3.3 Exact Algorithm

Using Def. 3.1, the problem of evaluating the structural similarity of graphs is turned into the problem of finding a minimum-cost edit path between graphs. The computation of an optimal edit path is usually performed by means of a search tree algorithm [Bunke and Allermann (1983); Tsai and Fu (1979)]. The idea is to represent the optimal edit path problem in a search tree, such that an optimal path through the search tree is equivalent to an edit path solution. Formally, inner nodes of the search tree correspond to partial edit paths and leaf nodes to complete edit paths. The search tree is constructed by processing one node from the first graph after the other. For each node, all possible edit operations are considered, and newly created partial solutions are inserted into the tree. In practice, the A* best-first search algorithm [Hart *et al.* (1968)] is often used for tree traversal. That is, from the current set of edit path candidates, the most promising one, the one with minimum cost, is chosen at each step of the tree traversal. To speed up the search, a heuristic function estimating the expected cost of the best route from the root node through the current node of the search tree to a leaf node can be used. If the estimated costs are always lower than the real costs, the search procedure is known to be admissible, that is, an optimal path from the root node to a leaf node is guaranteed to be found [Hart *et al.* (1968)]. For the graph edit distance problem, a number of heuristic functions estimating the expected costs of the unprocessed parts of two graphs have been proposed [Bunke and Allermann (1983); Tsai and Fu (1979)].

A best-first search algorithm for the computation of graph edit distance is given in Alg. 3.1. The algorithm is formulated in terms of node edit operations only, but the mapping of edges can easily be inferred from the mapping of their adjacent nodes. Note that the nodes of the first graph are processed in the order $(u_1, u_2, \ldots)$. The set OPEN contains all partial edit paths constructed so far. In each step, the next unprocessed node u_{k+1} of the first graph is selected and tentatively substituted by all unprocessed nodes of the second graph (line 11) as well as deleted (line 12), thus producing a number of sucessor nodes in the search tree. If all nodes of the first graph have been processed, the remaining nodes of the second graph are inserted into the graph in a single step (line 14). The procedure always selects the most promising partial edit path in OPEN (line 5) and terminates as soon as a complete edit path is encountered (line 7).

Unlike exact graph matching methods, the edit distance allows every

Algorithm 3.1 Edit distance algorithm

Input: Non-empty graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$, where $V_1 = \{u_1, \ldots, u_{|V_1|}\}$ and $V_2 = \{v_1, \ldots, v_{|V_2|}\}$

Output: A minimum-cost edit path from g_1 to g_2
e.g. $p = \{u_1 \to v_3, u_2 \to \varepsilon, \ldots, \varepsilon \to v_2\}$

1: Initialize OPEN to the empty set
2: For each node $w \in V_2$, insert the substitution $\{u_1 \to w\}$ into OPEN
3: Insert the deletion $\{u_1 \to \varepsilon\}$ into OPEN
4: **loop**
5: Retrieve minimum-cost partial edit path p_{min} from OPEN
6: **if** p_{min} is a complete edit path **then**
7: Return p_{min} as solution
8: **else**
9: Let $p_{min} = \{u_1 \to v_{i_1}, \ldots, u_k \to v_{i_k}\}$
10: **if** $k < |V_1|$ **then**
11: For each $w \in V_2 \setminus \{v_{i_1}, \ldots, v_{i_k}\}$, insert $p_{min} \cup \{u_{k+1} \to w\}$ into OPEN
12: Insert $p_{min} \cup \{u_{k+1} \to \varepsilon\}$ into OPEN
13: **else**
14: Insert $p_{min} \cup \bigcup_{w \in V_2 \setminus \{v_{i_1}, \ldots, v_{i_k}\}} \{\varepsilon \to w\}$ into OPEN
15: **end if**
16: **end if**
17: **end loop**

node of a graph to be matched to every node of another graph. This flexibility makes graph edit distance particularly appropriate for noisy data, but on the other hand increases the computational complexity in contrast to simpler graph matching models. The time and space complexity of graph edit distance is exponential in the number of nodes of the two involved graphs. This means that for large graphs the computation of edit distance is intractable. In practice, it turns out that edit distance can typically be computed for graphs with up to 12 nodes. Addressing this problem, the next section describes an efficient approximate edit distance algorithm for a common class of graphs.