ISSN 0361-7688, Programming and Computer Software, 2015, Vol. 41, No. 6, pp. 307–310. © Pleiades Publishing, Ltd., 2015.
Original Russian Text © I.B. Bourdonov, A.S. Kossatchev, V.V. Kulyamin, 2015, published in Programmirovanie, 2015, Vol. 41, No. 5.

Analysis of a Graph by a Set of Automata

I. B. Bourdonov, A. S. Kossatchev, and V.V. Kulyamin

Institute for System Programming, Russian Academy of Sciences, ul. Solzhenitsyna 25, Moscow, 109004 Russia
e-mail: igor@ispras.ru, kos@ispras.ru, kuliamin@ispras.ru

Received May 12, 2015

Abstract—An algorithm of traversal (retrieval of full information on the structure) of an a priori unknown directed graph by an unbounded set of finite automata that interact through the exchange of messages and can move along the arcs of the graph according to their direction is described. Under the assumption that the execution time of basic operations and the transmission time of individual messages are bounded, the total operating time of the algorithm is bounded, at worst, by $O(m + nd)$, where n is the number of vertices of the graph, m is the number of its arcs, and d is the diameter of the graph; moreover, this estimate is unimprovable. The full proofs of the propositions formulated in this paper have been published in [6].

DOI: 10.1134/S0361768815060031

1. INTRODUCTION

The problem of retrieving the structure of an unknown graph by successive traversals along its edges, starting from a certain vertex, arises in various fields. Examples are given by the analysis of the structure of computer networks, the interface of Web applications, flows of control and data in a code, as well as by the construction of other automaton models of systems. In the present study, we consider a similar problem for the case of a directed graph analyzed by a set of agents none of which can contain full information on the graph, but all together they can do it.

Each agent can be represented as a finite automaton with bounded memory and a finite set of possible actions. These actions include the retrieval of the identifier of a current vertex, iteration of the arcs of the graph emanating from the current vertex, motion along the next arc of the graph, as well as exchange of messages with other agents, for example, to find out if some of them was at the current vertex of this agent and whether the arc emanating from this vertex was used. Since the graph is not bounded initially, there is a possibility of creation of new agents, which are situated at some initial vertex of the graph. A graph is considered to be analyzed if, according to the information available to agents, one can establish that each arc has been traversed at least once, and, thus, all its vertices and arcs are known to the current set of agents. In this case it is not assumed that information on the structure of the graph is provided explicitly; it suffices that it can be retrieved without execution traversals long arcs.

A similar statement of the problem has practical applications in testing complex systems modeled by automata with a large number of states. In [1], the authors consider the testing of models of microprocessors in which there are millions of states and tens of millions of junctions. In [1, 2], the testing is performed by a limited set of agents each of which has access to full information on the structure of the already studied part of the graph. In the present work, we propose an algorithm for the analysis of a graph by a set of agents each of which works only with information locally available to it. Earlier, in [3–5], algorithms were proposed for the analysis of an a priori unknown graph by a single agent that has either unlimited or limited memory. In these papers, it is also proved that, for graphs on n vertices and m arcs, the operating time of such an agent is, at worst, $O(nm)$. In the present work, we show that, by involving many agents, this estimate can be reduced to $O(m + nd)$, where d is the diameter of the graph.

The further structure of the article is as follows: in the second section, we give a formal statement of the problem of analyzing a graph by a set of agents; then we present the algorithm of operation of such a set of agents and prove an estimate for the operating time of this algorithm in the worst case. In the Conclusions, we summarize the content of the work and list possible directions of development of the algorithm.

2. FORMAL STATEMENT OF THE PROBLEM

Next, we assume that the operational environment of agents supports the operations of creation and destruction of an agent. At any moment of time, each agent is at some vertex of the graph (a few agents can be situated simultaneously at the same vertex) and has a unique identifier (the identifier of the agent is returned by the operation of its creation). There are two possibilities for creating an agent: in one case, the agent is at the distinguished initial vertex after creation; in the second, the agent is created by another agent, as an agent situated at the same vertex where the second agent was situated; after that, the agent creator loses the capability to interact with the graph. The vertices of the graph also have unique identifiers. An agent may receive a command to pass along a certain arc emanating from a vertex at which it is situated. The arc in this case is defined by its number: all arcs emanating from the same vertex are numbered. The operation of traversal along an arc returns the identifier of the vertex at which the agent finds itself and the number of arcs emanating from this vertex. Moreover, agents can exchange messages: an agent can send a message either to all agents or to some specific one, having specified its identifier, and also can receive a message from any sender. The messages are not lost or distorted during transmission and do not overtake each other when transmitted between two agents. In addition to the type of a message (there is a finite number of types of messages), they can also carry a finite number of slots storing the identifiers of states or of agents.

To be able to work with graphs of arbitrary size, we do not impose restrictions on the number of agents involved. To be able to operate with unbounded identifiers of agents or states, we assume that an agent has a finite number of memory cells capable of storing the identifier of a state or of an agent. Only operations of copying between these cells and between a memory cell and a message slot are executed over identifiers; one can also compare the identifiers of states to check if they are equal or unequal. Thus, agents represent expanded finite automata that operate with variable identifiers. Another possible obstacle to the finiteness of agents is the necessity to operate with the numbers of an unbounded amount of arcs emanating from the same vertex. To avoid this, an arbitrary graph is reconstructed into a graph in which the number of the arcs emanating from vertices is bounded by a certain constant R; to this end, it suffices to split each vertex into several vertices. From each new vertex, at most $R - 1$ arcs will emanate, plus one more arc that connects each vertex from the bundle obtained with the next vertex. All vertices in this bundle, except for the first, may have an empty identifier because only one arc enters them, and it is not necessary to look for them among known vertices after passing along this arc; therefore, one can avoid the extension of the set of identifiers.

Thus, suppose given a graph with unknown structure with a fixed initial vertex, and suppose that it is possible to execute the operations described above. It is required to define a program of operations of agents and of the environment so that the set of agents finally obtains full information on the structure of the graph.

The notations used below in this paper are as follows: n is the number of vertices of the graph, m is the number of arcs in it, and d is the length of the longest path from the initial vertex to another vertex (the diameter of the graph).

3. THE ALGORITHM

We propose the following algorithm of operation of agents to solve the problem posed.

• Each agent can operate in two modes: as a vertex manager and as a slider. The slider is created at some vertex, passes a certain path, and finally either becomes a vertex manager or is destroyed. The vertex manager is responsible for storing data about the arcs emanating from the vertex, including yet not passed arcs, and determines where the next slider, which arrived at this vertex, will move. If the number of arcs emanating from a vertex of the graph is not bounded, several vertex managers are created, each of which stores information only on a limited number of outgoing arcs and the address of the next manager. Only the first manager is the manager associated with the identifier of the vertex.

• For each emanating arc, the vertex manager stores its status and the identifier of the manager of the terminal vertex, provided it is known. The status of an arc can be either active, passive, or complete. A yet not passed arc, or an arc along which a slider should be sent because the manager of its terminal vertex sent a request for such a transmission, is considered to be active. An arc along which one should not yet send sliders (in response to the request for a slider, one slider has already been sent) is considered to be passive. A complete arc is that along which one should not send a slider any longer, because a message has been received from the manager of its terminal vertex that all the paths leading from this vertex bring to the already passed vertices. Immediately after the creation of a manager, all the outgoing arcs are active.

The manager of a vertex with nonempty identifier also stores a reference to a next similar manager. Such references are used for processing requests for searching for the manager of a vertex by the identifier of the vertex by browsing the list formed by these references.

The vertex manager stores the number of the arc along which a slider was sent last time. Immediately after the creation of the manager, this number is equal to 0, which means that there is no such arc.

In addition, the vertex manager stores the number of the arc, entering its vertex, when passing along which this vertex was detected for the first time. Such arcs, together with the vertices known at the moment, form a tree—the skeleton of a part of the graph that has already been traversed by all the agents.

• The set of messages that agents exchange and the operations that they can execute is as follows.

◦ Creation of an agent at the initial vertex. It is created as a slider.

◦ Creation of a slider by another agent at the vertex at which this agent is situated. In this case, the second agent loses the capability to interact with the graph. Despite the seeming artificiality, such an operation is easily executed if the agents represent processes: this operation corresponds to the creation of a clone of the process in the current state and to the transfer of rights for operation with the graph to this clone.

◦ Transition along the arc with number *i* at the current vertex of the graph; as a result, the slider appears at the terminal vertex of this arc and gets its identifier and the number of arcs emanating from it.

◦ Request for the number of the arc for further motion from the slider to the manager of the current vertex containing the sender's identifier. The answer to this request contains the number of the arc for executing the transition and the identifier of the manager of the terminal vertex of the arc (which may be empty if the terminal vertex has no manager).

◦ A message on a search for the manager of a vertex with given identifier contains this identifier and the identifier of the agent that sent it. In response to this message, a message can arrive about the available manager with the identifier of this manager.

◦ Request for a new slider for further traversal along the graph. Such a request contains the number of the arc along which the slider and the identifier of the manager of its terminal vertex should be sent.

◦ A message on the completeness of an arc, which means that it does not make sense to go along it any longer, because either it ends at a vertex without emanating arcs or all the emanating arcs lead to already known vertices a transition to which can be organized through other routes.

1. At the beginning of the work, the first created agent becomes the manager of the initial vertex. He gets information on the number of arcs emanating from this vertex. As a vertex manager that just appeared at the vertex, he creates a slider located at the same vertex, delegates the capability of traversing along the arcs of the graph to it, and reports its identifier for exchanging messages.

2. The slider, just created or just learned the identifier of the manager of the current vertex, sends to this manager a request for the number of the arc along which it should move further.

3. Having received a request for the number of an arc for the motion, the vertex manager looks for an active arc following the employed one. This is either an active arc with the minimum number greater than the number of the employed arc, or, if there is no such an arc, simply an active arc with the minimum number. Its number is recorded as the number of the employed arc. If there are no active arcs at all at the current moment, then the number of the employed arc becomes equal to 0.

If the number obtained of the employed arc is not 0, then the slider that sent the request gets in response this number and the identifier of the end of the employed arc if it is known. The employed arc in this case is transferred to the status "passive." Moreover, after that the manager of the initial vertex creates a new slider, and the manager of the other vertex sends to the manager of the initial vertex of the entering arc a request for a new slider.

If the number of the employed arc is 0, the vertex manager stores the slider that sent the request as a waiting slider.

4. The slider that received the number of the arc to pass along it stores the identifier of the manager of the current vertex and performs a transition along this arc. As a result, it receives the identifier of the terminal vertex and the number of arcs that emanate from it.

If the slider also receives a nonempty identifier of the manager of the terminal vertex in the message with the number of arc, then it passes along the already known arc. After that it executes item 2.

If the identifier of the manager of the terminal vertex is empty, then it should be defined.

When a vertex has no outgoing arcs, the vertex is terminal and should not have a manager. In this case, a slider sends to the manager of the initial vertex of the arc along which it just passed a message on the completeness of the arc (it does not make sense to go along it any longer) and is destroyed.

If a vertex has outgoing arcs but its identifier is empty, this is a vertex with the only entering arc. In this case, the slider becomes a manager of this vertex, initializes the data of the manager, and creates a new slider at the current vertex, delegating interaction with the graph to it.

If a vertex has outgoing arcs and the identifier is not empty, the slider starts a search for a vertex manager, sending to the manager of the initial vertex a message on the search for the manager with the identifier of the vertex and with its own identifier as data.

5. Having received the message on the search for a vertex, the manager compares the identifier of the vertex with the identifier of its own vertex.

If the identifiers do not coincide, the manager checks whether there is a next manager in the list for search. If there is no such a manager, then there is no sought-for manager, and the manager sets a reference to the agent with the identifier contained in the message (the slider that sent the identifier) as to a next agent and forwards the message to this agent. If the message on the search for a vertex is received by the slider, then the manager of this vertex is not found, and it itself should become the manager of this vertex. In this case, it initializes the manager's data, creates a new slider at this vertex, and delegates interaction with the graph to this slider.

If the identifiers do not coincide and there is a next manager in the search list, the message on the search is forwarded to this manager.

If the identifiers coincide, then a message on the found manager with the identifier of this manager is sent to the agent with the identifier contained in the message. If the slider receives a message on the found vertex manager with its identifier, it executes item 2.

6. The vertex manager may receive a request for a new slider with the number of one of the arcs emanating from its vertex. In this case, the arc is declared active.

If the manager has a waiting slider at that, then a message with the instruction to go along this arc (with its number and the identifier of the manager of the terminal vertex) is sent to this slider. The arc remains passive, and the reference to the waiting slider is zeroed. If this manager is connected with the initial vertex, it creates a new slider; otherwise a request for a new slider with indication of the number of the entering arc is sent to the manager of the initial vertex of the arc entering the vertex of the given manager. If there is no waiting slider, nothing is done anymore.

7. The vertex manager may receive a message on the completeness of one of the arcs emanating from this vertex from the manager of the terminal vertex of this arc. In this case the arc is marked as a complete one.

If the vertex manager has a waiting slider and has no other incomplete arcs, this slider is destroyed. If this manager is connected with the initial vertex, then the operation of the whole algorithm is completed: the graph is completely traversed; otherwise a message on the completeness of the entering arc is sent to the manager of the initial vertex of the arc entering the vertex of this manager.

If the manager has no waiting slider, or if there are incomplete outgoing arcs, then nothing is done anymore.

In [6], the authors proved that the operating time of the algorithm described, which consists of the execution time of all the operations of agents listed above, including a transition along an arc, as well as the time of transmission of messages between agents bounded by some constant, is limited, at worst, to $O(m + nd)$. The authors also presented an example of a graph on which this estimate is satisfied; i.e., it takes a time of $\Omega(m + nd)$ for the algorithm to traverse the graph.

4. CONCLUSIONS

In this paper, we have presented an algorithm for the traversal (or retrieval of full information on the structure) of an a priori unknown graph by means of an unbounded set of parallel agents that represent finite automata and interact with each other through the exchange of messages. The operating time of this algorithm is $O(m + nd)$ in the worst case, where n is the number of vertices of the graph, m is the number of its arcs, and d is the diameter of the graph, under the assumption that the execution time of some basic operations and the transmission time of individual messages between agents are bounded. A family of graphs is obtained on which this estimate is satisfied. In this work, to facilitate the understanding, the algorithm is represented not in a completely formal form: the proof of the efficiency estimates is omitted. The reader can find a formal account with full proofs in [6].

A further development of the result is the design of a version of the algorithm that is adapted for operation in a multiprocessor system when one (or a few) slider agents and one process–manager that has a function of storing information on some part of the graph operate on each processor (or core). In this case, within a process–manager, it is worth storing information not about a single vertex, but about several vertices; otherwise the costs of maintenance of the operation of many managers will significantly reduce the overall efficiency. Moreover, in this case the exchange of messages between processes should be optimized so that this process occurs largely on a single processor. The details of such an algorithm have not yet been worked out.

Another possible direction in the development of the algorithm is its modification in order to be able to operate with nondeterministic graphs and with external memory. In nondeterministic graphs, one number of an arc emanating from a vertex may correspond to several real arcs, and the transition along the arc with such a number may bring to different vertices at different times. Such graphs are often met during modeling complex software or hardware systems at a high level of abstraction. The use of external memory can remove the requirement that all information on the graph should be located in the memory of the whole set of agents employed.

REFERENCES

1. Demakov, A., Kamkin, A., and Sortov, A., High-Performance Testing: Parallelizing Functional Tests for Computer Systems Using Distributed Graph Exploration, *Open Cirrus Summit 2011*, Moscow, 2011.
2. Bourdonov, I.B., Groshev, S.G., Demakov, A.V., Kamkin. A.S., Kossatchev, A.S., and Sortov, A.A., Parallel testing of large automata models, *Vestn. Nizhni Novgorod Gos. Univ.*, 2001, no. 3, pp. 187–193.
3. Burdonov, I.B., and Kossatchev, A.S., and Kulyamin, V.V., Irredundant algorithms for traversing directed graphs: The deterministic case, *Program. Comput. Software*, 2003, vol. **29**, no. 5, pp. 245–258.
4. Burdonov, I.B., and Kossatchev, A.S., and Kulyamin, V.V., Nonredundant algorithms for traversing directed graphs: Nondeterministic case, *Program. Comput. Software*, 2004, vol. **30**, no. 1, pp. 2–17.
5. I.B. Burdonov, Traversal of an unknown directed graph by a finite robot, *Program. Comput. Software*, 2004, vol. **30**, no. 4, pp. 188–203.
6. Bourdonov, I.B., and Kossatchev, A.S., Traversal of an unknown directed graph by a group of automata, *Tr. Inst. Syst. Program., Ross. Akad. Nauk*, 2014, vol. **30**, no. 2, pp. 43–86.

Translated by I. Nikitin

SPELL: OK