The Generalized Deadlock Resolution Problem

Kamal Jain[1], MohammadTaghi Hajiaghayi[2], and Kunal Talwar[1]

[1] Microsoft Research
{kamalj, kunal}@microsoft.com
[2] MIT Computer Science and Artificial Intelligence Laboratory,
hajiagha@csail.mit.edu

Abstract. In this paper we initiate the study of the AND-OR directed feedback vertex set problem from the viewpoint of approximation algorithms. This AND-OR feedback vertex set problem is motivated by a practical deadlock resolution problem that appears in the development of distributed database systems[1]. This problem also turns out be a natural generalization of the directed feedback vertex set problem. Awerbuch and Micali [1] gave a polynomial time algorithm to find a minimal solution for this problem. Unfortunately, a minimal solution can be arbitrarily more expensive than the minimum cost solution. We show that finding the minimum cost solution is as hard as the directed Steiner tree problem (and thus $\Omega(log^2 n)$ hard to approximate). On the positive side, we give algorithms which work well when the number of writers (AND nodes) or the number of readers (OR nodes) are small.

We also consider a variant that we call *permanent deadlock resolution* where we cannot specify an execution order for the surviving processes; they should get completed even if they were scheduled adversarially. When all processes are writers (AND nodes), we give an $O(\log n \log \log n)$ approximation for this problem.

Finally we give an LP-rounding approach and discuss some other natural variants.

1 Introduction

One of the best ways to understand deadlocks in databases is the dining philosophers problem. Say there are five philosophers sitting on a circular table to eat spaghetti, with a fork between every two of them. Each philosopher needs two forks to eat. But everybody grabs the fork on the right, hence everybody has one fork and waiting for another to be freed. This wait will be never ending unless one of the philosophers gave up and freed up his fork. This never ending is an example of a *deadlock*. Picking up a philosopher who can give up on eating the spaghetti is an example of *deadlock resolution*. Now suppose that these philosophers have different likings for the spaghetti and hence different inherent cost of giving up eating it. In this case we want to pick the philosopher who likes spaghetti the least. This is called the *minimum cost deadlock resolution* problem.

In databases, philosophers correspond to independent agents e.g., transactions and processes. Forks correspond to shared resources, e.g., shared memory. Eating spaghetti

[1] Thanks to Ondrej Such from Microsoft for asking an algorithm for this problem.

L. Caires et al. (Eds.): ICALP 2005, LNCS 3580, pp. 853–865, 2005.
© Springer-Verlag Berlin Heidelberg 2005

corresponds to actions which these independent agents want to perform on the shared resources e.g., reading or writing a memory location. So in general besides asking for two forks these philosophers may ask for two spoons too, while they have grabbed only one each. These spoons and forks can be of different kinds (e.g., plastic or metallic). In general demands for resources can be very complicated and it can be represented by a monotonic binary function, called *demand function*. A demand function takes a vector of resources as an input and outputs whether it can satisfy the demand or not.

When a process does not get all the resources to satisfy its demand then it has to wait. Like any other protocol involving waiting, there is a risk of deadlock. There are ways to avoid deadlock, like putting a total order on all the resources and telling to the users to ask them in the same order. In big or distributed databases, such solutions are difficult to implement. Moreover such a solution works when the demand functions consist of only ANDs. In essence deadlocks do happen and they need to be resolved at a small cost. In practice one of the convenient solution is to time out on wait, i.e., if it takes too long for a transaction to acquire further resources then it aborts and frees up the resources held so far. This solution does not have any guarantee on the cost incurred. For notational convenience, aborting a transaction will also be referred as *killing* it. We assume that there is an associated cost of killing a process (this cost can also be the cost of restarting it). The cost of a solution is the total cost of all the processes killed. For the minimum cost deadlock resolution problem we want to kill the least expensive set of processes to resolve the deadlock.

An instance of a generalized deadlock detection problem is captured by a *waits-for-graph* (WFG) on transactions. An old survey by Knapp [17] mentions many relevant models of WFG graphs. In the AND model, formally defined by Chandy and Misra [5], transactions are permitted to request a set of resources. A transaction is blocked until it gets all the resources it has requested. In the OR model, formally defined by Chandy et al. [6], a request for numerous resources are satisfied by granting any requested resource, such as satisfying a read request for a replicated data item by reading any copy of it. In a more generalized AND-OR model, defined by Gray et al. [12] and Herman et al. [15], requests of both kinds are permitted. A node making an AND request is called an AND node and a node making an OR request is called an OR node. An advantage of using both these kinds of nodes is that one can express[2] arbitrary demand functions e.g., if a philosopher wants any one fork and any one spoon then we can create two sub-agents for this philosopher, one responsible for getting a fork and the other for getting a spoon. This philosopher then becomes an AND node and the two sub-agents become two OR nodes. From the perspective of algorithm design, detecting deadlocks in all these models is not a difficult task (see e.g. [11, 21, 23]). The difficult task is to resolve it once detected and that too at a minimum cost (for some heuristics and surveys on the generalized AND-OR model see e.g. [1, 3, 4, 14]). In the next section we formally model the problem as an AND-OR directed feedback vertex set problem.

Often it may not be possible for the deadlock resolving algorithm to specify a schedule for the remaining processes, and when the cost of calling the deadlock resolution algorithm is large (as one would expect in a distributed setting), we would like that

[2] This expression may be of exponential size. See [17] for more models of waits-for-graphs.

no matter in what order the surviving transactions are scheduled, they do not deadlock again. This motivates the *permanent deadlock resolution* problem. For the case when the transactions are all writers (the AND only case), we show a polynomial-time approximation algorithm for the problem.

1.1 Our Results

When all the nodes are OR nodes then the problem can be solved in polynomial time via strongly connected components decomposition. But the problem quickly becomes at least as hard as the set-cover problem even in the presence of a single AND node. Our reduction has deadlock cycles of length 3 capturing the special case mentioned by Jim Gray, who says in practice deadlocks happen because of cycles of length 2 or 3. We give an $O(n_a \log(n_o))$ factor approximation algorithm, where n_o is the number of OR nodes and n_a is the number of AND nodes. On the other hand if all the nodes are AND nodes, the problem is the well-studied directed feedback vertex set problem. There are approximation algorithms with polylog approximation factor for this problem due to Leighton-Rao [18] and Seymour [22]. We generalize those algorithms to destroy all the *handles* at a pivot vertex. We define handles later in the paper and there we also show that destroying handles is a more general problem than destroying cycles. We use this generalization as a subroutine to develop an $O(n_o \log(n_a) \log\log(n_a))$ factor approximation algorithm.

From the hardness point of view, we show that the problem is as hard as the directed Steiner tree problem, which was shown to be hard to approximate better than a factor of $O(log^{2-\epsilon} n)$ by Halperin and Krauthgamer [13], and has no known polynomial time polylogarithmic approximation algorithm. One difficulty in designing an approximation algorithm for our problem is that we do not know any good LP relaxation. The natural LP relaxation itself is at least as hard as the directed Steiner tree problem, even for the case of one OR node. It will be interesting to interpret our algorithms in terms of LP rounding. We do that in case there is one (or a constant number of) OR nodes . The size of this LP is exponential in the number of OR nodes.

For the *permanent deadlock resolution* problem, we show that the case with only AND nodes is reducible to the feedback vertex set problem in mixed graphs. Acyclicity implies schedulability for both undirected and directed graphs - acyclic undirected graphs have leaves and acyclic directed graphs have sinks. Corresponding theorem for mixed graphs is not clear. We develop a corresponding theorem for bipartite mixed graphs. This leads to an $O(\log n \log\log n)$ approximation algorithm for this problem. We leave open the approximability of this problem in the general case.

This problem was also studied in theoretical computer science by Awerbuch and Micali [1]. In their paper, they mentioned that the ideal goal is to kill a set of processes with minimum cost, but the problem is a generalization of feedback vertex set and seems very hard. Thus they gave a distributed algorithm for finding a minimal solution. Unfortunately, a minimal solution can be arbitrarily more expensive than the minimum cost solution. We study this problem from approximation algorithm point of view. We are excited with the fact that the problem has such a rich mathematical structure. It allows use of many results, which were discovered after the paper due to Awerbuch and Micali. In this paper we try to find a proper place for the problem in the vast area of approximation algorithms. We show that this problem blends naturally with feedback vertex and arc set problems. From hardness point of view it blends naturally with the directed Steiner tree and set cover problems. In the discussion section we mention an alternative approach to design approximation algorithms for the directed Steiner tree problem. This approach is suggested by interpreting our algorithm for the case of one OR node in terms of linear programming. This approach does not seem to be based upon the standard LP for the directed Steiner tree problem, which some researchers suspect to have integrality ratio worse than polylogarithmic.

Due to interest of space, we omit several proofs in this extended abstract and defer them to the journal version of this paper.

2 Problem Definition and Preliminary Results

All the graphs in this paper are directed without loops or multiple edges, unless stated otherwise. Our graph terminology is as follows. A graph G is represented by $G = (V, E)$, where V (or $V(G)$) is the set of vertices (or nodes) and E (or $E(G)$) is the set of edges. We denote an edge e from u to v by (u, v), and we call it an *outgoing* edge for u and an *incoming* edge for v. We say node u can *reach* node v (or equivalently v is *reachable* from u) if there is a path from u to v in the graph. We shall use the notation $u \rightsquigarrow v$ to denote that v is reachable from u. We define n to be the number of vertices of a graph when this is clear from context. We denote the maximum out-degree by Δ_{out} and the maximum in-degree by Δ_{in}. We assume that the node set V is partitioned into two sets V_a and V_o. Nodes in V_a and V_o are referred to as AND nodes and OR nodes respectively. We let $n_a = |V_a|$ and $n_o = |V_o|$. With this terminology we now define the wait-for-graphs (WFG).

Each node of a wait-for-graph, $G = (V, E)$, represents a transaction. An edge (u, v) denotes that transaction u has made a request for a resource currently held by transaction v. There are two kinds of nodes. An AND node represents a transaction which has made an AND request on a set of resources, which are held by other transactions. An OR node represents a transaction which has made an OR request on a set of resources. Without loss of generality we assume that a transaction is allowed to make only one request. If a transaction makes multiple requests then we can create a sub-transaction for each request and put the necessary dependency edges. Each transaction has an associated weight. We denote the weight of a transaction u by w_u.

An AND transaction can be scheduled if it gets all the resources it has requested. An OR transaction can be scheduled if it gets at least one of the resources it has requested. Once a transaction is scheduled, it gives up all its locks, potentially allowing other processes to get scheduled. A wait-for-graph is called *deadlock free* if there exist an ordering of the transactions in which they can be executed successfully. If no such ordering exist then we say that the graph has a deadlock. The *minimum cost generalized deadlock resolution problem (GDR)* is to kill the minimum weight set of transactions to free up the resources held by them so that the remaining transactions are deadlock free. In other words, there exists an order on the remaining transactions so that for each AND transaction, each of its children is either killed or can be completed before it and for each OR transaction at least one of its children is either killed or can be completed before it.

2.1 Some Special Cases

We show some simple propositions which give us some intuition about the problem.

Proposition 1. *The GDR problem when there is no OR node has an approximation algorithm with ratio* $O(\log n \log\log n)$.

Proposition 2. *The GDR problem with all OR nodes can be solved in polynomial time.*

In fact, we can strengthen Proposition 2 as follows:

Proposition 3. *The GDR problem, when the reachability graph on the AND nodes is a directed acyclic graph, can be solved in polynomial time.*

Proposition 4. *The GDR problem with uniform weights and* $O(\log n)$ *AND nodes can be solved in polynomial time.*

Using ideas of Propositions 3 and 4, we can show the following theorem.

Theorem 1. *The GDR problem with uniform weights and* n_a *AND nodes has an* $O(n_a)$*-approximation algorithm.*

3 Hardness Results and Natural LP

In this section, we consider the hardness of the GDR problem. First, we show a simple approximation preserving reduction from the set cover problem to this problem. Recall that the set cover problem is to find a minimum collection $\mathcal{C}$ of sets from a family $\mathcal{F} \subseteq 2^U$, such that $\mathcal{C}$ covers U, i.e. $\cup_{S \in \mathcal{C}} S = U$. From the results of Lund and Yannakakis [20] and Feige [10], it follows that no polynomial time algorithm approximates the set cover problem better than a factor of $\ln n$ unless $NP \subseteq \text{DTIME}(n^{\log\log n})$. Our reduction then implies a similar hardness for the GDR problem. There is no similar inapproximability result known for the directed feedback vertex set problem.

Theorem 2. *There exists an approximation preserving reduction from (unweighted) set cover to GDR with only one AND node.*

It is worth mentioning that in the reduction of Theorem 2, there is only one AND node whose weight is $m + 1$ and the rest of the vertices are OR nodes with weight one. Moreover, the one AND node of high weight can be replaced by $m + 1$ AND nodes of unit weight placed "in parallel". Thus the uniform weight case is also hard to approximate better than a factor of $\Omega(\log n)$.

Now the question is that whether it is possible to get a better inapproximability result. To answer this question, we use a recent result of Halperin and Krauthgamer [13] on the inapproximability of the *directed Steiner tree* problem. In the directed Steiner tree problem, given a directed graph $G = (V, E)$, a *root* $r \in V$ and a set of *terminals* $T \in V$, our goal is to find a minimum subset $E' \subseteq E$ such that in graph $G' = (V, E')$ there is a path from r to every $t \in T$. Halperin and Krauthgamer [13] show that the *directed Steiner tree* problem is hard to approximate better than a factor of $\Omega(\log^2 n)$, unless

$NP \subseteq$ ZTIME$(n^{\text{polylog } n})$. So far, no polynomial-time polylogarithmic approximation algorithm is known for this problem. We show a similar non-approximability result in Theorem 3 for GDR by giving an approximation preserving reduction from directed Steiner tree.

Theorem 3. *There exists an approximation preserving reduction from directed Steiner tree to GDR.*

Proof. We consider an instance of directed Steiner tree given by a directed graph $G = (V, E)$, a set of terminals $T \subseteq V$ and a root node $r \in V$. The goal is to find a minimum cost subset E' of edges containing a path from r to every terminal $t \in T$. The reduction is as follows. For each vertex $v \in V - \{r\}$, we create an OR node v of weight ∞[3] in our GDR instance. For r, we create an OR node r of weight zero. In addition, we have an AND node a of weight ∞ which has an edge (a, t) for each $t \in T$ and an edge (v, a) for each $v \in V$. For each edge $e \in E$, we put an AND-OR gadget shown in Figure 1, with the weight of each node as shown in the figure. Recall that a is the global AND node introduced before and o_e and a_e are new OR and AND nodes corresponding to e respectively. Intuitively, using an edge e in the Steiner tree corresponds to killing the OR node o_e in this gadget.

Next we show that the cost of an optimum Steiner tree is equal to the minimum cost of nodes to be killed such that the remaining graph is deadlock-free. First consider a Steiner tree S in G. We kill all OR nodes corresponding to edges in S. For each edge $e = (u, v) \in S$, killing o_e allows v to be complete after u. Thus, first complete node r, then complete nodes according to the directed Steiner tree. Since the Steiner tree solution contains a path to each terminal, we can complete all terminals. Now, after completing all terminals, we can complete the global AND node a and then complete every other node in the graph.

On the other hand, since the only nodes with finite weight are the OR nodes corresponding to edges and the node corresponding to root r, any feasible solution of finite weight for GDR kills only such nodes. It is easy to check that the set of edges for which the OR nodes are killed contain a directed Steiner tree. □

Again, we might replace each node of weight ∞ with several nodes of unit weight, say $|E(G)|$, in order to reduce the directed Steiner tree problem to the uniform weighted case.

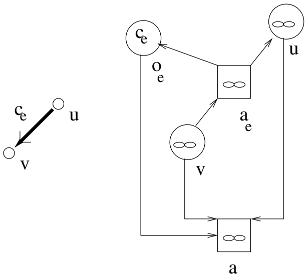

Fig. 1. Edge $e = (u, v)$ in graph G and its AND-OR gadget in the new instance of GDR

3.1 Natural LP and Hardness

We end this section by considering a natural LP for the GDR problem, which is a generalization of the LP for feedback vertex set (see e.g. [9]). We say a set of nodes H forms a *Minimal Deadlocked Structure (MDS)* if

[3] As usual, the ∞ weights can be replaced by a (polynomially) large weight.

1. For any OR node $u \in H$, all its outneighbors are in H.
2. For any AND node $u \in H$, at least one of its outneighbors is in H.
3. H is minimal[4] amongst sets satisfying (1) and (2).

We now write a linear program (called *LP 1*) is as follows: minimize $\sum_{v \in V} w_v x_v$ such that $\sum_{v \in H} x_v \geq 1$ for any *MDS* H and $x_v \geq 0$ for all $v \in V$

Clearly an integral solution to this linear program is a feasible solution to the underlying GDR instance and hence this is a relaxation. However, this linear program can potentially have exponentially many constraints. Note that if the graph G does not have any OR node, MDSs are exactly the minimal directed cycles and our LP is the same as the LP considered in [18,22,9] for applying region growing techniques for the feedback vertex set problem. In this special case of feedback vertex set, this LP has a simple separation oracle which enables us to solve it using Ellipsoid method. However, we now show that even the separation oracle for LP 1 is as hard as the directed Steiner tree problem.

Theorem 4. *The separation oracle for LP 1 is as hard as solving directed Steiner tree.*

Proof. A separation oracle for LP 1 solves the following problem: given a vector $\overrightarrow{x}$, is there an MDS H for which $\sum_{v \in H} x_v < 1$. We shall reduce the directed Steiner tree problem to this problem

We consider an instance of directed Steiner tree: given a root r and a set of terminals T in a directed graph $G = (V, E)$, is there is Steiner tree of weight at most 1 (by scaling). Without loss of generality we assume G is a directed acyclic graph (DAG), since the directed Steiner tree problem on DAGs is as hard as the one on general directed graphs (see e.g. [7]). Also without loss of generality assume we have weights on vertices instead of edges (again the two problems are equivalent). Now we are ready to demonstrate the reduction. For each vertex $v \in V$, we place an AND node v with x_v equal to its weight in the Steiner instance. For each edge (u, v) in G, we place an edge (v, u) in our new graph. In addition, we add an OR node with $x_o = 0$ which has an outgoing edge (o, t) for each terminal $t \in T$ and an incoming edge (r, o) (r is the root node). Call the new graph G'. It is easy to check that $H \cup \{o\}$ is an MDS in G' if and only if H is a directed Steiner tree in G. Hence the claim follows. □

As shown by Jain et. al. [16], for these kinds of problems optimizing LP 1 is equivalent to solving the separation oracle problem. Furthermore, these reductions are approximation preserving. Thus if we can optimize LP 1 within some factor then we can solve its separation oracle for the same factor. Hence by Theorem 4, we can solve the directed Steiner tree problem within the same factor.

Corollary 1. *Optimizing LP 1 is at least as hard as the directed Steiner tree problem.*

Finally, notice that finding the integrality gap of LP 1 is an interesting open problem.

[4] With respect to set inclusion.

4 Approximation Algorithms

In this section, we give an $O(\min\{n_a \log n, n_o \log n \log\log n\})$ algorithm for this problem, where n_a is the number of AND nodes and n_o is the number of OR nodes in the instance. Thus, when either of n_a or n_o is small, the problem is well approximable.

In subsection 4.1 we show how to use region growing to solve a slight generalization of feedback vertex set. We use this to get an $O(n_o \log n \log\log n)$ algorithm in subsection 4.2. In subsection 4.3, we give an $O(n_a \log n)$ approximation algorithm for the problem. Thus the better of these two algorithms gives the claimed performance guarantee.

4.1 Handle Removal Algorithm

In this section, we consider the following handle removal problem which plays an important role in the algorithm for the case of few OR nodes (see Section 4.2): Given a directed graph G, and a designated vertex r, delete the smallest number (weight) of vertices such that the remaining graph has no cycles reachable from r. For ease of exposition, we shall replace each vertex by a pair of vertices joined by an edge, and transfer the weight to this edge. The edges in the original graph are given an infinite weight. The problem then reduces to finding the smallest cost set of edges whose removal eliminates all cycles reachable from r.

We shall write this problem as an integer program, and consider its linear programming relaxation. We first formally define a *handle*.

Definition 1. *Let $H = (r = u_0, u_1, \ldots, u_k)$ be a simple path in G. We call H a* handle *if for some $p : 0 \leq p < k$, there is an edge (u_k, u_p). We refer to u_p as the* pivot *of the handle H. The edges on the path along with the edge $\{u_k, u_p\}$ constitute the edges of the handle.*

Let $\mathcal{H}$ be the set of all handles in G. We can write the following linear programming relaxation for this problem: minimize $\sum_{e \in E} w_e x_e$ such that $\sum_{e \in H} x_e \geq 1$ for all $H \in \mathcal{H}$ and $x_e \geq 0$ for all $e \in E$.

Note that the above linear program has an exponential number of constraints. The separation oracle for this LP requires us to find a violated handle in a given fractional solution. Note that we can find in polynomial time, for each $u \in V$, the smallest cycle passing through u, and the shortest path from r to u. The shortest handle in the graph is then just the minimum, over all u, of the sum of the above two quantities. Thus the LP has a polynomial time separation oracle, and hence can be solved by Ellipsoid method.

Given a solution to this linear program, we shall now argue that the techniques used by Seymour [22] and Even et.al. [9] for the feedback arc set problem apply here to give an $O(\log n \log\log n)$-approximation algorithm to the problem. Given a graph G, and a non negative length function x_e on the edges, we can define the shortest path function d_x on the vertices of G. A modification of the algorithm of Seymour implies the following theorem:

Theorem 5. *Given a weighted graph G, a special vertex r and non-negative length function x_e, let $W = \sum_e w_e x_e$. There exists a set of edges C such that:*

- $\sum_{e \in C} w_e \leq O(\log n \log\log n) \cdot W$
- *For any vertex v with $d_x(r, v) \geq \frac{1}{4}$, C is an r-v cut.*
- *For any pair of vertices u and v such that $d_x(u, v) \geq \frac{1}{4}$, C contains either a u-v cut or a v-u cut.*

We now argue that the rounding described in the theorem applied to a feasible LP solution, gives a feasible solution to the handle removal problem.

Claim. Let x be a feasible solution to the linear program above and d be the shortest path function defined accordingly. Then for any handle H with pivot u, *either* $d(r, u) \geq \frac{1}{4}$, *Or* there exists $v \in H$ such that $d(u, v) \geq \frac{1}{4}$.

From the above claim, and theorem 5, it follows that the handle removal problem is approximable within a factor of $O(\log n \log\log n)$.

4.2 Few OR Nodes Algorithm

Using the algorithm for the handle removal problem in section 4.1, we are now ready to prove the following theorem.

Theorem 6. *There is an $O(n_o \log(n_a) \log\log(n_a))$-approximation algorithm for the Generalized Deadlock Resolution Problem.*

Proof. We first assume that the OR nodes in the graph have infinite cost, and thus are all scheduled. We shall give an $O(\alpha n_o)$ solution where α is the approximability of the handle removal problem.

Let u be the first OR node to be scheduled. Since this node is scheduled, one of its outneighbors, say v, is killed/scheduled before any other OR node. Since no cycle of AND nodes can be scheduled, no such cycle reachable from v survives in the optimum solution. OPT thus includes a solution to the handle removal problem with root v. Let O_v be the optimum of the handle removal problem with root v, when all OR nodes are removed. Hence $OPT \geq \min_{u \in V_O} \min_{v:(u,v) \in E} O_v$.

Our algorithm is as follows. Using an α-approximation algorithm for the handle removal problem, we compute solutions to handle removal problems rooted at $\{v : (u, v) \in E, u \in V_O\}$. We pick the cheapest of these and kill the nodes in this solution. The cost of killing these nodes is at most αOPT. Now the OR node u can be scheduled, and consequently some more nodes can be scheduled. We remove all such nodes along with their incoming edges, and recur. In the base case, when there are no OR nodes, we have the feedback vertex set problem, which is also approximable within α (by a simple reduction to the handle removal problem). Thus we get obtain $(n_o+1)\alpha$ approximation.

We now show how to remove the assumption about OR node removal. To each OR node u in the graph, we add a new outneighbor u_a which is an AND node with cost equal to the original OR node. We add another AND node v of infinite cost to the graph, with edges to all the original vertices of the graph. Finally, we add an edge from u_a to the vertex v, and increase the costs of all OR nodes to infinity.

For any solution to the original instance that kills an OR node u, we can get a solution to the new instance by killing u_a. This lets us schedule u instead of killing u. Moreover, after scheduling/killing all original nodes, the node v can be scheduled, after which any unkilled u_a can be scheduled. Finally, the cost of the new solution is the same as the original one.

A solution to the new instance immediately gives a solution to the original instance: kill OR node u whenever the new solution killed u_a. It is easy to see that this transformation preserves feasibility and cost. □

4.3 Few AND Nodes Algorithm

In this section, we present an $O(n_a \log n)$-approximation algorithm for this problem. We note that in the reduction of set cover to generalized deadlock resolution (mentioned in Theorem 2), we have only one AND node and thus our result is tight in this case. However, in the reduction of directed Steiner tree to this problem, the number of AND nodes is linear and the best non-approximability result is in $\Omega(\log^2 n)$.

The algorithm is as follows. We start with the original graph G and in each iteration we update it. More precisely, if in an iteration graph G does not have any AND node, we can obtain the optimal solution for G by the procedure mentioned in Proposition 2 (and thus we stop). Otherwise, for each AND node a whose outgoing edges are $(a, c_1), (a, c_2), \cdots, (a, c_{\Delta_{out}})$ in graph G and all c_i's, $1 \leq i \leq \Delta_{out}$, are OR nodes, we construct the following hitting set instance (note that the hitting set problem is the dual of the set cover problem). For each c_i, $1 \leq i \leq \Delta_{out}$, we form a set S_i which contains all OR nodes reachable via OR nodes from c_i (i.e. paths from c_i to S_i do not use any AND nodes). Now, the collection $\mathcal{C}$ contains all sets $S_i \subseteq S$, where S is the set of all OR nodes. Using the $(1+\ln \Delta_{out}) = O(\log n)$ approximation for hitting set, we obtain a set S_a^* of weight w_a^* of OR nodes which hit every set. Let $W_a = \min\{w_a, w_a^*\}$ (w_a is the weight of node a). Choose the AND node a with minimum W_a over all AND nodes. Kill AND node a or all the OR nodes in the corresponding hitting set solution (the one with minimum weight). Clear graph G, i.e., remove every AND/OR node which can be completed after killing the aforementioned nodes, and repeat the above iteration for G. The final solution contain all AND/OR nodes killed during the iterations.

We finish by showing that

Theorem 7. *The above algorithm kills a set of AND/OR nodes such that the remaining graph is deadlock free and the weight of the solution is at most* $(1 + \ln \Delta_{out})n_a + 1 = O(n_a \log n)$ *times optimum.*

Proof. The correctness of the solution can be easily seen from the description of the algorithm. Thus, we only show the approximation factor here. To this end, we prove that in each iteration, except the case in which there is no AND node, we kill nodes of total weight at most $(1 + \ln \Delta_{out})$ times optimum weight for the updated graph G in that iteration. In the last iteration, we kill nodes of total weight at most OPT according to the description of the algorithm. Using these facts and that OPT in each iteration is at most the original optimum, we obtain the desired approximation factor.

Consider an optimum solution and let a be the first AND node which is completed or killed in the optimum resolution. Thus either we have killed a or we have completed a by killing at least one OR node from the OR nodes reachable from each of its children. Hence for at least one AND node, the weight of the solution to the corresponding hitting set instance is at most the weight of optimum. Since the approximation factor of hitting set is $1+\ln \Delta_{out}$ and we try all AND nodes and then take the minimum, the total weight of the killed nodes is at most $(1 + \ln \Delta_{out})$ times optimum, as desired. □

5 Permanent Deadlock Resolution

Here we consider another version of the deadlock resolution problem where it is not possible for the algorithm to specify a feasible schedule on the remaining processes. In particular, we want to kill enough processes, such that if the remaining processes try to acquire locks in any order, they cannot deadlock. We then say that the remaining processes are *adversarially schedulable*.

We consider the special case of this problem when all processes are writers (AND nodes). In this case, we show that this problem can be reduced to the feedback vertex set problem on mixed graphs (i.e. graphs with both directed and undirected edges). Since this problem yields to the same techniques as those used for feedback vertex set of directed graphs, we get an $O(\log n \log\log n)$-approximation.

We are given a set of resources R and a set of processes P, each holding a lock on some subset of resources, and waiting to get locks on another subset of resources. We construct a bipartite mixed graph as follows: create a vertex v_r for every resource r with infinite cost, and a vertex v_p for every process p. Whenever process p holds the lock on resource r, we add a directed edge from v_p to v_r. Moreover, we add an undirected edge between v_p and $v_{r'}$ whenever process p is waiting to get a lock on resource r'.

Theorem 8. *An instance is adversarially schedulable if and only if the corresponding graph is acyclic.*

Proof. We first argue that greedily schedulability implies acyclicity. Assume the contrary, and let the graph have a cycle $p_1, r_1, p_2, r_2, \ldots, p_k, r_k, p_1$. Now consider the schedule in which p_i grabs a lock on r_i (or already holds it, in case the edge is directed). Note that p_i waits for a lock on r_{i-1} and p_1 waits on r_k. this entails a cyclic dependency amongst processes $p_1, \ldots, p_k$: p_i cannot finish unless p_{i-1} finishes and releases r_{i-1}. This configuration is therefore deadlocked. Since we have shown how to reach a deadlocked state from the initial state, the initial state was not adversarially schedulable, which contradicts the assumption. Hence the claim follows.

Now suppose that the graph is acyclic. We claim that the initial configuration is adversarially schedulable. Suppose not. Then there is a sequence of lock acquisition that lead to a deadlocked configuration. Clearly, a deadlocked configuration corresponds to processes $p_1, p_2, \ldots, p_k$ such that p_{i+1} is waiting for p_i to release some resource r_i. Since p_i holds r_i in this configuration, (p_i, r_i) must be directed/undirected edge in the graph. Moreover, since p_{i+1} is waiting for r_i, (r_i, p_{i+1}) is an undirected edge in the graph. However, we have just shown that $p_1, r_1, p_2, r_2, \ldots, p_k, r_k, p_1$ is a cycle in G, which contradicts the acyclicity of G. Thus the claim follows. □

Theorem 9. *The permanent deadlock resolution problem for AND nodes has an* $O(\log n \log\log n)$ *approximation algorithm.*

Acknowledgement

We would like to thank, Ondrej Such, a developer at Microsoft for asking his question on the Algorithm's email list. The first author would also like to thanks Joseph Cheriyan and Laci Lovasz for many initial and fruitful discussions. He would also like to thank David Shmoys for a very short but helpful discussion. The last author would like to thank Christos Papadimitriou and Joe Hellerstein for very useful discussions.

References

1. B. Awerbuch and S. Micali, *Dynamic deadlock resolution protocols*, in The 27th Annual Symposium on Foundations of Computer Science, 1986, 196–207.
2. R. Bar-Yehuda, D. Geiger, J. Naor, and R. M. Roth, *Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference*, SIAM J. Comput., 27 (1998), 942–959.
3. G. Bracha and S. Toueg, *A distributed algorithm for generalized deadlock detection*, in Proceedings of the 3rd annual ACM symposium on Principles of distributed computing, ACM Press, 1984, 285–301.
4. K. M. Chandy and L. Lamport, *Distributed snapshots: determining global states of distributed systems*, ACM Transactions on Computer Systems (TOCS), 3 (1985), 63–75.
5. K. M. Chandy and J. Misra, *A distributed algorithm for detecting resource deadlocks in distributed systems*, in Proceedings of the 1st ACM SIGACT-SIGOPS symposium on Principles of distributed computing, ACM Press, 1982, 157–164.
6. K. M. Chandy, J. Misra, and L. M. Haas, *Distributed deadlock detection*, ACM Transactions on Computer Systems (TOCS), 1 (1983), 144–156.
7. M. Charikar, C. Chekuri, T.-y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li, *Approximation algorithms for directed Steiner problems*, J. Algorithms, 33 (1999), 73–91.
8. J. Cheriyan, H. J. Karloff, and Y. Rabani, *Approximating directed multicuts*, in The 42th Annual Symposium on Foundations of Computer Science, 2001, 348–356.
9. G. Even, J. Naor, B. Schieber, and M. Sudan, *Approximating minimum feedback sets and multicuts in directed graphs*, Algorithmica, 20 (1998), 151–174.
10. U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, J. ACM, 45 (1998), 634–652.
11. M. Flatebo and A. K. Datta, *Self-stabilizing deadlock detection algorithms*, in Proceedings of the '92 ACM annual conference on Communications, ACM Press, 1992, 117–122.
12. J. Gray, P. Homan, R. Obermarck, and H. Korth, *A straw man analysis of probability of waiting and deadlock*, in Proceedings of the 5th Internafional Conference on Distributed Data Management and Computer Networks, 1981.
13. E. Halperin and R. Krauthgamer, *Polylogarithmic inapproximability*, in The 35th Annual ACM Symposium on Theory of Computing (STOC'03), 2003, 585–594.
14. J.-M. Helary, C. Jard, N. Plouzeau, and M. Raynal, *Detection of stable properties in distributed applications*, in Proceedings of the 6th PODC, ACM Press, 1987, 125–136.
15. T. Herman and K. M. Chandy, *A distributed procedure to detect and/or deadlock*, Tech. Rep. TR LCS-8301, Dept. of Computer Sciences, Univ. of Texas, 1983.
16. K. Jain, M. Mahdian, and M. R. Salavatipour, *Packing Steiner trees*, in The 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03), 2003, 266–274.
17. E. Knapp, *Deadlock detection in distributed databases*, ACM Computing Surveys (CSUR), 19 (1987), 303–328.

18. T. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), 787–832.
19. R. J. Lipton and R. E. Tarjan, *Applications of a planar separator theorem*, SIAM J. Comput., 9 (1980), 615–627.
20. C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. Assoc. Comput. Mach., 41 (1994), 960–981.
21. K. Makki and N. Pissinou, *Detection and resolution of deadlocks in distributed database systems*, in Proceedings of the 4th international conference on Information and knowledge management, ACM Press, 1995, 411–416.
22. P. D. Seymour, *Packing directed circuits fractionally*, Combinatorica, 15 (1995), 281–288.
23. H. Wu, W.-N. Chin, and J. Jaffar, *An efficient distributed deadlock avoidance algorithm for the and model*, IEEE Transactions on Software Engineering, 28 (2002), 18–29.