MINIMAL DEPTH FIRST SEARCH CODES

Chris Wendler
Department of Computer Science
ETH Zurich
Switzerland

In this section we discuss the notion of minimal depth first search (DFS) codes as canonical labels for graphs introduced by Yan & Han (2002). Two graphs have the same canonical label if and only if they are isomorphic.

Definition 1 (Labelled graph)**.** A labelled graph is a 4-tuple $G = (V, E, L, \ell)$, where

- V is a set of vertices,
- $E \subseteq V \times V$ is a set of edges,
- L is a set of labels and
- $\ell : V \cup E \to L$ is a labelling function.

Definition 2 (Isomorphism)**.** An isomorphism between labelled graphs $G = (V, E, L, \ell)$ and $G' = (V', E', L, \ell')$ is a bijective function $f : V \to V'$, such that

1. for all $u \in V$, we have $\ell(u) = \ell'(f(u))$, and
2. for all $(u, v) \in E$, we have $(f(u), f(v)) \in E'$ and $\ell(u, v) = \ell'(f(u), f(v))$.

An automorphism of G is an isomorphism from G to G. A subgraph isomorphism from G to G' is an isomorphism from G to a subgraph of G'. For the sake of simplicity we limit the discussion to the case of undirected vertex and edge labelled graphs.

DFS codes An execution of DFS on a graph G yields a total order on the vertices $<_T$ that corresponds to the order in which they are discovered. This order uniquely determines an associated DFS tree T and can be used to classify edges into forward edges (the ones that are part of the DFS tree) and backward edges (the remaining ones). Note that in undirected graphs there cannot be cross edges.

For the rest of the section we consider an implementation of DFS that whenever it visits a new vertex, processes all backward edges originating from that vertex, in ascending order of their target vertices' discovery. This implementation of DFS gives rise to a total order $<_{E,T}$ on the edges E, the order in which they are processed by the implementation. This total order is formalized in Theorem 1 of Yan & Han (2002).

Using $<_{E,T}$ we can define DFS codes that correspond to the executions of our DFS starting from different root nodes (and selecting different forward edges).

Definition 3 (DFS code)**.** Given a graph G and an associated DFS order on the vertices $v_i <_T v_j$ iff v_i was discovered before v_j, we use an edge sequence $(e_k)_{k=1}^{|E|}$, with $e_k <_{E,T} e_{k+1}$ for all $k = 1, \ldots, |E| - 1$, to define a *DFS code* $code(G, <_T) = ((i_k, j_k, \ell(v_{i_k}), \ell(v_{i_k}, v_{j_k}), \ell(v_{j_k}))_{k=1}^{|E|}$, where $e_k = (v_{i_k}, v_{j_k})$, $\ell(v)$ is the label of vertex v and $\ell(u, v)$ is the label of edge (u, v).

Minimal DFS codes Now, in order to obtain a canonical graph label, we simply define a total order on the set of DFS codes and subsequently compute the DFS code that is minimal with respect to this order. This can be done by, first, defining a total order $<_e$ on the 5-tuples that constitute the entries, e.g., by simply using the lexicographic order on 5-tuples or by using Definition 5 of Yan & Han (2002), and, second, using that total order $<_e$ to define a lexicographic order on the set of all DFS codes $Z = \{code(G, <_T) : G \text{ is a graph and } <_T \text{ is a DFS order of } G\}$. Namely, for DFS

codes $\alpha = (a_1, \ldots, a_m) \in Z$ and $\beta = (b_1, \ldots, b_n) \in Z$, we have

$$\alpha \leq_Z \beta \text{ iff } \begin{cases} \text{there exists a } t \text{ with } 1 \leq t \leq \min(m,n) \text{ and } a_k = b_k \text{ for } k < t \text{ and } a_t <_e b_t, \\ \text{or } a_k = b_k \text{ for } 1 \leq k \leq m \text{ and } n \geq m. \end{cases} \quad (1)$$

Now we can define the notion of minimal DFS code.

Definition 4 (Minimal DFS code)**.** Given a graph G, let $Z(G)$ denote the set of all DFS codes of that graph. The minimal DFS code of G based on $\leq_Z$, $\min(Z(G))$, is called *minimal DFS code*. It is a canonical label of G.

The minimal DFS code of a given labelled graph is computed by enumerating all DFS codes that could potentially be minimal, which can be done by always choosing the forward edges with the smallest label first. To be a bit more precise:

Let v_k denote the kth vertex discovered by DFS. In the step in which v_k is discovered, we call v_k the rightmost vertex and we call the path $p = (v_1, \ldots, v_k)$ from the root v_1 to v_k via forward edges the rightmost path. Now in order to compute the minimal DFS code of G, we set our "minimal DFS code" α to the empty code and start for each (directed) edge with minimal label $(\ell(u), \ell(u,v), \ell(v))$ the following recursion with rightmost path $p = (u)$ and empty DFS code $\beta = ()$:

1. Update the rightmost path and current DFS code by adding vertex v / the edge (u, v). Let's denote the resulting rightmost path by $p = (v_1, \ldots, v_k)$.
2. Add all unused backward edges starting from v_k to the current DFS code, s.t., (v_k, v_i) is added before (v_k, v_j) if $i < j$.
3. For every vertex v_i in the rightmost path in descending order (from v_k to v_1), for every unused forward edge starting from v_i with minimal label (among the ones left), start the recursion with $(v_1, \ldots, v_i)$ as rightmost path and copy of the current DFS code β as DFS code.

If at any step the minimal DFS code candidate α is nonempty and smaller than the current partial code β (i.e., $\alpha <_Z \beta$), we stop processing this code further. Thus, whenever above recursion generates a full DFS code we need to update $\alpha = \beta$, which after checking all candidates will be the minimal DFS code of G.

REFERENCES

X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proc. of IEEE International Conference on Data Mining*, pp. 721–724. IEEE, 2002.