Labeling Schemes for Flow and Connectivity

Julio Pérez
Distributed Computing Seminar, WS 2003/04
ETH Zurich

Paper

- Labeling Schemes for Flow and Connectivity (extended abstract)
  M. Katz, N. Katz, A. Korman, D. Peleg
  SODA (Symposium of Discrete Algorithms) 2002

Outline

- Problem and Motivation
- Labeling Schemes, Flow and Connectivity
- Flow Labeling Schemes
- Vertex-Connectivity Labeling Schemes
- Discussion

- Problem and Motivation
- Labeling Schemes, Flow and Connectivity
- Flow Labeling Schemes
- Vertex-Connectivity Labeling Schemes
- Questions and Discussion

Problem and Motivation

- Network representation
  - Goal: Cheaply store useful information about a network
  - Examples for useful information:
    - Vertex adjacency
    - Distance
    - Tree ancestry
    - ...
  - Particularly important for large and geographically dispersed networks
  - Traditional network representations
    - Vertices with names that contain no useful information
    - Global representation of the network

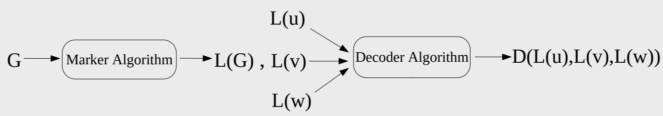
Problem and Motivation (2)

- Labeling schemes proposed in this paper
  - Use of more informative labels for network vertices
    - Flow
    - (Vertex-)Connectivity
  - Localized labels that allow to infer information directly from the labels of the vertices
  - Relatively short labels, i.e. length polylogarithmic in n (n = number of vertices in graph)

- Problem and Motivation
- Labeling Schemes, Flow and Connectivity
- Flow Labeling Schemes
- Vertex-Connectivity Labeling Schemes
- Questions and Discussion

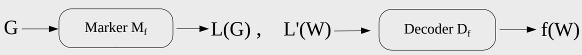
Labeling Schemes

- A vertex-labeling of a graph G is a function L assigning a label L(u) to each vertex u of G
- A labeling scheme has two components
  - Marker algorithm M
    - Given a graph G, selects a label assignment L = M(G)
  - Decoder algorithm D
    - Given a set L' = { L_1, ..., L_K } of labels, returns a value D(L')
    - Time complexity is required to be polynomial in input size



Labeling Schemes (2)

- f labeling scheme
  - Let f be a function defined on sets of vertices in a graph
  - Given a family Ĝ of weighted graphs, an f-labeling scheme for Ĝ is a marker-decoder pair (M_f, D_f) with following properties:
    - Consider G ∈ Ĝ and let L = M_f(G) be the vertex labeling assigned by the marker M_f to G
    - Then for any set of vertices W = { v_1, ..., v_k } in G, the value returned by the decoder D_f on the set of labels L'(W) = { L(v) | v ∈ W } satisfies D(L'(W)) = f(W)



Labeling Schemes (3)

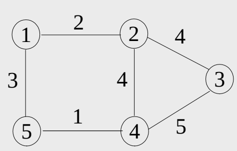
- For a labeling L for the graph G = (V,E) let |L(u)| denote the number of bits in the string L(u)
- $\mathcal{L}_M$(G) = $\max_{u \in V}$ |L(u)| for a given G and a marker algorithm M
- For a finite graph family Ĝ, set $\mathcal{L}_M$(Ĝ) = max { $\mathcal{L}_M$(G) | G ∈ Ĝ }
- Finally, given a function f and a graph family Ĝ, let $\mathcal{L}$(f, Ĝ) = min { $\mathcal{L}_M$(Ĝ) | ∃D, (M,D) is an f labeling scheme for Ĝ }

Flow

- Let G be a weighted undirected graph G = (V, E, w)
- For every edge e ∈ E, the weight w(e) represents the capacity of the edge (e.g. capacity = bandwidth)
- For two vertices u,v ∈ V, the maximum flow flow(u,v) is defined as follows (paper definition):
  - Maximum flow in a path p = (e_1, ..., e_m) is the max. value that does not exceed the capacity of any edge e in p, i.e. flow(p) = $\min_{1 \le i \le m}${ w(e_i) }
  - A set of paths P in G is edge-disjoint if each edge e ∈ E appears in no more than one path p ∈ P
  - The max. flow in a set P of edge-disjoint paths is flow(P) = $\sum_{p \in P}$ flow(p)
  - flow(u,v) = $\max_{P \in P_{u,v}}${ flow(P) }, where $P_{u,v}$ is the collection of all sets P of edge-disjoint paths between u and v

Flow (2)

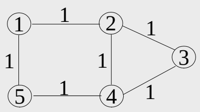
- Instead of demanding that the paths have to be edge-disjoint, demand that for the flow between two nodes u,v the edge capacities have to be respected, i.e. $flow_{in}$(e) ≤ w(e) (aggregated over all p_i ∈ P), for all edges e ∈ p_i, p_i ∈ P



flow	*1*	*2*	*3*	*4*	*5*
1	∞	3	3	3	4
2		∞	9	9	3
3			∞	9	3
4				∞	3
5					∞

Edge-Connectivity

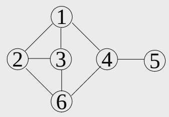
- Edge-connectivity
  - e-conn(u,v) = flow(u,v) assuming each edge is assigned one capacity unit



e-conn	1	2	3	4	5
1	∞	2	2	2	2
2		∞	2	3	2
3			∞	2	2
4				∞	2
5					∞

Vertex-Connectivity

- Vertex-connectivity
  - A set of paths P connecting the vertices u and v in G is vertex-disjoint if each vertex except u and v appears in at most one path p ∈ P
  - v-conn(u,v) of two vertices u,v in an unweighted graph equals the cardinality of the largest set P of vertex-disjoint paths connecting them



v-conn	1	2	3	4	5	6
1	–	3	3	2	1	3
2		–	3	2	1	3
3			–	2	1	3
4				–	1	2
5					–	1
6						–

- Problem and Motivation
- Labeling Schemes, Flow and Connectivity
- Flow Labeling Schemes
- Vertex-Connectivity Labeling Schemes
- Questions and Discussion

Equivalence Relations

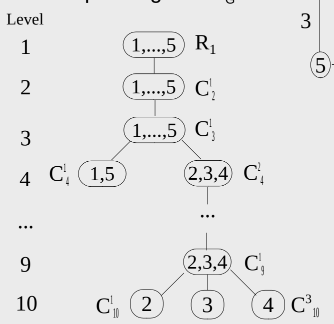
- We consider the family $\hat{G}(n,\hat{w})$ of undirected, capacitated and connected n-vertex graphs with maximum integral capacity $\hat{w}$
- Given $G = (V,E,w) \in \hat{G}$ and $1 \leq k \leq \hat{w}$, define the following relation
  - $R_k = \{ (x,y) \mid x,y \in V, \text{flow}(x,y) \geq k \}$
    - R_k is an equivalence relation
      - Reflexive (flow(x,x) ≥ k)
      - Symmetric (flow(x,y) ≥ k ↔ flow(y,x) ≥ k)
      - Transitive (flow(x,y) ≥ k and flow(y,z) ≥ k → flow(x,z) ≥ k)
    - For every k ≥ 1, R_k induces a collection of equivalence classes on V, $C_k = \{ C^1_{k,}, \ldots, C^m_k \}$, such that $C^i_k \cap C^j_k = \emptyset$ and $\cup_i C^i_k = V$
      (equivalence class = subset whose elements are related to each other by an equivalence relation)

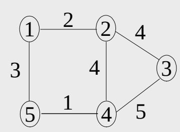
Basic Idea

- Given G, construct a tree T_G corresponding to G's equivalence relations
- k^{th} level of T_G corresponds to the relation R_k
- Each node at a level k represents an equivalence class
- Nodes representing equivalence classes with one element are leaves

Basic Idea (2)

- Corresponding tree T_G



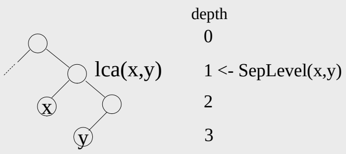


flow	1	2	3	4	5
1	∞	3	3	3	4
2		∞	9	9	3
3			∞	9	3
4				∞	3
5					∞

If max. capacity of any edge is $\hat{w}$, then depth of T_G cannot exceed $\hat{w}$ levels?

Separation Level

- For two nodes x,y in a tree T with root r, the separation level of x and y $SepLevel_T(x,y)$ is defined as the depth of the least common ancestor of x and y, lca(x,y)

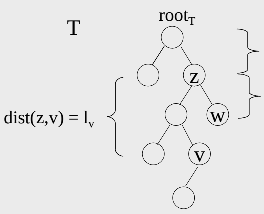


- Let t(u) be the leaf in T_G associated with the singleton set {u}
- Lemma 1: $flow_G(u,v) = SepLevel_{T_G}(t(u),t(v)) + 1$, where u,v in V

Separation Level Labeling Scheme

- For the class T(n) of n-node unweighted trees, there exists a SepLevel labeling scheme with O(log²n)-bit labels ([1])
  - Based on a given distance labeling scheme (M_{dist}, D_{dist}) for T(n)
  - $M_{SepLevel}$
    - Let L be the labeling assigned by M_{dist} for a T in T(n)
    - $M_{SepLevel}$ augments each label L(v) into L'(v) = (L(v),depth(v))
  - $D_{SepLevel}$
    - Consider v,w in T with z = lca(v,w), l_v = dist(z,v), l_w = dist(z,w), l_z = depth(z)
    - Given the labels L'(v) = (L(v),depth(v)) and L'(w) = (L(w),depth(w)), dist(v,w) = D_{dist}(L(v),L(w)) = $l_v + l_w$
    - Moreover depth(v) = D_{dist}(L(v),L($root_T$)) = $l_v + l_z$ and depth(w) = D_{dist}(L(w),L($root_T$)) = $l_w + l_z$
      -> $D_{SepLevel}$ can deduce SepLevel(v,w):
      $D_{SepLevel}$ (L'(v),L'(w)) = depth(z) = (depth(v) + depth(w) - dist(v,w)) / 2

Separation Level Labeling Scheme (2)



z = lca(v,w), depth(z) = l_z

dist(z,w) = l_w

dist(v,w) = D_{dist}(L(v),L(w)) = $l_v + l_w$
depth(v) = D_{dist}(L(v),L($root_T$)) = $l_v + l_z$ and
depth(w) = D_{dist}(L(w),L($root_T$)) = $l_w + l_z$
$D_{SepLevel}$ (L'(v),L'(w)) = depth(z) =
= (depth(v) + depth(w) - dist(v,w)) / 2

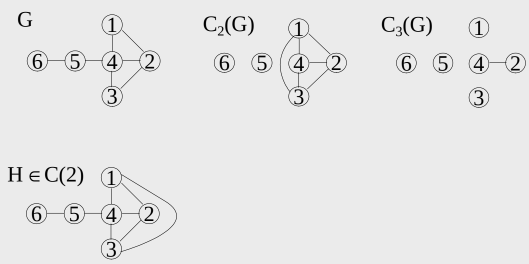
from L'(v) from L'(w) from D_{dist}(L(v),L(w))

- Problem and Motivation
- Labeling Schemes, Flow and Connectivity
- Flow Labeling Schemes
- Vertex-Connectivity Labeling Schemes
- Questions and Discussion

K-Connectivity

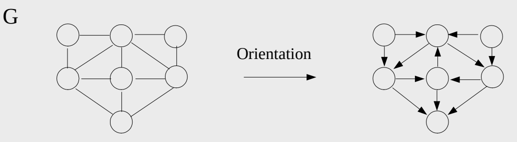
- Unweighted, undirected n-vertex graphs
- Two vertices are called k-connected if there exist at least k vertex-disjoint paths between them
- The k-connectivity graph of G = (V,E) is $C_k(G)$ = (V,E'), where (u,v) $\in$ E' iff u and v are k-connected in G
- A graph G is closed under k-connectivity if it has the property that if u and v are k-connected in G then they are neighbors in G, i.e. $C_k(G)$ is a subgraph of G. C(k) denotes the family of graphs which are closed under k-connectivity

K-Connectivity (2)



K-Orientability

- A graph G is called k-orientable if there exists an orientation of the edges such that the out-degree of each vertex is bounded above by k. $\mathcal{J}_{or}$(k) denotes the class of k-orientable graphs



-> G ∈ $\mathcal{J}_{or}$(2)

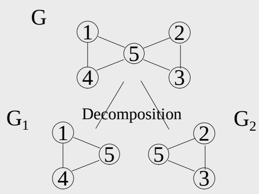
Basic Idea

- Labeling k-connectivity for some graph G is equivalent to labeling adjacencies for C_k(G)
  - Labeling k-connectivity / adjacencies means constructing a marker-decoder pair (M,D), such that D(L(u),L(v)) = 1 iff u and v are k-connected / adjacent in G, 0 otherwise (L is the vertex labeling assigned to G by M)
- Moreover $C_k(G) \in C(k)$ (without proof)

  -> Instead of presenting a k-connectivity labeling scheme for general graphs, present an adjacency labeling scheme for the graphs in C(k)

Basic Idea (2)

- General idea for labeling adjacencies for some G in C(k) is to decompose G into simpler graphs
  - We say that a graph G can be decomposed into the graphs $G_i = (V_i,E_i)$, i > 1, if $\bigcup_i V_i = V$, $\bigcup_i E_i = E$ and the E_i's are pairwise disjoint

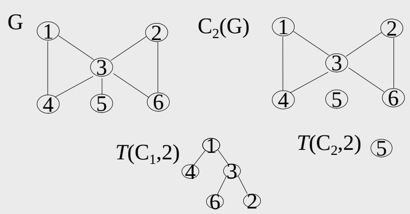


- Make use of leftmost Breadth-First Search (BFS) trees

Leftmost BFS tree

- Let C be a connectivity component of C_k(G) for a graph G (for two vertices u,v in C there exists a path between them)
- A leftmost BFS for C, denoted T(C,k), is a BFS tree spanning C, constructed as follows
  - Take a vertex r from C as root of T(C,k), set level(r) = 1
  - Assuming we constructed i levels of T(C,k) and there are still unused vertices of C, repeatedly take a vertex v of level i and connect it to all the unused vertices w adjacent to it in C_k(G). Set level(w) = i + 1 (v is the parent of w in T(C,k))

Leftmost BFS tree (2)



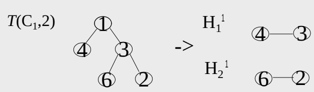
- It's easy to see that for k = 2 and a vertex u ∈ G, the only neighbor of u that has a strictly lower level than u in $T(C_i,2)$ is the parent of u in $T(C_i,k)$

2-Connectivity Labeling Scheme

- As already mentioned, labeling 2-connectivity for a family of graphs Ĝ is equivalent to labeling adjacencies for the family C(2)
- G ∈ C(2) can be decomposed into a forest F and a graph H of disjoint cliques
  - Let $C_1, \ldots, C_m$ be the components of G
  - Fix i and let $T = T(C_i,2)$, then each subgraph H_j^i of C_i induced by level j of T is in C(1)

    -> H_j^i is a collection of disjoint cliques
  - Forest F = { $T(C_i,2)$ | C_i is a component of G }

    H = { H_j^i | for all i's and j's }



2-Connectivity Labeling Scheme (2)

- Let $C_n(2)$ be the family of n-vertex graphs in C(2)
- Marker algorithm $M_{adjacency,C(2)}$
  - Assume each vertex has a unique identity from 1 to n
  - Decompose G into a forest F and a graph H of disjoint cliques
  - To each clique C in H give a distinct identity from the range {1, ..., n}, id(C)
  - For a vertex u in G denote p(u) u's parent in F and C(u) the clique in H containing u
  - L(u) = (id(C(u)), id(p(u)), id(u)), where each id is log(n)-bit long
    -> 3log(n)-bit labels

2-Connectivity Labeling Scheme (3)

- Decoder algorithm $D_{adjacency,C(2)}$
  - Given L(u) and L(v) for u,v in V(G), compare id(p(u)) with id(v) and id(p(v)) with id(u) to check whether one is the parent of the other in F
  - Furthermore we compare id(C(u)) and id(C(v)) to see whether u and v are neighbors in H
  - D(L(u),L(v)) = 1 iff one of the cases above applies, 0 otherwise
  - Correctness: u and v are neighbors in G iff they are neighbors in F or H

3-Connectivity Labeling Scheme

- Idea similar to 2-connectivity labeling scheme
- Labeling 3-connectivity for a family of graphs $\hat{G}$ is equivalent to labeling adjacencies for the family C(3)
- Consider a graph G in C(3), and let $C_1, ..., C_m$ be its connected components. It is clear that C_i is in C(3) for all i
- Let $T(C_i,3)$ for a certain i
- Lemma 2: Each vertex u in $T(C_i,3)$ has at most one neighbor of G which has a strictly lower level than u in $T(C_i,3)$ apart from p(u) (see construction of leftmost BFS tree)

3-Connectivity Labeling Scheme (2)

- Decompose G element of C(3) into a graph H $\in$ C(2) and a 2-orientable graph
  - Proof for H $\in$ C(2) similar to the proof of the decomposition of G for 2-connectivity labeling scheme
  - Let U be the graph C after deleting the edges of H (H = union of all subgraphs H_j of C induced by the vertices of level j in $T(G,3)$)
    - By Lemma 2 each vertex u of U has at most 2 neighbors of a strictly lower level
      -> Direct the edges of U from higher level to lower level vertices
      -> Each u has out-degree at most 2
      -> U is 2-orientable

3-Connectivity Labeling Scheme (3)

- Assuming we have $(M_1,D_1) = (M_{adjacency,C(2)}, D_{adjacency,C(2)})$ and $(M_2,D_2) = (M_{adjacency,J2(2)}, D_{adjacency,J2(2)})$
  - Marker algorithm $M_{adjacency,C(3)}$
    - $L(u) = (L_1(u),L_2(u))$
  - Decoder algorithm $D_{adjacency,C(3)}$
    - Given the two labels $L(u) = (L_1(u),L_2(u))$ and $L(v) = (L_1(v),L_2(v))$ let $D(L(u),L(v)) = D_1(L_1(u),L_1(v))$ or $D_2(L_2(u),L_2(v))$

K-Connectivity Labeling Scheme

- Not shown in this presentation
- Idea
  - Again labeling k-connectivity for a family of graphs $\hat{G}$ is equivalent to labeling adjacencies for the family C(k)
  - Each G in C(k) can be decomposed into two graphs in C(k -1) and a (k -1)-orientable graph

Conclusion

- Some labeling schemes for flow and vertex-connectivity
- Quite a lot of definitions, lemmas and theorems
- Various labeling schemes not presented
- A few mistakes
- Few figures!

References

- [1] David Peleg, Informative labeling schemes for graphs, in Proc. 25th Symp. on Mathematical Foundations of Computer Science, vol. LNCS-1893, Springer-Verlag, Aug. 2000, pp. 579-588