I. Burdonov, A. Kossatchev.
Building direct and back spanning trees by automata on a graph.
French-Russian Seminar on Software Verification, Testing, and Quality Estimation, Paris, France, November 24-25, 2014.
см.Труды Института системного программирования РАН Том 26-6. 2014 г., стр. 57-62.
6 стр.

Building direct and back spanning trees by automata on a graph

Igor Burdonov <igor@ispras.ru>, Alexander Kossatchev <kos@ispras.ru>

Introduction

A task of graph exploration with a goal to uncover a structure of unknown graph by moving along its arcs can be met in many domains. In this paper we regard as possible applications exploration of networks and exploration of Web-application structure. In many cases such an exploration can be considered as being performed by agents working in graph vertices and sending each other messages along graph arcs.

Graph exploration starts from some specified vertex, called root vertex. Directed graph exploration isn't a trivial task. In the worst case it takes time of an order $O(mn)$, where n — the number of graph vertices, m — the number of graph arcs. This holds for various algorithms of graph exploration based on breadth-first or depth-first traversal [1,2,3].

In 1966 M. O. Rabin posed the problem of directed graph exploration with a finite automaton [4]. Automaton on a graph is an analogue of the Turing machine — tape cells correspond to graph vertices, where the automaton can store some data, and moves along the tape correspond to moves along graph arcs. This system can be considered also as an aggregate of finite automatons located in graph vertices and interacting by message sending. Each automaton changes its state according to the data stored in the corresponding vertex, and moves along graph arcs are replaced with messages sent by the automaton of the arc's starting vertex to the one of the ending vertex.

Messages are both input and output symbols of vertex automata. If message size and number of each automaton states are bounded globally, all automata are just finite state machines. Usual graph exploration corresponds to possibility for a single message to have a size linear on the number of vertices. The most efficient known algorithm of graph exploration with a single finite automaton is suggested in [5] and has worst case working time $O(nm+n^2 loglogn)$. If the traversal is repeated by message interchange between automata located in vertices known after the first exploration worst case working time becomes $O(nm+n^2 l(n))$, where $l(n)$ is the number of times logarithm calculation is repeated until $1 \le log(log...(n)...) < 2$ holds [6].

In this paper we consider parallel exploration of a graph — many messages can walk through its arcs in parallel. Working time estimation in this case depends on *the capacity* of an arc k, which means the maximum number of messages that can be transmitted through a single arc simultaneously.

The suggested parallel graph exploration algorithm has worst case working time bound $O(n/k+D)$, where n is the number of vertices, and D is the graph diameter, the maximum length of simple path (non-self intersecting path). As a result the algorithm builds two spanning trees of the graph: *the direct spanning tree*, which has the root vertex as its tree root and is directed from the root, and *the back spanning tree*, directed to the root. The detailed description and proofs of all statements can be found in [7].

Spanning trees building algorithm

Sending a message an automaton working in some vertex should specify the arc, by which this message should be delivered. We count that arcs starting in some vertex are numbered and an arc is specified by its

number. We also suppose that operation time of an automaton is negligibly small and the time of message transport through an arc is bounded by some constant.

Let us denote *s* the maximum number of arcs starting in the same vertex of the graph, then $m \le ns$. We suggest the algorithm of spanning trees building with the following features:

- vertex automaton memory is bounded by $\boldsymbol{O}(nD\log s)$,
- message size is bounded by $\boldsymbol{O}(D\log s)$,
- arc capacity is *k*,
- algorithm worst case working time is $\boldsymbol{O}(n/k + D)$.

Below we call *direct arcs* the arcs belonging to the direct spanning tree constructed by the algorithm, *chords* — all other arcs, *back arcs* — arcs belonging to the back spanning tree. A back arc can be a chord or a direct arc. *Path vector* is the list of arc numbers along the path. *Vertex vector* is the vector of the simple path. leading to this vertex from the root along the direct spanning tree. The root has empty vertex vector ε. The size of simple path vector or vector of a simple cycle is $\boldsymbol{O}(D\log s)$. A message in the algorithm below consists of a constant number of simple path vectors, or $\boldsymbol{O}(D\log s)$ bits.

The algorithm is partitioned in four parts. The first part describes building of the back spanning tree, the second part performs the check that back spanning tree is completely constructed, the third part depicts classification of arcs starting in one vertex, the fourth one describes maintenance of incoming back arc counters in vertices.

The first part uses messages of four kinds: **Start**, **Root search**, **Direct,** and **Back**. **Start** message is sent by the root automaton to automata of all other vertices, it contains the vertex vector and initiates vertex automaton operation, which is started by sending **Root search** messages. **Root search** messages are sent through all outgoing arcs in such a way that they pass some simple path to the root vertex and report the root automaton the vector of this simple path. In response root vertex automaton sends **Direct** message, which reaches the **Root search** initiator and reports it the back simple path vector, computed by root search. The initiator then sends **Back** message, which sets some tags along the back simple path.

The second part has the task to determine that the back spanning tree is completely constructed. It uses arc counting — when the arc counter in the root vertex becomes zero, the tree building is complete. For each arc $a \to b$ the root automaton gets first the message from *a*, where "+1" is set for the arc $a \to b$, and then it gets the message from *b*, where "-1" is set for $a \to b$. The second message comes from *b* later than messages from the same *b*, setting "+1" for all arcs starting in *b*. For this goal the modified **Root search** and two additional message kinds — **Finish** and **Minus** — are used. **Root search** contains the number of arcs starting from its initiator. When the root automaton gets **Root search** message, it adds this number to the global counter of outgoing arcs. **Finish** message is sent from a vertex after getting **Direct** message. After sending **Finish** message and setting back arc in the vertex, the **Minus** message is send. The root automaton decreases its arc counter by 1 after getting **Minus** message.

During the third part of the algorithm the outgoing arcs from each vertex are marked as either direct arcs or chords. At first they all are considered as chords. Then each arc, along which **Direct** message is sent, is marked as a direct arc.

During the fourth part the incoming back arc counters are set in all vertices. Two message kinds — **Start counting** and **End counting** — are used for that. **Start counting** messages move from the root along

direct arcs to all other vertices, **End counting** message move from each vertex along back arcs and each vertex counts such messages created in the start vertices of back arcs ending in this vertex.

Conclusion

At the end of algorithm work each vertex automaton stores type of each outgoing arc and the number of incoming back arcs. These data can further be used for parallel computation of some functions of values stored in graph vertices.

References

1. Steven S. Skiena. The Algorithm Design Manual. Springer-Verlag, New York, 1997.
2. I. B. Burdonov, A. S. Kossatchev, V. V. Kuliamin. Irredundant Algorithms for Traversing Directed Graphs: The Deterministic Case. Programming and Computer Software, 29(5):245-258, 2003.
3. I. B. Burdonov, A. S. Kossatchev, V. V. Kuliamin. Irredundant Algorithms for Traversing Directed Graphs: The Nondeterministic Case. Programming and Computer Software, 30(1):2-17, 2004.
4. M.O. Rabin. Maze Threading Automata. An unpublished lecture presented at MIT and UC, Berkeley, 1967.
5. I. B. Burdonov. Traversal of an unknown directed graph by a finite automaton. Programming and Computer Software, 30(4): 11-34, 2004.
6. I. B. Burdonov. Backtracking on a tree in traversal of an unknown directed graph by a finite automaton. Programming and Computer Software, 30(6): 6-29, 2004.
7. I. B. Burdonov, A. S. Kossatchev, V. V. Kuliamin Parallel computations on graphs. Programming and computer Software, 2015 г., No 1 (in print).