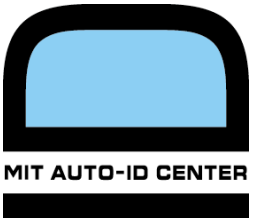


MIT-AUTOID-TR-004

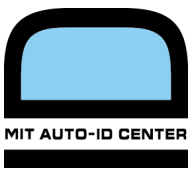
ePC: 21.0000001.05004.XXXXXXXXXX

An O(log n) Randomized Resource Discovery Algorithm

Ching Law, Kai-Yeung Siu

November 2000

Auto-ID Center
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
http://auto-id.mit.edu/



MIT-AUTOID-TR-004
An $O(\log n)$ Randomized Resource Discovery Algorithm *

Ching Law and Kai-Yeung Siu
Massachusetts Institute of Technology
{ching,siu}@list.mit.edu

Abstract

The problem of a distributed network of computers discovering one another by making network connections is called the *resource discovery* problem. In this paper, we present the first randomized algorithm, called '*Absorption*', that can solve the resource discovery problem in $O(\log n)$ running time with high probability and with $O(n^2)$ expected pointer complexity. In particular, the expected running time of our *Absorption* algorithm is at most $4\log_{4/3} n + 1$ steps on any strongly-connected network. We also describe a variation of the algorithm, which runs in $O(\log^2 n)$ expected time but has $O(n)$ expected message complexity.

Keywords: Distributed Algorithms, Resource Discovery

1 Introduction

The resource discovery problem of networked machines seeking and learning about each others is introduced by Harchol-Balter, Leighton, and Lewin in [1]. In a distributed network, if machine u knows the address of machines v, machine u can connect machine v and inform it of other machines known to u. A resource discovery algorithm specifies how the machines should communicate with each other, with the goal that each machine will be aware of all other machines.

There are three performance measures for a resource discovery algorithm:

1. time complexity – number of time steps taken;
2. message complexity – number of messages sent (called connection communication complexity in [1]) ; and
3. pointer complexity – number of pointers (machine addresses) passed (called pointer communication complexity in [1]).

Prior algorithms for resource discovery include the Flooding algorithm, the Swamping algorithm, and the Random Pointer Jump algorithm (all described in the paper by Harchol-Balter *et al* [1]). However, it was shown in [1] that these algorithms do not perform well in certain network topology.

Their paper [1] introduced the Name-Dropper algorithm: During each round, each machine picks a neighbor randomly and passes the neighbor all its known pointers. It was shown that the machines can learn about one another in $O(\log^2 n)$ time, $O(n \log^2 n)$ message complexity, and $O(n^2 \log^2 n)$ pointer complexity, all with high probability.

Kutten and Peleg [3] recently proposed a deterministic resource discovery algorithm with $O(\log n \log^* n)$ time complexity[1], $O(n \log n \log^* n)$ message complexity, and $O(n^2 \log^2 n)$ pointer complexity.

In this paper, we introduce a new randomized algorithm, called *Absorption*, which assumes a strongly-connected graph. Our algorithm runs in $O(\log n)$ time with high probability and achieves $O(n^2)$ expected pointer complexity. A variant with $O(\log^2 n)$ expected running time achieves $O(n)$ expected message complexity. Moreover, we present the first exact analysis of the expected performance of resource discovery algorithms.

*A preliminary version of this work appeared in Brief Announcements of the 14th International Symposium on Distributed Computing, October 4-6, 2000, Toledo, Spain.

This work was supported in part by the MIT Auto-ID Center. Please forward any future correspondence to K.-Y. Siu at siu@list.mit.edu.

[1] $\log^* n$ is the minimum number of times that the logarithm function is applied to n such that the resulting value is less than or equal to 1.

In Section 2, we describe a graph-theoretic model for the resource discovery problem. We introduce the *Absorption* algorithm in Section 3 and analyze its performance in Section 4. Section 5 discusses two variants of the algorithm. We conclude with remarks on future work in Section 6. Because of space limitation, the proofs are omitted in this extended abstract.

2 Resource Discovery

In this section, we discuss how we can model the resource discovery problem by studying distributed algorithms that evolve a connected directed graph into a complete graph. We will also give simple lower bounds on the three performance measures defined in Section 1.

A network can be modeled by a directed graph (V, E), such that each machine is a node in V. If machine u knows about machine v, then there is an edge $(u, v) \in E$.

For any node u, we let $\Gamma(u)$ be the known set of u, i.e. the set of machines known to u:

$$(u, v) \in E \quad \text{iff} \quad v \in \Gamma(u).$$

We always have $u \in \Gamma(u)$. A *message* is a set of nodes. A node u can send messages to any node in set $\Gamma(u)$. When a node u receives a message M from v, u's known set is updated to $\Gamma(u) \cup M$. In this case, we also say node u *passes* a set M of pointers to node v.

The goal of a resource discovery algorithm is to evolve a given connected graph into a complete graph (in other words, every node knows all other nodes: $\Gamma(u) = V$ for all $u \in V$). The algorithm needs to be distributed in the sense that each node runs the algorithm without knowledge of the global state. However, we assume that a global clock is available such that the distributed algorithm in each node can run in discrete steps.

Remark 1. *Any resource discovery algorithm on a strongly-connected graph requires at least* $\log_2(n-1)$ *time steps,* n *messages, and* $n(n-2)$ *pointers passed in the worst case.*

3 The *Absorption* Algorithm

We now describe the algorithm *Absorption*. It assumes a strongly-connected graph as input and consists of 2 stages.

In stage 1, a graph of n nodes is partitioned into *clusters*. Each cluster has one *leader*. All members of each cluster know their leader. For any node v, let $l(v)$ denote the leader of v. Therefore, we have $l(v) \in \Gamma(v)$ for any node v.

For any leader u, $\mathcal{C}(u)$ is the set of nodes that have u as their leader:

$$\mathcal{C}(u) = \{w \mid l(w) = u\}.$$

We will show in Lemma 1 that a leader's known set is the superset of the union of its members' known sets.

In the beginning of stage 1, there are n clusters such that every node is the leader of its single-node cluster. Each round of stage 1 consists of the following four procedures.

1. Each leader u becomes *active* with probability $1/2$. If leader u is active, it randomly chooses a node $v \in \Gamma(u) \setminus \mathcal{C}(u)$ and then passes the set $\Gamma(u)$ to v. Lemma 2 will show that such a node v always exists unless u is the only leader. We call these messages seek messages.

2. Each node v that was contacted by some leader u in the previous procedure now passes the seek message to its leader $l(v)$. (If node v itself is a leader, then it does nothing.)

3. For each leader u, if u was active in procedure 1 or had not received a seek message in procedure 2, then it remains idle in this procedure.

   For each leader u that was inactive in procedure 1 and had received $k > 0$ messages in procedure 2, we will construct a bigger cluster led by u. Let $v_1, \ldots, v_k$ be the leaders that had contacted some members of $\mathcal{C}(u)$ in procedure 1. Leader u informs leaders $v_1, \ldots, v_k$ that they will retire and have u as their new leader. We call these messages retire messages.

   The new cluster is a merge of the original cluster of u and the clusters of these retiring leaders $v_1, \ldots, v_k$. After the merge, $\mathcal{C}(u)$ is updated to contain all members of this new cluster:

   $$\mathcal{C}'(u) = \mathcal{C}(u) \cup \bigcup_{i=1}^{k} \mathcal{C}(v_i),$$

   and $\Gamma(u)$ will absorb the pointers received from $v_1, \ldots, v_k$:

   $$\Gamma'(u) = \Gamma(u) \cup \bigcup_{i=1}^{k} \Gamma(v_i).$$

4. Each retiring leader v informs its members in $\mathcal{C}(v)$ of the new leader specified in the received retire message. If $\mathcal{C}(u) = \Gamma(u)$, then terminates and go to stage 2; otherwise, go to procedure 1.

It can be shown (see Lemma 2) that eventually (with probability 1) $\mathcal{C}(u) = \Gamma(u)$ for some u.

Stage 2: the ultimate leader left in stage 1, who knows about all other nodes in the network, can now broadcast the pointers to the entire network in one time step.

Note that in the first procedure of stage 1, each node independently decides to be active or inactive. This is similar to the pointer-jumping techniques used in parallel algorithms [2].

The *Absorption* algorithm assumes a strongly-connected graph. On a weakly-connected graph, we can run the Name-Dropper algorithm [1] for $O(\log n)$ time steps to obtain a strongly-connected graph with high probability.

4 Performance Analysis

This section analyzes *Absorption*'s asymptotic complexity as well as the upper bounds of its expected performance.

First, we need to extend our notation of Γ for a set of nodes:

$$\Gamma(W) = \bigcup_{w \in W} \Gamma(w).$$

The next lemma states that the known set of a leader is superset of the known sets of the members of its cluster.

Lemma 1. *During stage 1 of the* Absorption *algorithm, we have*

$$\Gamma(\mathcal{C}(u)) = \Gamma(u)$$

for any leader u.

Moreover, any leader knows about at least one node not among its own cluster.

Lemma 2. *During stage 1 of the* Absorption *algorithm, unless there is only one leader left,*

$$\Gamma(u) - \mathcal{C}(u) \neq \emptyset$$

for any leader u.

Using Lemma 2, we can show that the expected number of leaders in the graph is reduced by a constant factor in each round of stage 1.

Lemma 3. *In each round of stage 1 of the* Absorption *algorithm, each leader is retired with probability* $1/4$.

Now we are ready to state our major results.

Theorem 1. *The* Absorption *algorithm terminates in $O(\log n)$ rounds with probability greater than $1 - \frac{1}{n^{O(1)}}$*

Corollary 1. *With high probability, the message complexity of the* Absorption *algorithm is $O(n \log n)$ and the pointer complexity of the* Absorption *algorithm is $O(n^2 \log n)$.*

Next, we derive the upper bounds on the constants of the asymptotic bounds.

Theorem 2. *Running the* Absorption *algorithm on a strongly-connected graph, the expected time steps is $4 \log_{4/3} n + 1$; the expected total number of messages is at most $n \log_{4/3} n + 6n - 2$; and the expected total number of pointers passed is at most $5n^2 + n \log_{4/3} n - 5n$.*

If the given graph is weakly-connected, we can first run the Name-Dropper algorithm for $O(\log n)$ rounds. According to [1], the Name-Dropper algorithm makes $O(n)$ connections per round, and passes $O(n^2)$ pointers per round in high probability.

Table 1 compares the performance of *Absorption* with Name-Dropper and Kutten-Peleg. We can see that *Absorption* has better asymptotic bounds for all of the three complexity measures. We note that both *Absorption* and Name-Dropper are randomized algorithms, but Kutten-Peleg is a deterministic algorithm.

5 Variants

In this section we discuss two variants of the *Absorption* algorithm.

5.1 $O(n^2)$ Pointers on Weakly-Connected Graphs

We can see from Table 1 that, on a weakly-connected graph, the subroutine of evolving the graph to be

	Time	Messages	Pointers
Name-Dropper [1]	$O(\log^2 n)$	$O(n \log^2 n)$	$O(n^2 \log^2 n)$
Kutten-Peleg [3]	$O(\log n \log^* n)$	$O(n \log n \log^* n)$	$O(n^2 \log^2 n)$
Absorption (strong)	$O(\log n)$	$O(n \log n)$	h:$O(n^2 \log n)$, e:$O(n^2)$
Absorption (weak)	$O(\log n)$	$O(n \log n)$	$O(n^2 \log n)$

Table 1: Performance of Name-Dropper, Kutten-Peleg, and *Absorption* in terms of the three complexity measures. The bounds on row "*Absorption* (strong)" assume a strongly-connected graph. The bounds on row "*Absorption* (weak)" assume a weakly-connected graph and Name-Dropper is run for $O(\log n)$ rounds before the start of *Absorption*. The pointer complexity of *Absorption* is $O(n^2 \log n)$ with high probability, and $O(n^2)$ in expectation.

strongly-connected is the bottleneck of the *Absorption* algorithm's pointer complexity. We describe a method to improve the pointer complexity at the cost of higher message complexity. Instead of Name-Dropper, we can use the following simple algorithm to obtain a strongly-connected graph.

Double-Link: In one time step, each node u sends a message about itself to each node in its known set.

Algorithm Double-Link takes 1 time step, sends at most $n(n-1)$ messages, and passes at most $n(n-1)$ pointers. On a weakly-connected graph, if Double-Link (instead of Name-Dropper) is executed before *Absorption*, the overall pointer complexity is improved to $O(n^2)$ in expectation. However, the message complexity would degrade to $O(n^2)$ with high probability.

5.2 $O(n)$ Messages on Strongly-Connected Graphs

We describe a variant of the *Absorption* algorithm that reduces the expected message complexity of the *Absorption* algorithm to $O(n)$, at the cost of higher time complexity. We call this variant "*Absorption-M*", for optimizing the message complexity.

We now describe the changes to the original *Absorption* algorithm. First, we remove procedure 4 of stage 1. Instead of notifying all the members, a retiring leader will just forward future seek messages to its new leader. Therefore, during procedure 2, a retired leader u will forward the seek messages received from its members to u's leader. The side effect of this modification is that procedure 2 can no longer be finished in a single time step. In fact, during the ith round of stage 1, procedure 2 needs $i-1$ time steps for the chain of retired leaders to forward the seek messages to the current leaders.

Theorem 3. *The expected time complexity of* Absorption-M *is* $O(\log^2 n)$. *The expected message complexity of* Absorption-M *is* $O(n)$. *The expected pointer complexity of* Absorption-M *is* $O(n^2)$.

6 Concluding Remarks

In this paper, we describe and analyze the *Absorption* algorithm and its variant *Absorption-M*. We have proved stronger asymptotic bounds on the performance measures for *Absorption* and *Absorption-M*, compared to previously known algorithms. The open question is whether there is a resource discovery algorithm that achieves optimal asymptotic bounds on all three complexity measures.

In addition, we present the first exact analysis of expected performance bounds for resource discovery algorithms. We also note that on a strongly-connected graph, the *Absorption* algorithm will terminate when all the machines are found, unlike the Name-Dropper algorithm [1].

The major weakness of the *Absorption* algorithm is its reliance on a few machines (the leaders) to distribute the pointers. However, we note that fault tolerance can be easily obtained by running independent instances of *Absorption* in parallel. In particular, any node in a strongly-connected graph has the opportunity to become the ultimate leader.

Of related interest is the problem of transforming a weakly-connected graph to a strongly-connected graph, since the performance of *Absorption* on weakly-connected graphs is limited by the transformation algorithm used. We are currently developing a transformation algorithm which has more balanced message complexity and pointer complexity.

References

[1] Mor Harchol-Balter, Tom Leighton, and Daniel Lewin. Resource discovery in distributed networks. In *18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, May 1999.

[2] Joseph Jájá. *An Introduction to Parallel Algorithms.* Addison-Wesley, Reading, 1992.

[3] Shay Kutten and David Peleg. Deterministic distributed resource discovery. In *Nineteenth Annual ACM SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, Portland, Oregon, 16-19 July 2000.