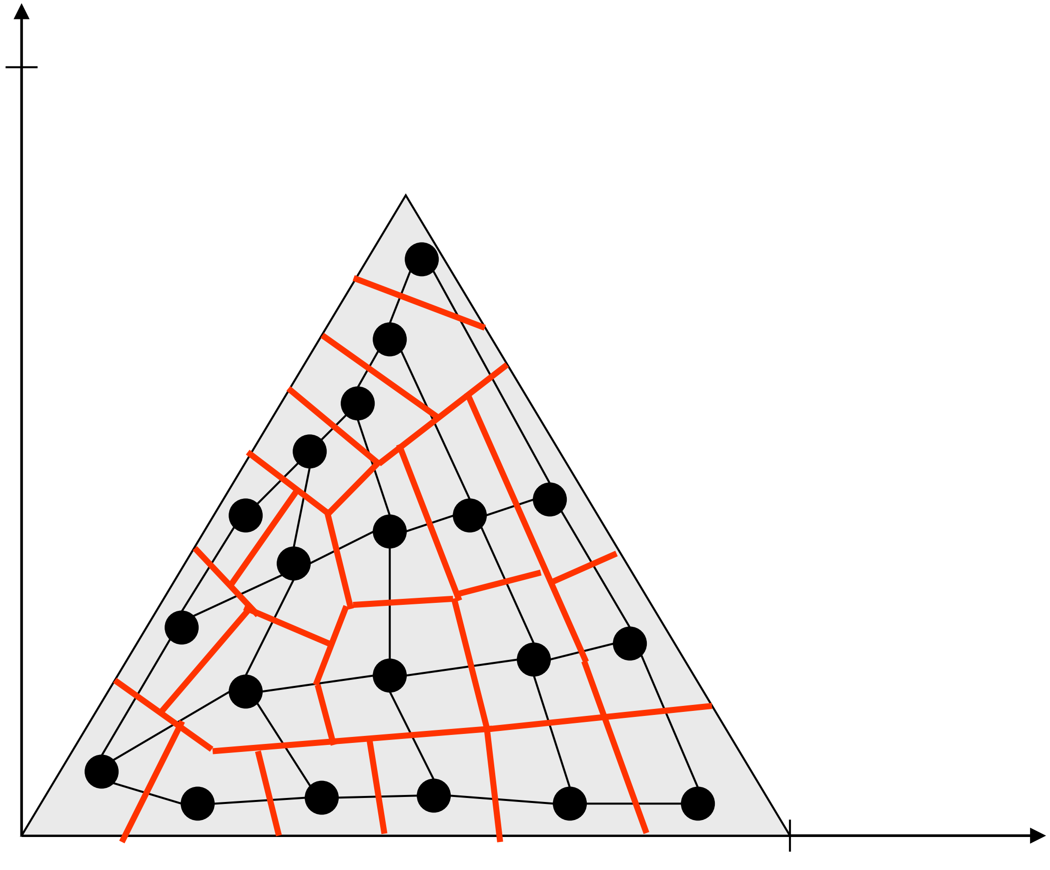


Kohonen Networks

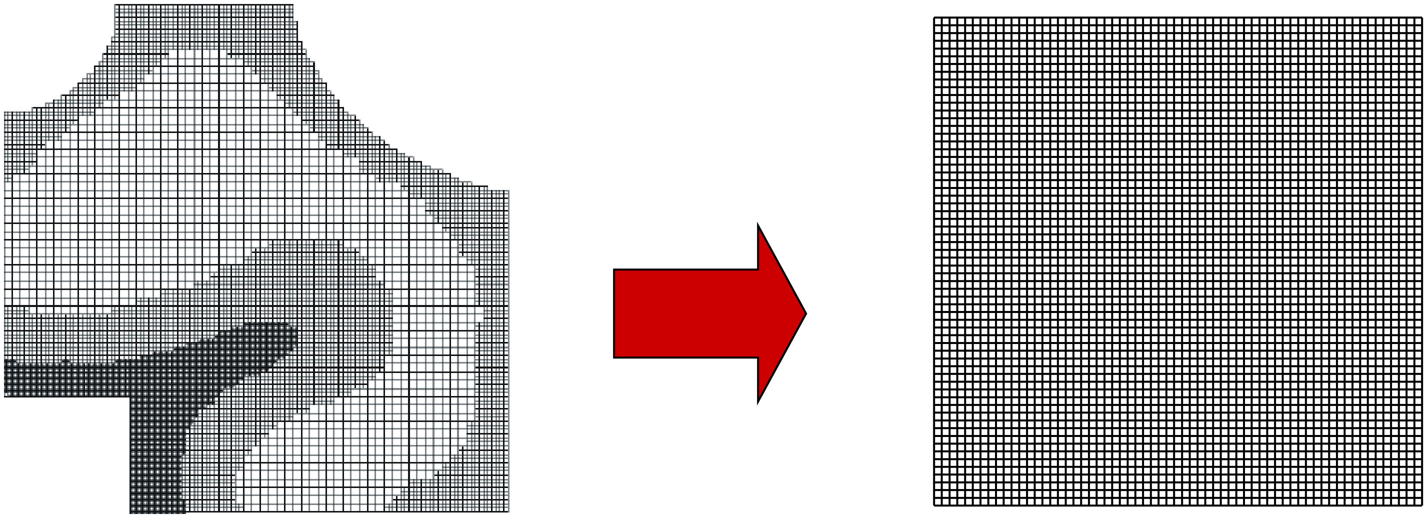
...after many thousand steps



Kohonen Networks: Voronoi decomposition



Real Example

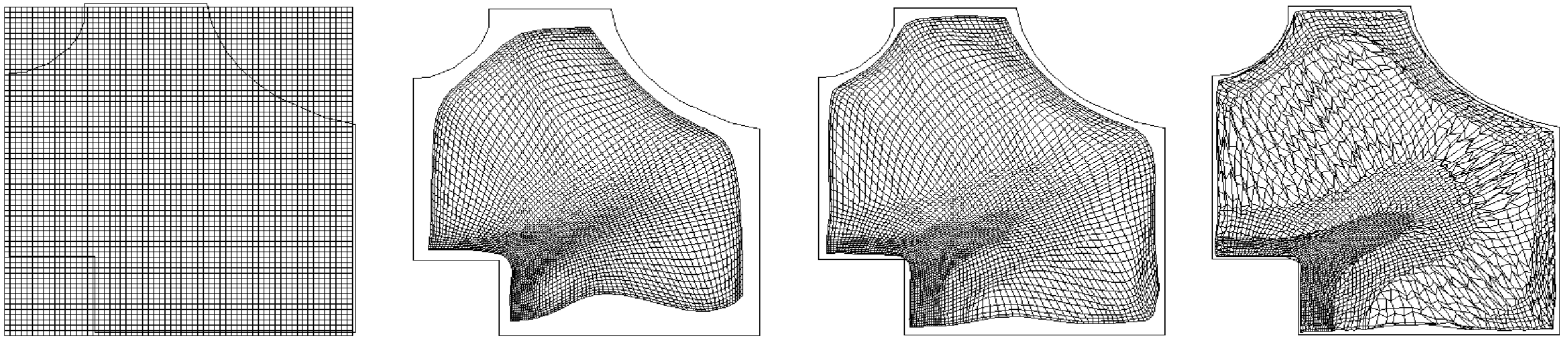


FEM graph with 44663 vertices

parallel computer with 64x64 nodes

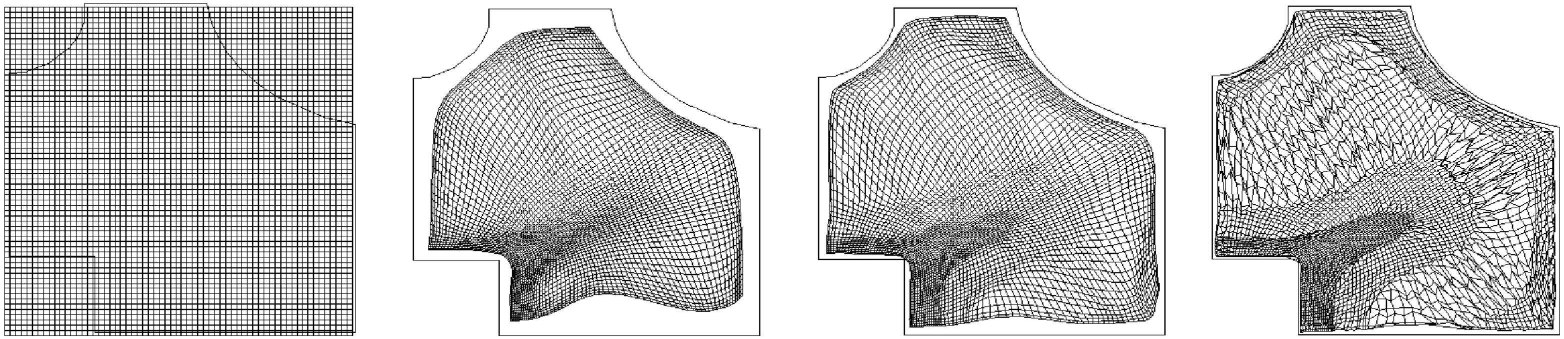
Course of the mapping

Topological process →

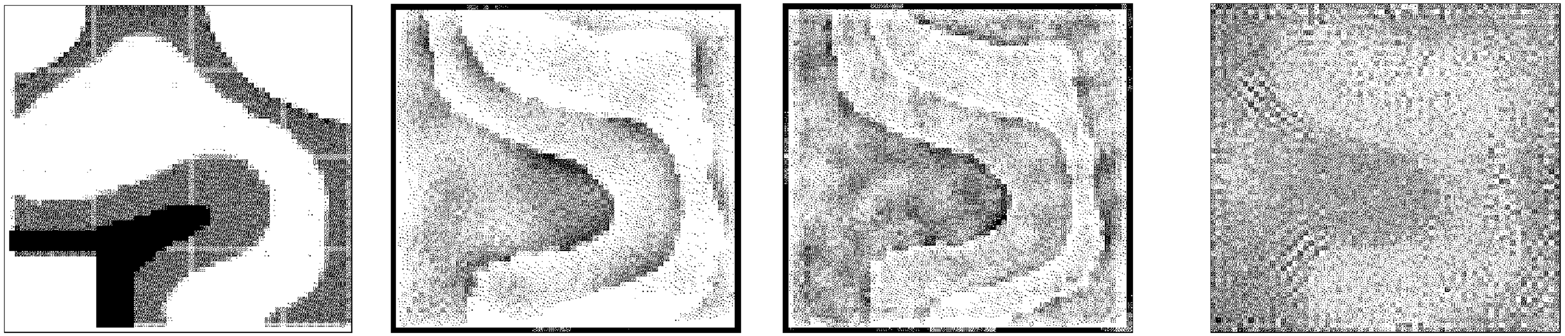


Course of the mapping

Topological process



Development of load distribution



References

- Arnold L. Rosenberg: *GRAPH EMBEDDINGS 1988: Recent Breakthroughs, New Directions*. 3rd Aegean Workshop on Computing 1988: 160-169
- B. Monien, H. Sudborough: Embedding one Interconnection Network in Another. Computing Suppl. 7, 1990, pp. 257-282, 1990.
- Mee Yee Chan. *Embedding of grids into optimal hypercubes*. SIAM Journal on Computing, 20(5):834-864, October 1991.
- Kohonen,T.: *Self-Organization and Associative Memory.* Springer-Verlag Berlin, 3rd edition (1989)
- H. Ritter, T. Martinetz, K. Schulten: *Neural Computation and Self-Organizing Maps*, Addison-Wesley, 1992
- Dormanns,M.; Heiss,H.-U.: *Partitioning and Mapping of Large FEM-Graphs by Self-Organization.* Proc. 3rd Euromicro Workshop on Parallel and Distributed Processing, San Remo, 25.-27.Jan. 1995, S. 227-235.
- Heiss, H.-U.; Dormanns, M.: *Partitioning and Mapping of Parallel Programs by Self-Organization.* Concurrency - Practice and Experience 8,9 (Nov. 1996) S. 685-706.