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).
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 $\theta$ 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 $e'$:

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

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

$$vcong(v') := |\{e \in E_G | v' \in Vl(\pi_E(e))\}|$$

- 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:

\[
\text{expansion}(\theta) := \frac{|V_H|}{|V_G|}
\]

- 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 \(\text{card}(\theta)\) 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 $\theta$ with $\text{dil}(\theta)=2$, $\text{econg}(\theta)=2$, $\text{vcong}(\theta)=2$, $\text{expansion}(\theta)=4/3$ and $\text{card}(\theta)=4$
Graph isomorphism

- An embedding $\theta = (\pi_V, \pi_E)$ is called an **isomorphism**, if $\pi_V$ as well as $\pi_E$ 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 Injctive 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 \textit{dilation} and/or minimal \textit{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):

![Binary tree diagrams]
2D-mesh as guest graph

- With regard to the rectangular mesh $b \times h$ to be embedded, the quadratic mesh of side length $s$ is called

  **ideal**, if \[ s = \left\lfloor \sqrt{b \cdot h} \right\rfloor \]

  **almost ideal**, if \[ s = \left\lfloor \sqrt{b \cdot h} \right\rfloor + 1 \]

- 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 $\rho := h/b$ 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 $\text{dil}=3$.
- With a small modification we achieve that more edges are dilated but the dilation does not exceed 2.

![Step embedding](image1)

![Modified step embedding](image2)
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 $\text{dil} = 2$ at an expansion of

$$\text{expansion}(\theta) = \left\lceil \sqrt{\rho} \right\rceil^2 / \rho$$
Further examples

**FIG. 4.** Embedding of a $5 \times 25$ mesh into a $8 \times 16$ mesh by modified

**FIG. 5.** Embedding of a $3 \times 20$ mesh into a $4 \times 16$ 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 $V$:
  $$ P = \{V_1, V_2, \ldots, V_k\}, \quad V_i \subseteq V \text{ with } \bigcup_{i=1}^{k} V_i = V $$
  
  - „exhaustive“
  
  - $i \neq k \Rightarrow V_i \cap V_k = \emptyset$  
  - „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**:
  $$ \text{cut}(P) = E - \bigcup_{i=1}^{k} E_i = \{(u, v) \in E \mid u \in V_i, v \in V_j \text{ with } i \neq j\} $$
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 \( G \):
  \[
  V' = \{V'_1, V'_2, ..., V'_k\}
  \]
  \[
  (V'_p, V'_q) \in E' \iff \exists v_p \in V_p, v_q \in V_q : (v_p, v_q) \in E, \text{ where } V_p, V_q \in P
  \]

- 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

Partitioning of problem graph

Contraction graph

Embedding

Barry Linnert, linnert@inf.fu-berlin.de, Cluster Computing SoSe 2020
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 $n = \text{number of nodes}$ and $m = \text{number of edges}$).
  - 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 $2m$ vertices and edge weights.
- Goal: two equally sized partitions $X$ and $Y$ with minimal cut costs:

$$\Gamma(X,Y) := \min_{u \in X, v \in Y} \sum_{\gamma(u,v)}$$
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
  \[
g = \Delta \Gamma = E(u) + E(v) - \gamma(u,v) - \left[ I(u) + I(v) + \gamma(u,v) \right) = D(u) + D(v) - 2 \ \gamma(u,v)
  \]
Example

Do we want to exchange $u$ and $v$?
Example

Cut costs $\Gamma = \gamma(u,v)$

\[
\begin{align*}
\gamma(u,v) &= \quad = \\
E(u) &= \quad = \\
I(u) &= \quad = \\
D(u) &= \quad = \\
E(v) &= \quad = \\
I(v) &= \quad = \\
D(v) &= \quad = \\
\end{align*}
\]

Cut costs $\Gamma =
\begin{align*}
g &= D(u) + D(v) - 2 \gamma(u,v) \\
g &= \\
\end{align*}$
Example

Cut costs $\Gamma = 3+5+7+9 = 24$

| $\gamma(u,v)$ | 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 $\Gamma = 4+5+6+8+9 = 32$

$g = D(u)+D(v)-2 \gamma(u,v) = 2+0-10 = -8$
Kernighan-Lin-Algorithm 1

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\text{mincut}(X_{in}, Y_{in}, X_{out}, Y_{out})$</td>
</tr>
<tr>
<td>2</td>
<td>$X \leftarrow X_{in}; \ Y \leftarrow Y_{in}$ \hspace{1cm} Init. balanced bipartitioning</td>
</tr>
<tr>
<td>3</td>
<td>$G \leftarrow \infty$ \hspace{1cm} Initialize gain</td>
</tr>
<tr>
<td>4</td>
<td>$\textbf{while } G &gt; 0 \textbf{ do}$ \hspace{1cm} Search as long as we achieve</td>
</tr>
<tr>
<td>5</td>
<td>$X' \leftarrow X$ \hspace{1cm} an improvement</td>
</tr>
<tr>
<td>6</td>
<td>$Y' \leftarrow Y$ \hspace{1cm} Auxiliary variable to store</td>
</tr>
<tr>
<td>7</td>
<td>$\textbf{for } i \leftarrow 1 \textbf{ to } m \textbf{ do}$ \hspace{1cm} temporary node movements</td>
</tr>
<tr>
<td>8</td>
<td>$\textbf{for all } v \in V, \ v \text{ unmarked } \textbf{ do}$</td>
</tr>
<tr>
<td>9</td>
<td>calculate $D[v]$ resp. $X', Y'$ \hspace{1cm} Calculation of difference costs</td>
</tr>
<tr>
<td>10</td>
<td>$\textbf{end for}$</td>
</tr>
<tr>
<td>11</td>
<td>$\textbf{for all } (u, v) \text{ unmarked, } u \in X', v \in Y' \textbf{ do}$ \hspace{1cm} Calculation of gain when</td>
</tr>
<tr>
<td>12</td>
<td>$g[u, v] \leftarrow D[u] + D[v] - 2 \gamma[u, v]$ \hspace{1cm} exchanging $u$ and $v$</td>
</tr>
<tr>
<td>13</td>
<td>$\textbf{end for}$</td>
</tr>
<tr>
<td>14</td>
<td>$g[i] \leftarrow \text{max}{g[u, v]}$ \hspace{1cm} Storing and marking of that pair</td>
</tr>
<tr>
<td>15</td>
<td>$(u^<em>[i], v^</em>[i]) \leftarrow (u^<em>, v^</em>) \text{ with max}{g[u, v]}$ \hspace{1cm} the exchange of which maximizes</td>
</tr>
<tr>
<td>16</td>
<td>mark $u^<em>, v^</em>$ \hspace{1cm} the gain.</td>
</tr>
<tr>
<td>17</td>
<td>$X' \leftarrow X' - {u^<em>[i]} \cup {v^</em>[i]}$ \hspace{1cm} Temporary exchange of marked pair</td>
</tr>
<tr>
<td>18</td>
<td>$Y' \leftarrow Y' - {v^<em>[i]} \cup {u^</em>[i]}$</td>
</tr>
</tbody>
</table>

Barry Linnert, linnert@inf.tu-berlin.de, Cluster Computing SoSe 2020
### Kernighan-Lin-Algorithm 2

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>( G[k] \leftarrow \sum_{j=1}^{k} g[j] ) (( k = 0, \ldots, n )) Series of ( m ) exchange pairs is found. ((X' = Y, Y' = X)) ( G[k] ) indicates the gain accumulated over the first ( k ) exchange steps</td>
</tr>
<tr>
<td>21</td>
<td>( k^* \leftarrow \min k ) with ( \max {G[k]} ) With ( k^* ) a subsequence optimal with regard to the initial partitioning</td>
</tr>
<tr>
<td>22</td>
<td>( G \leftarrow G[k^*] ) has been found</td>
</tr>
<tr>
<td>23</td>
<td>if ( G &gt; 0 ) Exchange steps are performed only if</td>
</tr>
<tr>
<td>24</td>
<td>then</td>
</tr>
<tr>
<td>25</td>
<td>( X \leftarrow X - {u^<em>[1], \ldots, u^</em>[k^<em>]} \cup {v^</em>[1], \ldots, v^<em>[k^</em>]} ) Factual exchange</td>
</tr>
<tr>
<td>26</td>
<td>( Y \leftarrow Y - {v^<em>[1], \ldots, v^</em>[k^<em>]} \cup {u^</em>[1], \ldots, u^<em>[k^</em>]} )</td>
</tr>
<tr>
<td>27</td>
<td>end then</td>
</tr>
<tr>
<td>28</td>
<td>end while Further exchange steps do not lead to any improvement (( G=0 )),</td>
</tr>
<tr>
<td>29</td>
<td>( X_{\text{out}} \leftarrow X ) Algorithm stops.</td>
</tr>
<tr>
<td>30</td>
<td>( Y_{\text{out}} \leftarrow Y )</td>
</tr>
<tr>
<td>31</td>
<td>end ( \text{mincut} )</td>
</tr>
</tbody>
</table>
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.
Example

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

![Graph with nodes and edges labeled with numbers]

<table>
<thead>
<tr>
<th></th>
<th>E</th>
<th>I</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0</td>
<td>16</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>V1</td>
<td>20</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>V2</td>
<td>13</td>
<td>27</td>
<td>-14</td>
</tr>
<tr>
<td>V3</td>
<td>16</td>
<td>32</td>
<td>-16</td>
</tr>
<tr>
<td>V4</td>
<td>4</td>
<td>20</td>
<td>-16</td>
</tr>
<tr>
<td>V5</td>
<td>11</td>
<td>14</td>
<td>- 3</td>
</tr>
</tbody>
</table>
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.
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.
Example

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

Kohonen Networks

- Self-organizing Maps (SOM)
- Developed by T. Kohonen (Helsinki)
- Mathematical model to explain the self-organization 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

\[
g_{11} g_{21} g_{n1} \quad g_{12} g_{22} g_{n2} \quad g_{1n} g_{2n} g_{nn}
\]

\[
\sum\quad \sum\quad \sum
\]

\[
w_{11} w_{21} w_{m1} \quad w_{12} w_{22} w_{m2} \quad w_{1n} w_{2n} w_{mn}
\]

Barry Linnert, linnert@inf.fu-berlin.de, Cluster Computing SoSe 2020
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 \( \mathbf{x} \) 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 \( x \to \infty \) and 0 for \( x \to -\infty \).
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:

\[
\|\vec{w}_{i^*} - \vec{x}\| = \min_{i=1}^{n}\|\vec{w}_i - \vec{x}\|
\]

with

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

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

\[
\pi_w : \vec{x} \mapsto i^* = \pi_w(\vec{x})
\]
Excitation and Adaptation

Learning rule: \[ \tilde{W}_i^\text{new} = \tilde{W}_i^\text{old} + \varepsilon \cdot h_{i^*,i} \cdot (\bar{X} - \tilde{W}_i^\text{old}) \]
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.

<table>
<thead>
<tr>
<th></th>
<th>Kohonen Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>initialize $w_{ji}$</td>
</tr>
<tr>
<td>2</td>
<td>while (not stopping_condition) do</td>
</tr>
<tr>
<td>3</td>
<td>select $\tilde{x} \in \mathcal{X}$ according to $p(\tilde{x})$</td>
</tr>
<tr>
<td>4</td>
<td>determine $i^<em>$ with $|\tilde{w}_{i^</em>} - \tilde{x}| = \min_{i=1}^{n}|\tilde{w}_i - \tilde{x}|$</td>
</tr>
<tr>
<td>5</td>
<td>$w_{ji} = w_{ji} + \varepsilon \cdot h_{i^*,i} \cdot (x_j - w_{ji}) \quad \forall i, j$</td>
</tr>
<tr>
<td>6</td>
<td>end while</td>
</tr>
<tr>
<td>7</td>
<td>end</td>
</tr>
</tbody>
</table>
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

Processor graph → Mapping → Program graph

Barry Linnert, linnert@inf.fu-berlin.de, Cluster Computing SoSe 2020
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


