UNIVERSITÄT BERN

5TH INT. WORKSHOP ON MLG, FIRENZE, 2007

SPEEDING UP GRAPH EDIT DISTANCE COMPUTATION WITH A BIPARTITE HEURISTIC

Kaspar Riesen and Horst Bunke
riesen@iam.unibe.ch
Institute of Computer Science and Applied Mathematics
University of Bern, Switzerland

UNIVERSITÄT BERN

Outline

- Graph edit distance
- Tree search for graph edit distance
- Munkres' algorithm
- Munkres' algorithm as a heuristic for graph edit distance
- Experimental results
- Conclusions

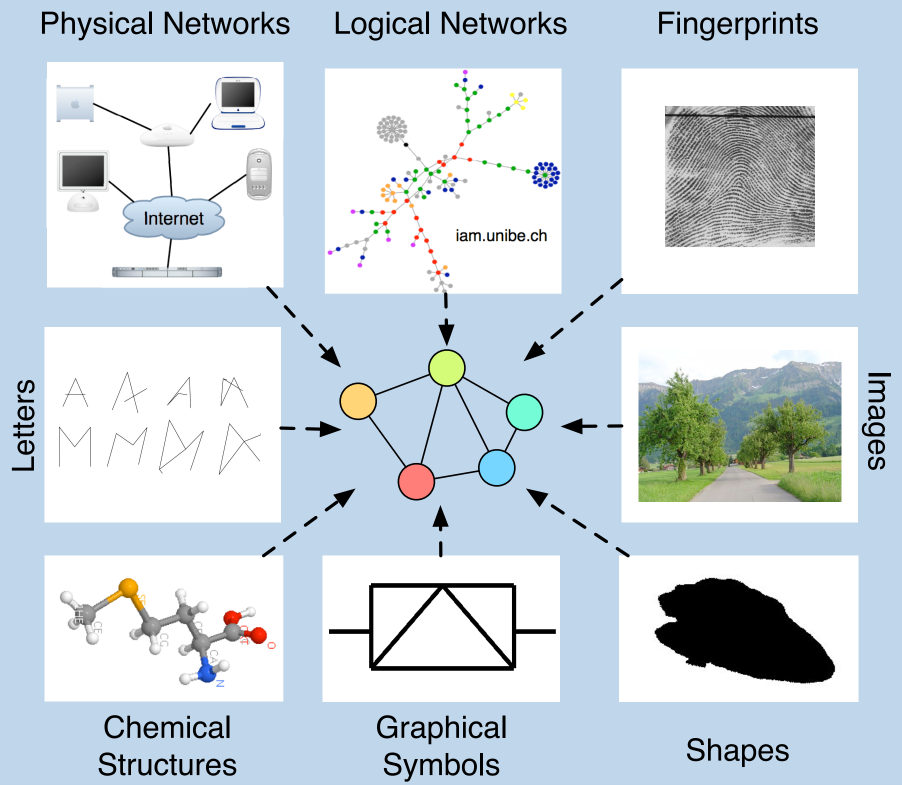
Outline

UNIVERSITÄT BERN

- Graph edit distance
- Tree search for graph edit distance
- Munkres' algorithm
- Munkres' algorithm as a heuristic for graph edit distance
- Experimental results
- Conclusions

- **Main contribution:** We provide a new heuristic for speeding up graph edit distance computation.

Graph Based Representation



A graph g is defined by the 4-tuple $g = (V, E, \mu, \nu)$, where

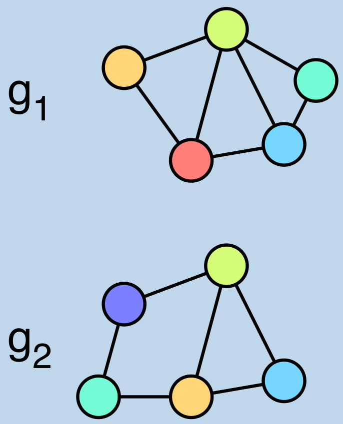
- V is the finite set of nodes
- $E \subseteq V \times V$ is the set of edges
- $\mu : V \rightarrow L$ is the node labeling function
- $\nu : E \rightarrow L$ is the edge labeling function

$L = \{1, 2, 3, \ldots\}$, $L = \mathbb{R}^n$, or $L = \{\alpha, \beta, \gamma, \ldots\}$.

Graph Edit Distance 1/2

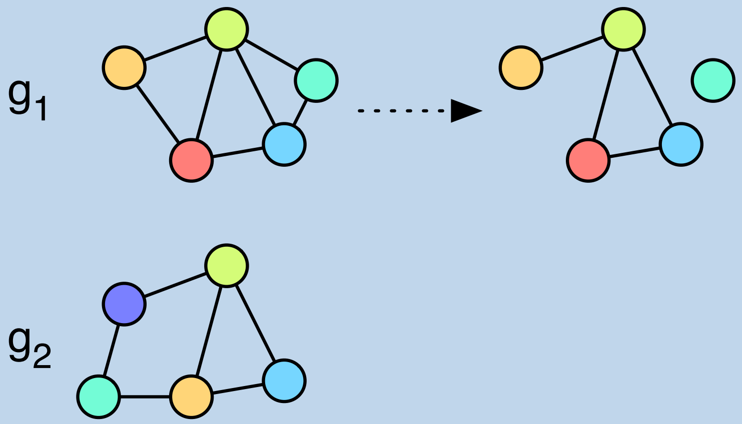
UNIVERSITÄT BERN

- Define the dissimilarity of graphs by the minimum amount of distortion that is needed to transform one graph into another.
- The edit operations e_i consist of deletions, insertions, and substitutions of nodes and edges.



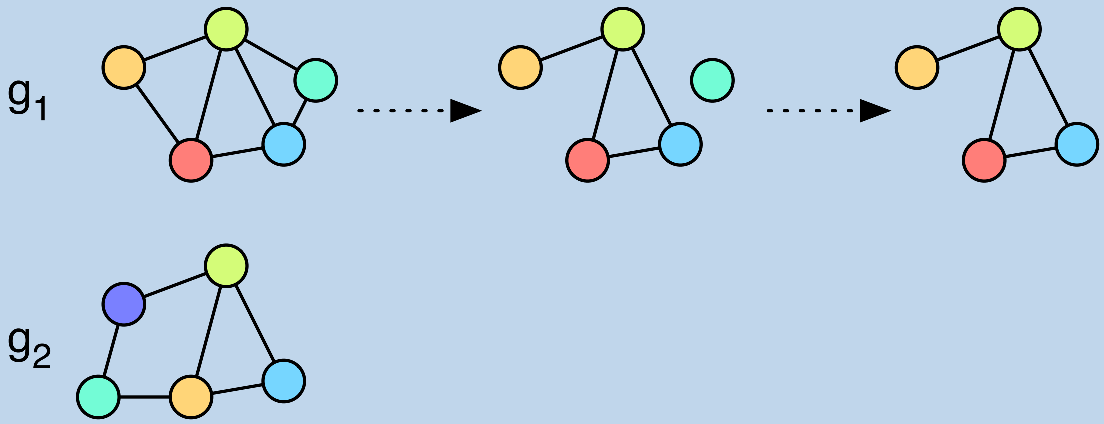
Graph Edit Distance 1/2

- Define the dissimilarity of graphs by the minimum amount of distortion that is needed to transform one graph into another.
- The edit operations e_i consist of deletions, insertions, and substitutions of nodes and edges.



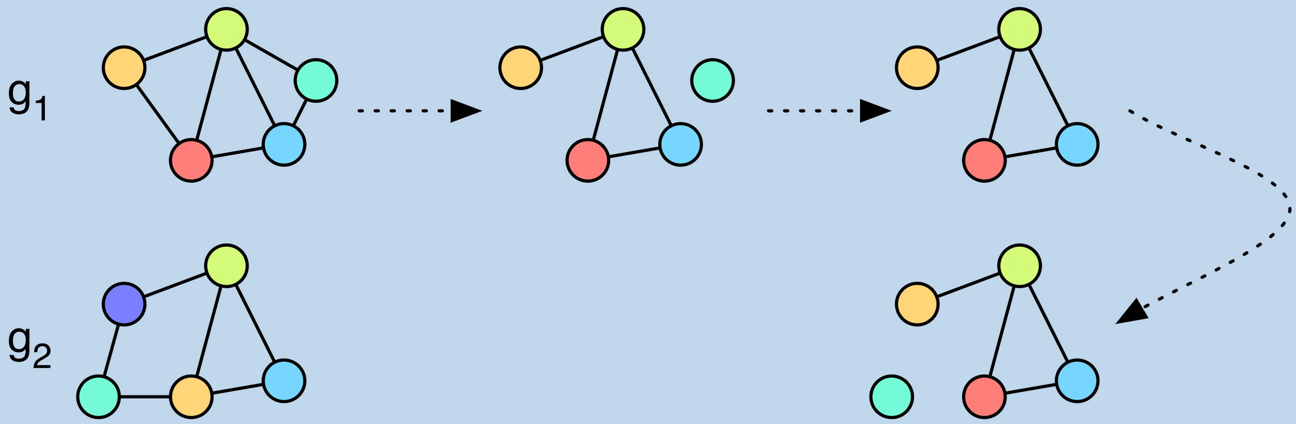
Graph Edit Distance 1/2

- Define the dissimilarity of graphs by the minimum amount of distortion that is needed to transform one graph into another.
- The edit operations e_i consist of deletions, insertions, and substitutions of nodes and edges.



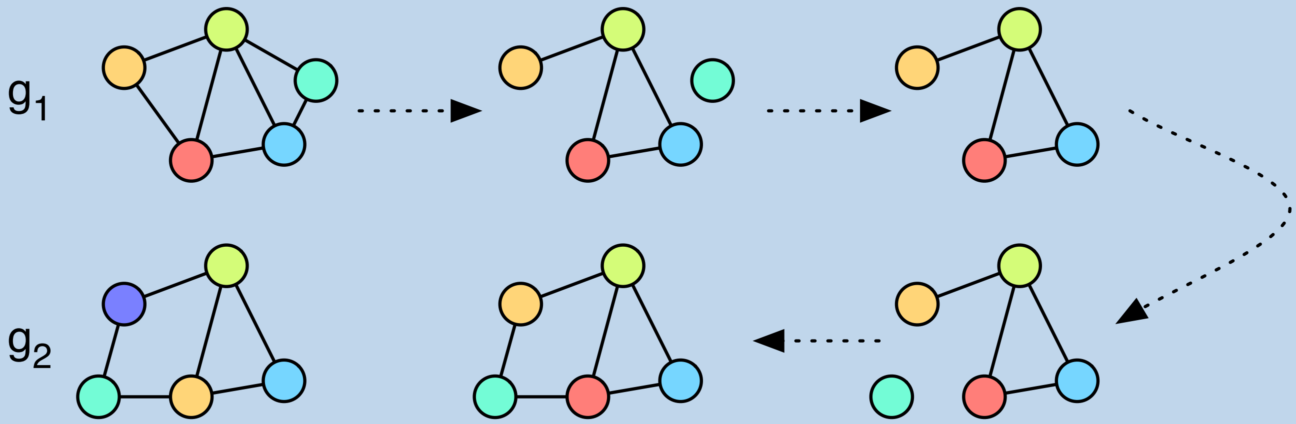
Graph Edit Distance 1/2

- Define the dissimilarity of graphs by the minimum amount of distortion that is needed to transform one graph into another.
- The edit operations e_i consist of deletions, insertions, and substitutions of nodes and edges.



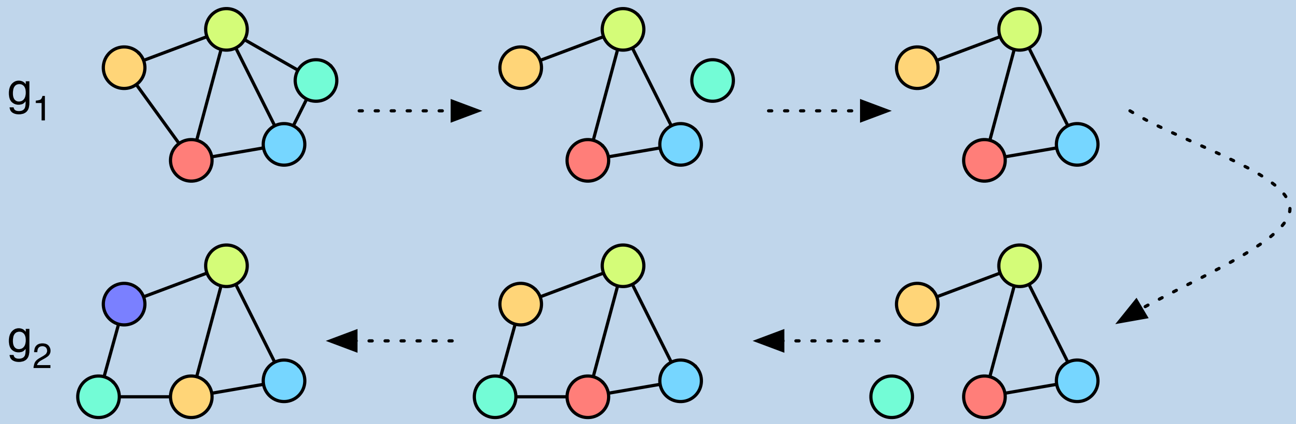
Graph Edit Distance 1/2

- Define the dissimilarity of graphs by the minimum amount of distortion that is needed to transform one graph into another.
- The edit operations e_i consist of deletions, insertions, and substitutions of nodes and edges.



Graph Edit Distance 1/2

- Define the dissimilarity of graphs by the minimum amount of distortion that is needed to transform one graph into another.
- The edit operations e_i consist of deletions, insertions, and substitutions of nodes and edges.



Graph Edit Distance 2/2

- Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ be the source graph and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be the target graph.
- The graph edit distance between g_1 and g_2 is defined by

$$d(g_1, g_2) = \min_{(e_1,\ldots,e_k)\in\Upsilon(g_1,g_2)} \sum_{i=1}^{k} c(e_i),$$

where $\Upsilon(g_1, g_2)$ denotes the set of edit paths transforming g_1 into g_2, and c denotes the edit cost function measuring the strength $c(e_i)$ of edit operation e_i.

Graph Edit Distance 2/2

UNIVERSITÄT BERN

- Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ be the source graph and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be the target graph.
- The graph edit distance between g_1 and g_2 is defined by

$$d(g_1, g_2) = \min_{(e_1,\ldots,e_k)\in\Upsilon(g_1,g_2)} \sum_{i=1}^{k} c(e_i),$$

  where $\Upsilon(g_1, g_2)$ denotes the set of edit paths transforming g_1 into g_2, and c denotes the edit cost function measuring the strength $c(e_i)$ of edit operation e_i.
- **Graph edit distance provides us with a general dissimilarity model for graphs.**

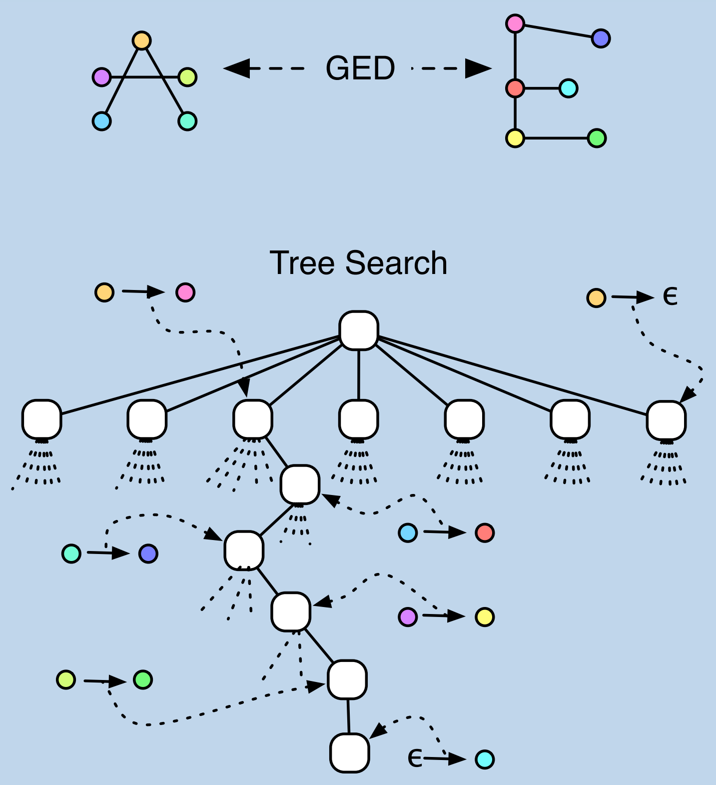
Applications of Graph Edit Distance

- **Classifiers Applicable in the Graph Domain**
  - k-NN classifier
- **Edit Distance Based Graph Kernels**
  - Trivial graph kernels in conjunction with SVM, e.g. $\kappa(g, g') = exp(-d(g, g'))$
  - Graph kernels based on graph edit distance, e.g. Random Walk Edit Kernel [Neuhaus, 2006]
  - Graph embedding in real vector spaces by means of prototype selection [Riesen and Bunke, 2007]
- **Graph Clustering**

Complexity of Graph Edit Distance

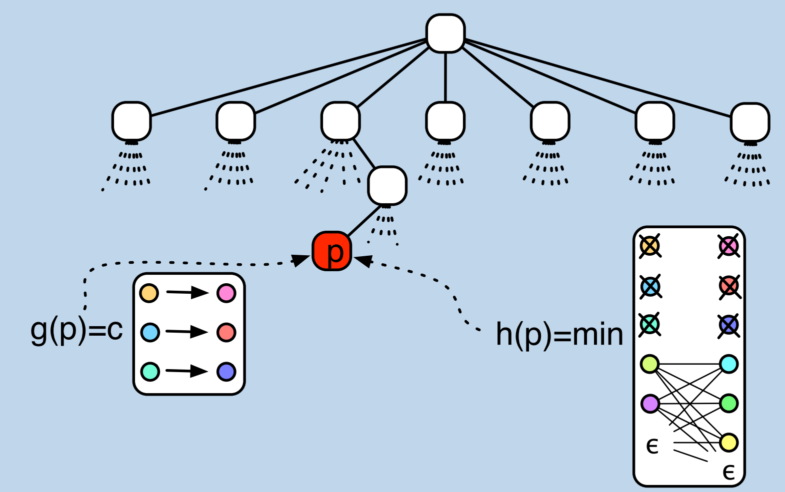
- In contrast with exact graph matching algorithms, the nodes of the source graph can potentially be mapped to any node of the target graph.
- The computational complexity for edit distance is exponential in the number of nodes of the involved graphs. (For graphs with unique node labels the complexity is linear.)
- Graph edit distance is usually computed by a tree search algorithm which explores the space of all possible mappings of the nodes and edges of g_1 to the nodes and edges of g_2.
- Note that edit operations on edges are implied by edit operations on their adjacent nodes.

Tree Search



- Underlying search space is a tree.
- Search tree is constructed dynamically at runtime by creating successor nodes linked by edges to the currently considered node.
- A heuristic function is usually used to determine the node p used for further expansion.

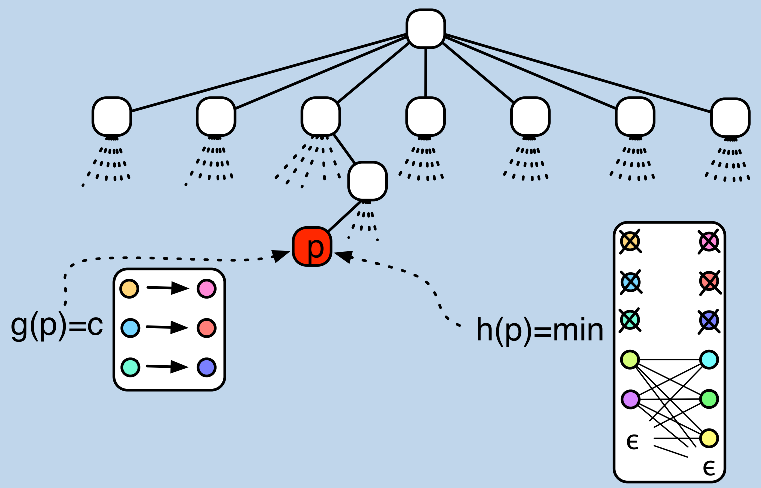
Tree Search Heuristics



- For each node p in the search tree $g(p) + h(p)$ is computed.
- $g(p)$: Cost of the partial edit path accumulated so far.
- $h(p)$: Estimated lower bound for the costs from p to a leaf node.

- $h(p) = 0$: efficient but inaccurate estimation.
- $h(p) =$ exact GED to leaf node: accurate estimation but inefficient.

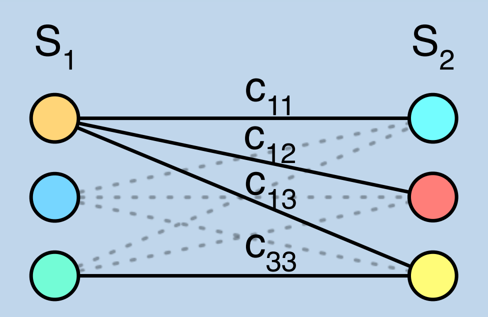
Tree Search Heuristics



- For each node p in the search tree $g(p) + h(p)$ is computed.
- $g(p)$: Cost of the partial edit path accumulated so far.
- $h(p)$: Estimated lower bound for the costs from p to a leaf node.

- $h(p) = 0$: efficient but inaccurate estimation.
- $h(p) =$ exact GED to leaf node: accurate estimation but inefficient.
- How do we estimate a lower bound of the future cost **efficiently** and **accurately**?

The Assignment Problem 1/2



- Find an optimal assignment of n elements of a set $S_1 = \{u_1, \dots, u_n\}$ to n elements of a set $S_2 = \{v_1, \dots, v_n\}$.
- Let c_{ij} be the costs of the assignment $(u_i \rightarrow v_j)$.
- The optimal assignment is a permutation $p = (p_1, \dots, p_n)$ of the integers $1, \dots, n$ that minimizes $\sum_{i=1}^{n} c_{ip_i}$.

The Assignment Problem 2/2

UNIVERSITÄT BERN

- Given the $n \times n$ matrix (c_{ij}) of real numbers corresponding to the assignment ratings.
- The assignment problem can be stated as finding a set of n independent elements of (c_{ij}) such that the sum of these elements is minimum.

p	$\sum_{i=1}^{n} c_{ip_i}$
1 2 3	7
1 3 2	**6**
2 1 3	**6**
2 3 1	8
3 1 2	7
3 2 1	10

1	2	3
2	4	3
3	2	2

Munkres' Algorithm

- Munkres' algorithm finds the best, i.e. the minimum cost, assignment in $O(n^3)$ time.
- It finds an $n \times n$ matrix (b_{ij}) equivalent to the initial one (a_{ij}) having n independent zero elements.

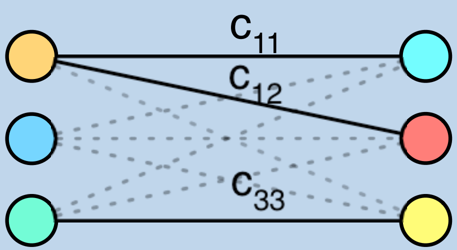
1	2	3
2	4	3
3	2	2

0	1	2
0	2	1
1	0	0

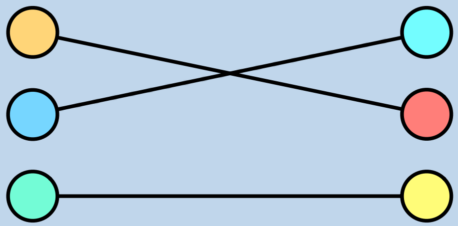
0*	1	2
0	2	1
1	0*	0

. . .

0	0*	1
0*	1	0
2	0	0*

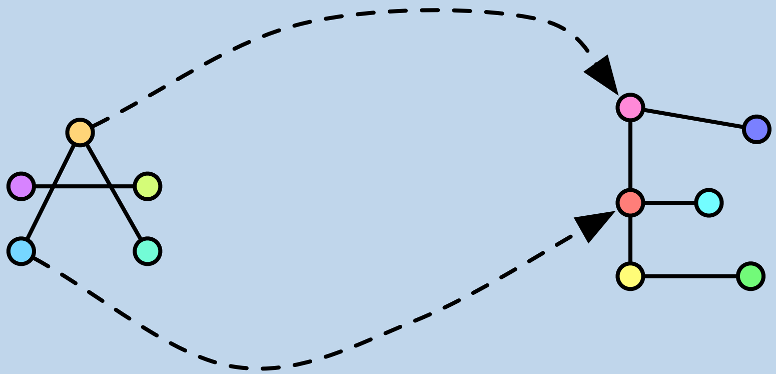


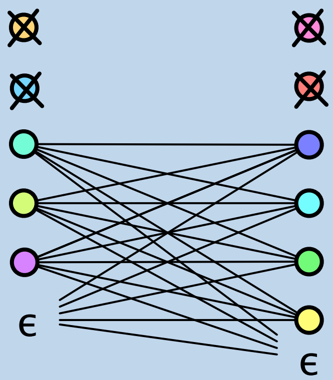
1	2	3
2	4	3
3	2	2



Munkres' Algorithm as a Heuristic

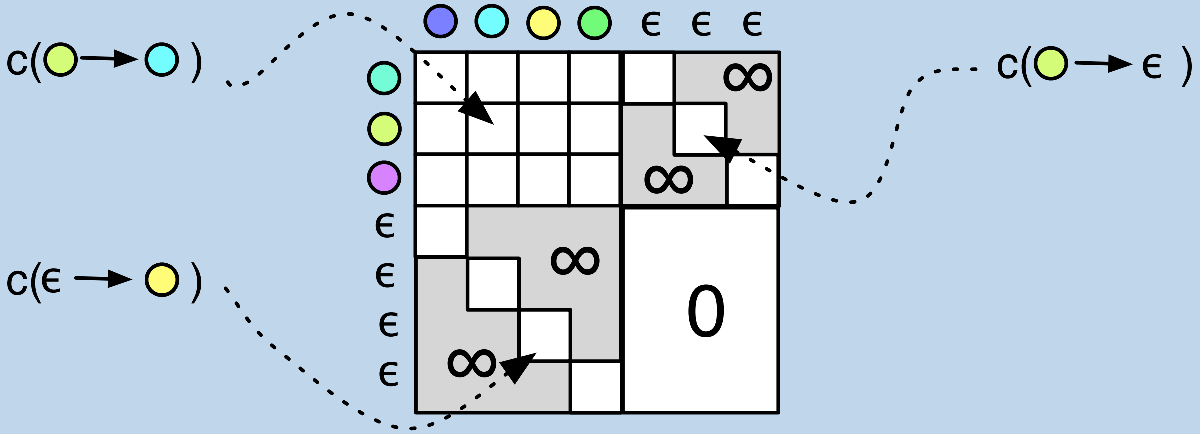
- The problem of estimating a lower bound $h(p)$ for the costs from the current node p to a leaf node can be seen as an assignment problem:
- How can one assign the unprocessed nodes of graph g_1 to the unprocessed nodes of g_2 such that the resulting edit costs are minimal?





Node Cost Matrix

- $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$ are the unprocessed nodes of g_1 and g_2. Define an $(n+m) \times (n+m)$ node cost matrix $\mathbf{C}_n$.
- The left upper corner represents the costs of all possible node substitutions.
- The diagonal of the right/left upper/bottom corner represents the costs of all possible node deletions/insertions.



Bipartite Heuristic

UNIVERSITÄT BERN

- We construct an edge cost matrix $\mathbf{C}_e$ analogously.
- For each open node p in the search tree we run Munkres algorithm twice: Once with $\mathbf{C}_n$ and once with $\mathbf{C}_e$.
- The accumulated minimum cost of both assignments serves us as a lower bound for the future costs to reach a leaf node.
- $h(p) = \textit{Munkres}(\mathbf{C}_n) + \textit{Munkres}(\mathbf{C}_e)$.

Experimental Setup

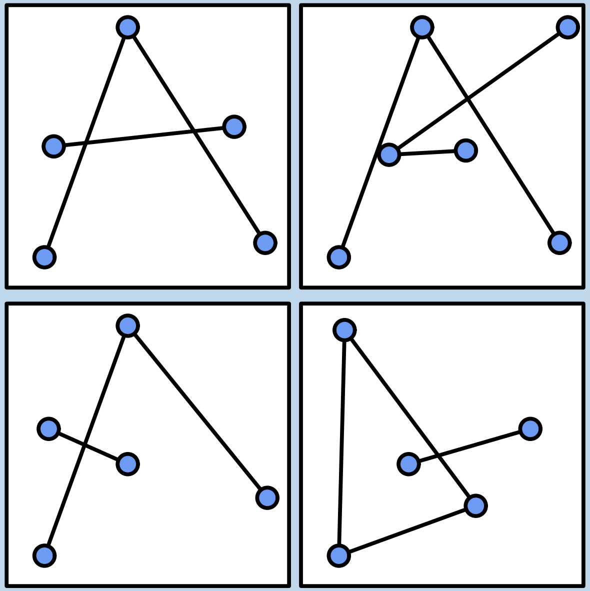
UNIVERSITÄT BERN

- We use four different graph datasets: *Letter*, *Image*, *Fingerprint*, and *Molecule*.
- We compute the edit distance between graphs with and without bipartite heuristic.
- We measure the mean computation time and the mean number of open paths in the search tree during the graph matching process.

Letter Dataset

UNIVERSITÄT BERN

- Graphs representing capital letter line drawings, 15 classes, $562'500$ matchings, $\varnothing|V| = 4.6$, $\varnothing|E| = 4.5$



Method	Time [ms]	OPEN
Plain-A*	465	478
BP-A*	14	72

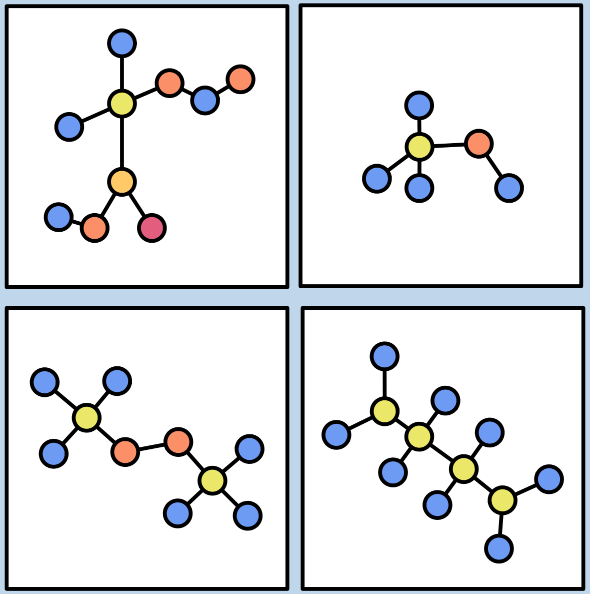
Image Dataset

- Graphs representing images, 5 classes (city, countryside, people, snowy, streets), $26'244$ matchings, $\varnothing|V| = 2.7$, $\varnothing|E| = 2.4$

Method	Time [ms]	OPEN
Plain-A*	0.5	9
BP-A*	0.5	4

Molecule Dataset

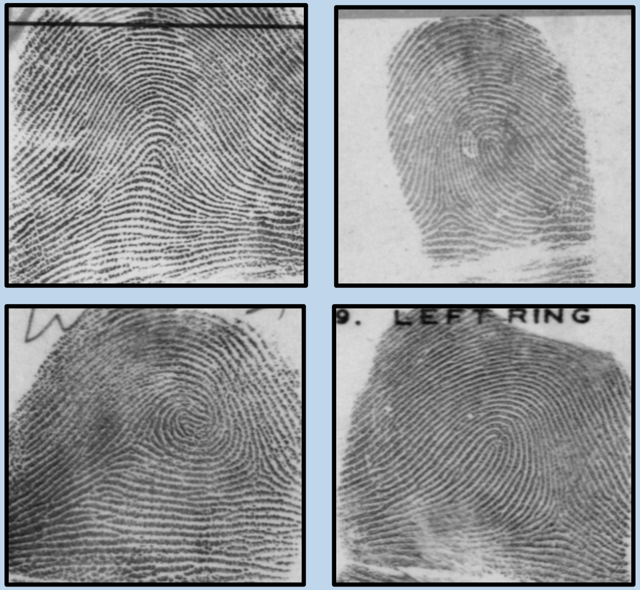
- Graphs representing molecules, 2 classes (active and inactive), $21'300$ matchings, $\varnothing|V| = 5.5$, $\varnothing|E| = 4.7$



Method	**Time [ms]**	**OPEN**
Plain-A*	3799	2195
BP-A*	2	18

Fingerprint Dataset

- Graphs representing fingerprint images, 4 classes (arch, left loop, right loop, whorl), $65'025$ matchings, $\varnothing|V| = 5.4$, $\varnothing|E| = 4.4$



Method	Time [ms]	OPEN
Plain-A*	6140	2465
BP-A*	374	507

Summary

UNIVERSITÄT BERN

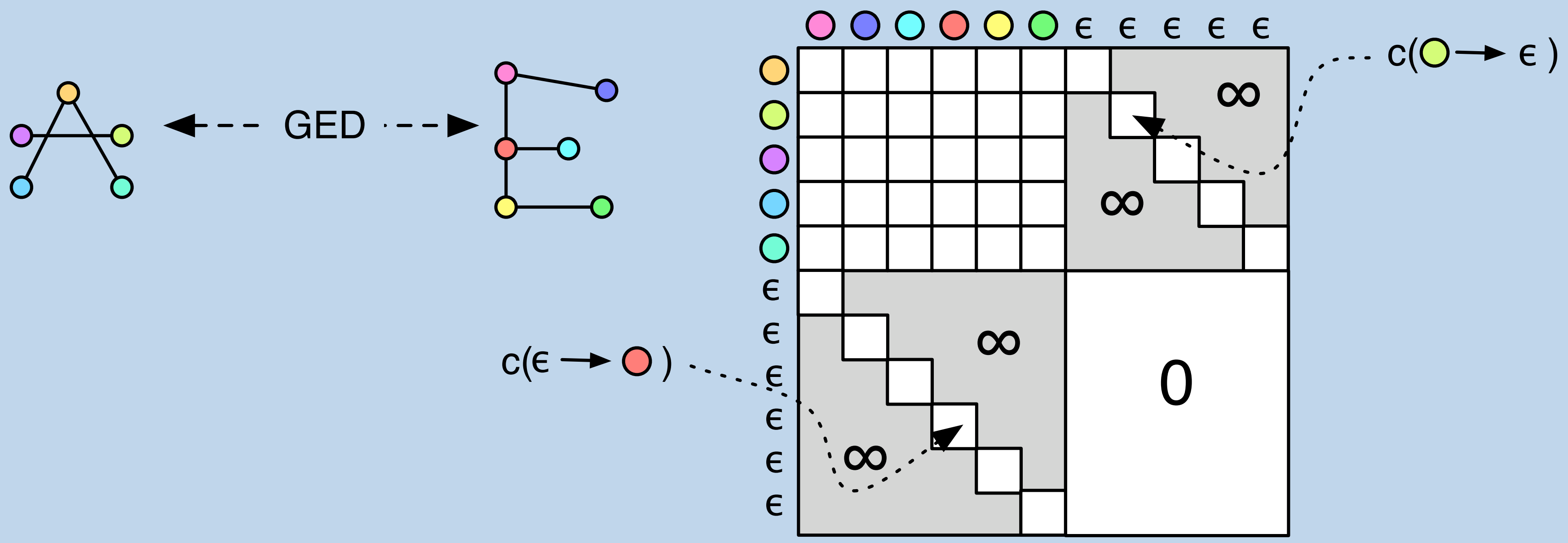
- Thanks to the bipartite heuristic we can achieve significant speed-ups for **exact** graph edit distance.
- Further speed-ups can be achieved if we resort to **suboptimal** algorithms.

Summary

- Thanks to the bipartite heuristic we can achieve significant speed-ups for **exact** graph edit distance.
- Further speed-ups can be achieved if we resort to **suboptimal** algorithms.
- **Transform the bipartite heuristic $h(p)$ into a suboptimal graph matching procedure.**

Fast Suboptimal Edit Distance 1/2

- Define node cost matrix for whole graphs g_1 and g_2.



Fast Suboptimal Edit Distance 2/2

UNIVERSITÄT BERN

- Munkres' algorithm finds the optimal node assignment by considering node operations or the local structure only.
- The implied edge operations are added at the end of the computation.
- Consequently, the edit distance found by Munkres' algorithm need not necessarily correspond to the exact edit distance.
- However, a significant speed-up can be expected.

Fast Suboptimal Edit Distance 2/2

- Munkres' algorithm finds the optimal node assignment by considering node operations or the local structure only.
- The implied edge operations are added at the end of the computation.
- Consequently, the edit distance found by Munkres' algorithm need not necessarily correspond to the exact edit distance.
- However, a significant speed-up can be expected.
- **Future Work:** Find out whether or not the suboptimal distance remains sufficiently accurate for pattern recognition and machine learning applications.

Conclusions

- We propose a new heuristic based on Munkres' algorithm for speeding up graph edit distance.
- Our heuristic finds an optimal node and an optimal edge assignment for the unprocessed nodes and edges of both graphs in polynomial time.
- Our heuristic helps in speeding up exact graph edit distance substantially.
- The proposed heuristic can also be used for fast suboptimal graph matching.