Execution Ordering in AND/OR Graphs with Failure Probabilities

Priyankar Ghosh and **P. P. Chakrabarti** and **Pallab Dasgupta**
Indian Institute of Technology, Kharagpur-721302, India

Abstract

In this paper we consider finding solutions for problems represented using AND/OR graphs, which contain tasks that can fail when executed. In our setting each node represent an atomic task which is associated with a failure probability and a rollback penalty. This paper reports the following contributions - (a) an algorithm for finding the optimal ordering of the atomic tasks in a given solution graph which minimizes the expected penalty, (b) an algorithm for finding the optimal ordering in the presence of user defined ordering constraints, and (c) a counter example showing the lack of optimal substructure property for the problem of finding the solution graph having minimum expected penalty, and a pseudo-polynomial algorithm for finding the solution graph with minimum expected penalty.

Introduction

Traditionally AND/OR structures are being used for problem reduction search (Pearl 1984; Nilsson 1980; Martelli and Montanari 1973; Chang and Slagle 1971), where the root node represents the problem to be solved and the leaf nodes represent atomic subproblems. In this paper we investigate a variant of the AND/OR graph evaluation problem, where the atomic tasks associated with the nodes have a known probability of failure. This variant has notable practical significance since in many problem domains which exhibit AND/OR structure, probability of failure needs to be factored into the process of choosing the best solution strategy. Consider the following examples.

1. In majority of industrial project planning problems, ranging from civil construction to software development, the project goal is typically factored into a hierarchy of subtasks, and for each subtask multiple options (possibly using different types of components/strategy) exist for solving the subtask (Ren, Bai, and Guo 2010). Failure probabilities (Casali 2010; González et al. 2010; Demeulemeester et al. 2007; Pich, Loch, and Meyer 2002; Nozik, Turnquist, and List 2001; Mori and Tseng 1997) for the atomic tasks may be attributed to a variety of factors ranging from reliability of the vendors to natural calamities.

2. In the context of service composition in semantic web services, transactional model of web services (Gaaloul, Bhiri, and Rouached 2010; Liu et al. 2010) are emerging as a result of several factors like – (a) the inherent unreliability of the communication medium (network failure), (b) dynamic nature of the underlying business model (like purchasing a ticket may fail due to unavailability), etc. The need for ensuring global atomicity of the overall composed service is also greatly emphasized (Liu et al. 2008; Liu and Li 2007; Montagut and Molva 2006) and the probability of ensuring the global atomicity is largely dependent on the probability of success of the individual services. Thus considering the failure probabilities of individual services is becoming imperative.

In the traditional AND/OR graph the optimal solution graph is unordered, that is, the sequence in which the atomic subtasks are executed within the solution graph does not affect the cost of the solution graph. In our problem, this is not the case. If any task in a given solution graph fails, then that solution graph cannot provide a solution to the problem any more. We associate a penalty for rolling back the completed tasks when some node in the solution graph fails. Therefore, in our problem, each atomic task is associated with a probability of failure and a cost which is incurred only when the task is solved but later rolled back due to the failure of other subsequent tasks in the solution graph. We have addressed the following problems.

1. Given a solution graph, we study the problem of finding the sequence of executing the atomic tasks in that solution graph which minimizes the expected penalty.
2. Given a AND/OR graph with failure probabilities and rollback penalties for the atomic tasks, we study the problem of finding the solution graph for which the minimum expected penalty is the least.

The overall structure of the paper is as follows. In the next section we present the necessary definition and formalisms. We describe an algorithm for finding the optimal sequence of executing the tasks in a solution graph which minimizes the expected penalty in the succeeding section. The extensions for handling user-defined ordering constraints among the tasks is discussed in the next section. Next we present a counter example to show that the problem of finding the optimum solution graph (that is, the one having minimum expected penalty) does not enjoy the optimal substructure property, thereby ruling out straight forward dynamic programming solutions to the problem. We present an algo-

rithm, named CR-REV* that uses a user defined parameter k, for finding the optimum solution graph. We show empirically that in practice the optimal solution is found with small value of k.

Definitions

In this section we present the definitions and necessary formalisms required for presenting our approach. We develop the formalisms around the notion of the *atomic tasks* which may be associated with the nodes in the AND/OR graph. Let $\mathcal{Q}$ denote a set of *atomic tasks*. With each task, $t_i \in \mathcal{Q}$ we associate the following two attributes:

a. $s(t_i)$: the probability of completing task t_i successfully
b. $c(t_i)$: the cost for committing task t_i, given that it has completed successfully.

Also, the failure probability of a task is denoted as $\ell(t_i) = 1 - s(t_i)$. It may be noted that no cost is incurred for an atomic task which failed to complete. Henceforth, we will use the term *task* and *atomic task* interchangeably.

Definition 1 ***[Evaluation Sequence]*** *An evaluation sequence for a set of tasks, $\mathcal{Q}$, is a permutation, π, of the tasks in $\mathcal{Q}$. We use the notation $\pi[i]$ to denote the i^{th} task in the sequence, π.* □

The semantics of task execution is as follows. Given a set of tasks, $\mathcal{Q}$, and a evaluation sequence, π, the tasks are executed, one at a time, following the given evaluation sequence. If the task, $\pi[i]$, completes successfully, then it is committed with cost $c(\pi[i])$. The execution terminates with success only if all tasks complete successfully. If any task fails to complete, then execution is aborted and none of the subsequent tasks in the evaluation sequence are attempted. We are interested in finding the evaluation sequence which minimizes the expected cost incurred when the execution aborts. Therefore we adopt the following definition of *penalty*.

Definition 2 ***[Expected Penalty]*** *Consider the evaluation sequence, π, for tasks in $\mathcal{Q}$ where $|\mathcal{Q}| = n$. The* expected penalty $\mathcal{P}(\pi)$ *is defined as follows:*
$\mathcal{P}(\pi) = \sum_{k=1}^{n} U_k(\pi)$, *where* $U_1(\pi) = 0$, *and*
$\forall k > 1,\ U_k(\pi) = \big[\prod_{i=1}^{k-1} s(\pi[i])\big].\ell(\pi[k]).\big[\sum_{i=1}^{k-1} c(\pi[i])\big]$ □

In the above definition, the term, $U_k(\pi)$, represents the weighted value of the penalty corresponding to the scenario, where the tasks, $\pi[0], \ldots, \pi[k-1]$, are completed successfully and subsequently the task $\pi[k]$ fails to complete. The scenario where every task is completed successfully does not contribute anything to the computation of the expected penalty. The evaluation sequence having the minimum expected penalty will be denoted by π_{min}, and will be called the *optimal/best evaluation sequence*.

Recursive problem decomposition techniques (such as dynamic programming) decompose the given problem into combinations of atomic tasks, where each combination represents a set of tasks which suffice to solve the problem. AND/OR graphs are used to capture such alternative decompositions in a succinct way (Chang and Slagle 1971).

Also in the context of service composition, a typical use of the AND/OR graph is to model the input/output dependencies (Shin, Jeon, and Lee 2010; Gu, Xu, and Li 2010; Gu, Li, and Xu 2008) among the services and such graphs are also known as *service dependency graphs (SDGs)*. Typically these SDGs are used to describe all possible input-output dependencies among the available web services. In an SDG two types of nodes are used – (*a*) service operation nodes, and (*b*) data nodes corresponding to the inputs and outputs. In order to determine the compositions, AND/OR graphs are constructed from the SDGs. In this AND/OR graph representation the services are modeled as AND nodes and the data nodes (inputs and the outputs of services) are modeled as OR nodes. For every AND node the children are the input nodes and the parents are the output nodes. The data nodes whose values are known to the service requester are marked as solved, and a dummy AND node is connected to the OR nodes corresponding to the requested data nodes. This dummy AND node forms the root node of the AND/OR graph which is searched for finding the compositions.

In our model of AND/OR graph we associate a task with every node. Therefore for the hierarchical decompositions, where the actual task is typically associated with the leaf nodes only, the tasks corresponding to the intermediate nodes will have *zero* failure probability and an arbitrary positive value ε the rollback penalty. Whereas, for SDGs the tasks corresponding to all other nodes that do not represent a service will have *zero* failure probability and ε rollback penalty. We develop our algorithms on the generic model discussed above so that the same model can be used for both type of AND/OR structures.

We use $G = \langle V, E \rangle$ to denote an AND/OR graph, where V is the set of nodes, and E is the set of edges. $v_R \in V$ is the root node of G. The nodes of G with no children are called *terminal* nodes. Each terminal node represents an atomic task. The *non-terminal* nodes of G are of two types, namely OR-nodes and AND-nodes. Edges emanating out of *OR-nodes* are called *OR-edges*, and edges emanating out of *AND-nodes* are called *AND-edges*.

Definition 3 ***[Solution Graph]*** *A* solution graph, $S(v_q)$, *rooted at any node $v_q \in V$, is a finite subgraph of G defined as:*

1. *v_q is in $S(v_q)$;*
2. *If v'_q is an OR node in G and v'_q is in $S(v_q)$, then exactly one of its children in G is in $S(v_q)$;*
3. *If v'_q is an AND node in G and v'_q is in $S(v_q)$, then all of its children in G are in $S(v_q)$;*
4. *No node other than v_q or its successors in G is in $S(v_q)$.*

By a solution graph S of G we mean a solution graph with root, v_R, of G. The set, $\mathcal{Q}(S)$, denotes the set of tasks represented by the terminal nodes of S. □

Typically an AND/OR graph may have multiple solution graphs. The set of terminal nodes of each solution graph represents an alternative combination of tasks that suffice to solve the problem represented by the root of the AND/OR graph. Figure 1 shows an example of an AND/OR tree. This tree represents a hierarchical problem decomposition. In the rest of the paper also we will only use examples of AND/OR trees representing hierarchical problem decomposition for brevity. This AND/OR tree has four solution trees, namely: $S_1 = \{n_8, n_9, n_{12}, n_{13}\}$, $S_2 = \{n_8, n_9, n_{14}, n_{15}\}$,

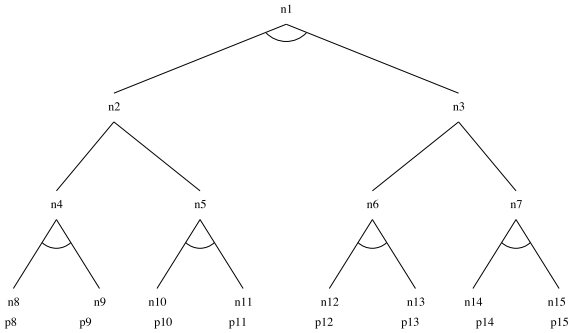

Figure 1: Example of an AND/OR Tree

$S_3 = \{n_{10}, n_{11}, n_{12}, n_{13}\}$ and $S_4 = \{n_{10}, n_{11}, n_{14}, n_{15}\}$.

Given a solution graph, S, we can order its terminal nodes in various ways. Each permutation of the terminal nodes represents an evaluation sequence of the tasks represented by those nodes. In Figure 1, the values of $s(t_i)$ and $c(t_i)$ are shown within angular brackets below each terminal node, t_i. Consider the solution tree, S_1 and the evaluation sequence, $\pi = \langle n_8, n_9, n_{12}, n_{13}\rangle$. Following Definition 2, the expected penalty for π on the solution tree, S_1, is 13.23. As it turns out, the best evaluation sequence (based on expected penalty) for this solution tree is $\pi_{min} = \langle n_9, n_{13}, n_{12}, n_8\rangle$ having expected penalty of 9.325.

Finding the Best Evaluation Sequence

Given a set of tasks, our first objective is to determine the evaluation sequence, π_{min}, having the minimum expected penalty. Given any evaluation sequence, π, suppose we exchange the tasks, $\pi[k]$ and $\pi[k+1]$, to obtain the sequence π'. We define the following function which returns the value of $P(\pi) - P(\pi')$, that is, the difference between the expected penalties of π and π'.

Definition 4 ***[Swap Function on Pair of Tasks]*** *Given an evaluation sequence, π, and two consecutive tasks, $\pi[k]$ and $\pi[k+1]$, we define a* swap function *as follows:*
$f(\pi[k], \pi[k+1]) = U_k(\pi) + U_{k+1}(\pi) - U_k(\pi') - U_{k+1}(\pi')$,
where $\pi[k] = \pi'[k+1], \pi[k+1] = \pi'[k]$,
$\forall j \in \mathbb{N}_1^n,\ (j \neq k \wedge j \neq k+1) \Rightarrow \pi[j] = \pi'[j]$, *and*
U_k is defined earlier in Definition 2 □

In this paper we use $\mathbb{N}_1^n$ to denote the set $\{1, \cdots, n\}$. Now,
$f(\pi[k], \pi[k+1]) = \big[\prod_{i=1}^{k-1} s(\pi[i])\big].\ell(\pi[k]).\big[\sum_{i=1}^{k-1} c(\pi[i])\big]$
$\quad + \big[\prod_{i=1}^{k} s(\pi[i])\big].\ell(\pi[k+1]).\big[\sum_{i=1}^{k} c(\pi[i])\big]$
$\quad - \big[\prod_{i=1}^{k-1} s(\pi'[i])\big].\ell(\pi'[k]).\big[\sum_{i=1}^{k-1} c(\pi'[i])\big]$
$\quad - \big[\prod_{i=1}^{k} s(\pi'[i])\big].\ell(\pi'[k+1]).\big[\sum_{i=1}^{k} c(\pi'[i])\big]$
$\because\ \forall j \in \mathbb{N}_1^n,\ (j \neq k \wedge j \neq k+1) \Rightarrow \big(\pi[j] = \pi'[j]\big)$
$\therefore \prod_{i=1}^{k-1} s(\pi[i]) = \prod_{i=1}^{k-1} s(\pi'[i]) = A$(suppose), and
$\sum_{i=1}^{k-1} c(\pi[i]) = \sum_{i=1}^{k-1} c(\pi'[i]) = B$(suppose). Also $A, B > 0$.

$\therefore f(\pi[k], \pi[k+1])$
$= A.B.\ell(\pi[k]) + A.s(\pi[k]).\big[B + c(\pi[k])\big].\ell(\pi[k+1])$
$\quad - A.B.\ell(\pi'[k]) - A.s(\pi'[k]).\big[B + c(\pi'[k])\big].\ell(\pi'[k+1])$
$= A.B.\ell(\pi[k]) + A.s(\pi[k]).B.\ell(\pi[k+1])$
$\quad + A.s(\pi[k]).c(\pi[k]).\ell(\pi[k+1]) - A.B.\ell(\pi'[k])$
$\quad - A.s(\pi'[k]).B.\ell(\pi'[k+1]) - A.s(\pi'[k]).c(\pi'[k]).\ell(\pi'[k+1])$
$= A.B.\ell(\pi[k]) - A.s(\pi[k+1]).B.\ell(\pi[k])$
$\quad - A.B.\ell(\pi[k+1]) + A.s(\pi[k]).B.\ell(\pi[k+1])$
$\quad + A.s(\pi[k]).c(\pi[k])\ell(\pi[k+1])$
$\quad - A.s(\pi[k+1]).c(\pi[k+1])\ell(\pi[k])$
$\qquad \big|\ \because\ \pi[k] = \pi'[k+1]$ and $\pi[k+1] = \pi'[k]$
$= A.\big[s(\pi[k]).c(\pi[k]).\ell(\pi[k+1])$
$\quad - s(\pi[k+1]).c(\pi[k+1]).\ell(\pi[k])\big]$

Definition 5 ***[Reduced Form of Swap Function]*** *The reduced form of swap function, g, on a pair of tasks, t_i and t_j, is defined as follows:*
$g(t_i, t_j) = s(t_i).c(t_i).\ell(t_j) - s(t_j).c(t_j).\ell(t_i)$ □

Therefore, $f(\pi[k], \pi[k+1]) = A.g(\pi[k], \pi[k+1])$, where $A = \prod_{i=1}^{k-1} s(\pi[i])$. This shows that $f(\pi[k], \pi[k+1])$ can be factored into A, which involves only tasks preceding $\pi[k]$, and $g(\pi[k], \pi[k+1])$, which involves only $\pi[k]$ and $\pi[k+1]$.

We present a few lemmas about function g and function f. Some of the proofs are skipped for brevity; details are available in (Ghosh, Chakrabarti, and Dasgupta 2011).

Lemma 1 *Consider the evaluation sequence π of a set of tasks $\mathcal{Q}$ and $|\mathcal{Q}| = n$. We claim that*
a. $\big[f(\pi[k], \pi[k+1]) > 0\big] \Leftrightarrow \big[g(\pi[k], \pi[k+1]) > 0\big]$
b. $\big[f(\pi[k], \pi[k+1]) < 0\big] \Leftrightarrow \big[g(\pi[k], \pi[k+1]) < 0\big]$
c. $\big[f(\pi[k], \pi[k+1]) = 0\big] \Leftrightarrow \big[g(\pi[k], \pi[k+1]) = 0\big]$

Lemma 2 *For tasks t_1, t_2 and t_3, we claim that,*
a. $\big[g(t_1, t_2) > 0\big] \wedge \big[g(t_2, t_3) > 0\big] \Rightarrow \big[g(t_1, t_3) > 0\big]$
b. $\big[g(t_1, t_2) = 0\big] \wedge \big[g(t_1, t_3) < 0\big] \Rightarrow \big[g(t_2, t_3) < 0\big]$

If a given evaluation sequence π contains a pair of consecutive tasks, $\pi[k]$ and $\pi[k+1]$, such that $g(t(v_i), t(v_j)) > 0$, then we can reduce the expected penalty by swapping $\pi[k]$ with $\pi[k+1]$. Therefore evaluation sequences of minimal costs will never have such pairs. Since the value of $g(t_i, t_j)$ depends only on t_i and t_j (and not on the tasks which precede and succeed them), we can define the following *evaluation graph* and claim that every evaluation sequence of minimal expected penalty is a topological ordering of this graph. We will never construct this graph in practice – we use this representation only to prove the correctness of our approach.

Definition 6 ***[Evaluation Graph]*** *We define the* evaluation graph *for a set of tasks $\mathcal{Q}$ as $D = \langle V_e, E_e\rangle$, where V_e is the set of nodes and E_e is the set of edges. The node $v_i \in V_e$ represents a task t_i. Let $t(v_i)$ denote the task represented by node v_i. E_e contains a directed edge from node v_i to node v_j iff $g(t(v_i), t(v_j)) < 0$. E_e contains an undirected edge from node v_i to node v_j iff $g(t(v_i), t(v_j)) = 0$.* □

Lemma 3 *No orientation of the undirected edges of an evaluation graph D can create a directed cycle involving one or more original directed edges.*

Definition 7 ***[Topological Ordering]*** *We define a* topological ordering*, O, of the nodes of an evaluation graph D as follows. For any pair of nodes, v_i and v_j, v_i appears before v_j in the topological ordering, if there exists a path from v_i to v_j with at least one directed edge. D does not have any cycle involving one or more directed edges (by Lemma 3). Hence at least one topological ordering exists.* □

Lemma 4 *Consider an evaluation graph $D = \langle V_e, E_e \rangle$. Consider a node $v_i \in V_e$, such that $\forall v_j \in V_e,\ (v_i \neq v_j) \Rightarrow g(t(v_i), t(v_j)) \neq 0$. We claim that v_i appears at a fixed position in all possible topological ordering of the nodes in D.*

Proof: Consider the set nodes, V_e^1, where $\forall v_j \in V_e^1,\ g(t(v_j), t(v_i)) < 0$. Clearly $\forall v_j \in V_e^1$, v_j will appear before v_i in any topological order. Similarly consider the set nodes, V_e^2, where $\forall v_k \in V_e^2,\ g(t(v_i), t(v_k)) < 0$. Clearly in this case, $\forall v_k \in V_e^2$, v_k will appear after v_i in any topological order. Since $V_e = V_e^1 \cup V_e^2 \cup \{v_i\}$, the location of v_j is fixed for all topological order. □

Theorem 1 *Given the set of tasks $\mathcal{Q}$ and the evaluation graph D, every topological ordering of D yields an evaluation sequence having the same expected penalty which is the minimum expected penalty.*

Proof: Let O_1 and O_2 be two topological orderings of D, which yield the evaluation sequences π_1 and π_2 respectively. According to Lemma 4, consider a node v_i, such that for all other nodes $v_i' \in V_e$, $g(t(v_i), t(v_i')) \neq 0$. The position of each such node v_i is the same in O_1 and O_2.

Therefore, O_1 and O_2 can differ only in the position of nodes of the type v_j and v_k, such that $g(t(v_j), t(v_k)) = 0$. By Lemma 2, the number of directed incoming edges to v_j and v_k are the same (say x). Likewise, the number of outgoing edges from v_j and v_k are also similar (say y). Clearly these x nodes will appear before v_j and v_k in both O_1 and O_2, and these y nodes will appear after v_j and v_k in both O_1 and O_2. Between the x nodes and the y nodes, another node v_j' can only appear, if $g(t(v_j), t(v_k)) = 0$. Therefore O_1 and O_2 can differ only in terms of such intermediate nodes where swapping two such nodes does not change the expected penalty. Therefore we can conclude that $\mathcal{P}(\pi_1) = \mathcal{P}(\pi_2)$.

Any evaluation sequence π that does not follow any topological ordering of D will contain at least a pair of consecutive tasks $\pi[i], \pi[i{+}1]$ such that $g(\pi[i], \pi[i{+}1]) > 0$. Clearly, swapping that pair of tasks, $\pi[i]$ and $\pi[i+1]$, will create an evaluation sequence with a lower expected penalty. Thus, any evaluation sequence that does not follow a topological ordering of D can not have the minimum expected penalty. Also all possible topological ordering yields evaluation sequences having same expected penalty. Hence, the expected penalty of an evaluation sequence that is based on a topological ordering of D has the minimum expected penalty. □

A alternative proof line for Theorem 1 can be derived from a theorem that is presented in (Smith 1956) in the context of a different class of problems, after proving some of the necessary properties (Lemma 1 and Lemma 2) of the functions presented in this paper. However, the proof presented in this paper is self-contained.

Minimizing the Expected Penalty

From Theorem 1 it follows that the evaluation sequence corresponding to a topological order has the minimum expected penalty. Therefore starting from any arbitrary evaluation sequence π_j of a set of atomic tasks $\mathcal{Q}$, evaluation sequence π_{min} with minimum expected penalty can be obtained by repeatedly swapping pair of successive elements of the evaluation sequence depending on the value of the g function on that pair. The complexity of such repetitive swapping is $O(n^2)$ where $|\mathcal{Q}| = n$. This time complexity can be further improved as follows. We define another function, h, on a task, t_i, as : $h(t_i) = \dfrac{s(t_i).c(t_i)}{\ell(t_i)} = \dfrac{s(t_i).c(t_i)}{1 - s(t_i)}$

Consider a set of tasks $\mathcal{Q}$ and an evaluation sequence π. For a pair of consecutive tasks, the relation between function h and function g is shown in the following equation, where $|\mathcal{Q}| = n$ and $k \in \mathbb{N}_1^{n-1}$.

$$\begin{aligned} g(\pi[k], \pi[k{+}1]) &= s(\pi[k]).c(\pi[k])\ell(\pi[k{+}1]) - \\ &\qquad s(\pi[k{+}1]).c(\pi[k{+}1])\ell(\pi[k]) \\ &= \ell(\pi[k]).\ell(\pi[k{+}1])\big[h(\pi[k]) - h(\pi[k{+}1])\big] \end{aligned}$$

Lemma 5 *Consider the evaluation sequence π of a set of tasks $\mathcal{Q}$. We claim that,*

$$\big[h(\pi[k]) - h(\pi[k{+}1]) > 0\big] \Leftrightarrow \big[g(\pi[k], \pi[k{+}1]) > 0\big]$$

where $|\mathcal{Q}| = n$ and $k \in \mathbb{N}_1^{n-1}$.

Lemma 6 *The function h is transitive.*

Theorem 2 *Given the set of tasks $\mathcal{Q}$, the complexity of determining π_{min} is $O(n.\log n)$, where $|\mathcal{Q}| = n$.*

Proof: From Theorem 1, for any evaluation sequence π_m of $\mathcal{Q}$, $\forall i \in \mathbb{N}_1^{n-1}, \big[g(\pi_m[i], \pi_m[i{+}1]) \leq 0\big] \Rightarrow \big[\mathcal{P}(\pi_m) = \mathcal{P}(\pi_{min})\big]$. Also from Lemma 5, it follows that $\big[h(\pi[k]) - h(\pi[k{+}1]) > 0\big] \Rightarrow \big[g(\pi[k], \pi[k{+}1]) > 0\big]$. Therefore, $\big(\forall i \in \mathbb{N}_1^{n-1}\big),\ \big[h(\pi_m[i]) - h(\pi_m[i{+}1]) \leq 0\big]$ is the sufficient condition for π_m to be the evaluation sequence with minimum expected penalty. Therefore, the evaluation sequence with minimum expected penalty, π_{min}, for a set of tasks, $\mathcal{Q}$, is obtained by sorting the tasks in the non decreasing order of the h values. Hence, the complexity of determining π_{min} is $O(n.\log n)$. □

Incorporating Ordering Constraints

Often practical considerations impose partial orders on the set of tasks represented by the solution graph of an AND/OR graph. For example in the context of SDGs represented by AND/OR graph, it is often the case that in a solution of a given SDG, the task associated with an intermediate service node cannot be performed until all other services on which that service node is dependent are completed. Such restrictions may also arise in the case of hierarchical problem decomposition also. We discuss such ordering constraints that can be imposed over the nodes of a solution graph and the algorithms to find the evaluation sequence of minimum expected penalty, under the ordering constraints. An ordering constraint is formally defined as follows.

Definition 8 ***[Ordering Constraint]*** *Consider a solution graph, S, of an AND/OR graph. An ordering constraint imposed on an intermediate node, v_q, of S, forces the subproblem represented by that node to be treated as an atomic task, that is, all terminal nodes of the subgraph rooted at v_q must be executed consecutively. Therefore an evaluation sequence, π, satisfies an ordering constraint at a node, v_q, if all the terminal nodes of the subgraph rooted at v_q appear consecutively in π.* □

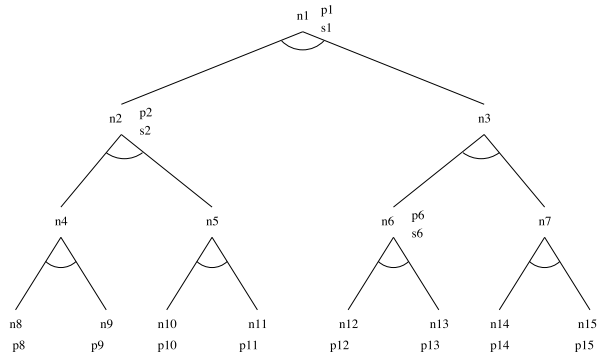

Figure 2: Solution Tree with Ordering Constraints

Consider the solution tree shown in Figure 2. In this solution tree the intermediate node, n_2 and n_6, highlighted using circles, have ordering constraints. Therefore, the set of tasks, $\{n_8, n_9, n_{10}, n_{11}\}$ and $\{n_{12}, n_{13}\}$, are executed atomically, i.e., if an evaluation sequence starts the execution of any of the tasks belonging to one of the mentioned set, all other tasks from that set must be executed consecutively.

Given a solution graph and a set of ordering constraints, our objective is to find the evaluation sequence with minimum expected penalty under the specified ordering constraint. To approach the problem we derive the swap function on a pair of evaluation sequences and then define the $\hat{h}$ function which is similar to the h function.

We reduce the expression of the expected penalty to a simpler expression. This reduced expression will be used to derive the swap function on a pair of evaluation subsequences. Consider a evaluation sequence $\pi = \langle t_1, \cdots, t_n \rangle$. The expression $\mathcal{P}(\pi)$ is reduced as follows:

$$\begin{aligned}
\mathcal{P}(\pi) &= \textstyle\sum_{k=2}^{n} \big[\prod_{i=1}^{k-1} s(t_i)\big].\big[\sum_{i=1}^{k-1} c(t_i)\big].\ell(t_k) \\
&= s(t_1)c(t_1)\big[1-s(t_2)\big] + s(t_1)s(t_2)\big[c(t_1)+c(t_2)\big]\big[1-s(t_3)\big] \\
&\quad + s(t_1)s(t_2)s(t_3)\big[c(t_1)+c(t_2)+c(t_3)\big]\big[1-s(t_4)\big] + \cdots \\
&= s(t_1)c(t_1) + s(t_1)s(t_2)c(t_2) + s(t_1)s(t_2)s(t_3)c(t_3) + \cdots \\
&\quad + s(t_1)\ldots s(t_n)c(t_n) - s(t_1)\ldots s(t_n)\big[c(t_1)+\cdots+c(t_n)\big] \\
&= \textstyle\sum_{i=1}^{n} x_i - y, \text{ where } x_i = \prod_{j=1}^{i} s(t_j).c(t_i), \text{ and} \\
&\qquad\qquad y = \textstyle\prod_{j=1}^{n} s(t_j).\big[\sum_{j=1}^{n} c(t_j)\big]
\end{aligned}$$

Definition 9 ***[Swap Function on a Pair of Evaluation Sequences]*** *Consider an evaluation sequence π of length n. Within π, π_1 and π_2 are two consecutive evaluation subsequences, where $\pi_1 = \langle \pi[a], \cdots, \pi[b-1] \rangle$, $\pi_2 = \langle \pi[b], \cdots, \pi[c] \rangle$ and $1 \leq a < b \leq c \leq n$. Let π' be the evaluation sequence where position of π_1 and π_2 are interchanged. The* swap function*, $\hat{f}$, on two consecutive evaluation subsequences, π_1 and π_2, is defined as follows:*

$$\begin{aligned}
\hat{f}(\pi_1, \pi_2) &= \mathcal{P}(\pi) - \mathcal{P}(\pi') = x_a + \cdots x_c - (x'_a + \cdots + x'_c) \\
&= A\big[X_1 + Y_1.X_2 - (X'_1 + Y'_1.X'_2)\big] \\
&= A\big[X_1 + Y_1.X_2 - (X_2 + Y_2.X_1)\big]
\end{aligned}$$

Here $A = \prod_{i=1}^{a-1} s(\pi[i])$, $Y_1 = Y'_2 = \prod_{i=a}^{b-1} s(\pi[i])$,
$Y_2 = Y'_1 = \prod_{i=b}^{c} s(\pi[i])$
$X_1 = X'_2 = \sum_{j=a}^{b-1} \Big(\Big[\prod_{i=a}^{j} s(\pi[i])\Big].c(\pi[j]\Big)$,
$X_2 = X'_1 = \sum_{j=b}^{c} \Big(\Big[\prod_{i=b}^{j} s(\pi[i])\Big].c(\pi[j]\Big)$, □

Definition 10 ***[Reduced Form of Swap Function on a Pair of Evaluation Sequences]*** *Consider two evaluation sequences, π_1 and π_2, corresponding to the sets of tasks, $\mathcal{Q}_1$ and $\mathcal{Q}_2$, where $|\mathcal{Q}_1| = n_1$ and $|\mathcal{Q}_2| = n_2$. The reduced form of swap function, $\hat{g}$, on a pair of evaluation sequences, π_1 and π_2, is defined as follows:*

$\hat{g}(\pi_1, \pi_2) = \big[X_1.(1-Y_2) - X_2.(1-Y_1)\big]$, *where*
$X_1 = \sum_{j=1}^{n_1} \big[\big(\prod_{i=1}^{j} s(\pi_1[i])\big).c(\pi_1[j]\big], Y_1 = \prod_{i=1}^{n_1} s(\pi_1[i])$
$X_2 = \sum_{j=1}^{n_2} \big[\big(\prod_{i=1}^{j} s(\pi_2[i])\big).c(\pi_2[j]\big], Y_2 = \prod_{i=1}^{n_2} s(\pi_2[i])$ □

We define function $\hat{h}$ on a evaluation sequence, π, of a set of tasks $\mathcal{Q}$, as : $\hat{h}(\pi) = \Big(\dfrac{X}{1-Y}\Big)$, where $|\mathcal{Q}| = n$,
$X = \sum_{j=1}^{n} \big[\prod_{i=1}^{j} s(\pi[i])\big].c(\pi[j],$ and $Y = \prod_{i=1}^{n} s(\pi[i])$
Consider the special case when $|\mathcal{Q}| = 1$, i.e., $\mathcal{Q}$ contains a single task, there can be only one possible evaluation sequence for $\mathcal{Q}$. In this case function $\hat{h}$ reduces to h.

Overall Approach

We address the problem of computing the evaluation sequence having minimum expected penalty under one or more ordering constraints into two stages. In the first stage the tasks in every subset (specified by a constraint), are ordered in the non-decreasing value of the h function. In other words, the best evaluation subsequence is computed for every such subset of tasks. In the second stage, we use the function $\hat{h}$ to order the evaluation subsequences corresponding to the subsets of tasks in order to get the evaluation sequence for the tasks in the given solution graph.

We show the computation of the sequence having minimum expected penalty under ordering constraints on the example solution tree shown in Figure 2. In the solution tree the ordering constraints are given on intermediate nodes, n_2 and n_6. Therefore the set of terminal nodes belonging to the solution tree is split into the following 4 subsets, $\mathcal{Q}_1 = \{n_8, n_9, n_{10}, n_{11}\}$, $\mathcal{Q}_2 = \{n_{12}, n_{13}\}$, $\mathcal{Q}_3 = n_{14}$, and $\mathcal{Q}_4 = n_{15}$. Let the evaluation sequences yielding minimum expected penalty for $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be π^1_{min} and π^2_{min} respectively. $\pi^1_{min} = \langle n_9, n_8, n_{10}, n_{11} \rangle$ and $\pi^2_{min} = \langle n_{13}, n_{12} \rangle$. The value of $\hat{h}$ function for $\mathcal{Q}_1, \cdots, \mathcal{Q}_4$ are - $\hat{h}(\mathcal{Q}_1) = 13.99$, $\hat{h}(\mathcal{Q}_2) = 12.23$, $\hat{h}(\mathcal{Q}_3) = 12$, and $\hat{h}(\mathcal{Q}_4) = 16$. The evaluation sequence having minimum expected penalty for the solution tree, π_{min}, is $\langle \underbrace{n_{14}}_{\mathcal{Q}_3}, \underbrace{n_{13}, n_{12}}_{\mathcal{Q}_2}, \underbrace{n_9, n_8, n_{10}, n_{11}}_{\mathcal{Q}_1}, \underbrace{n_{15}}_{\mathcal{Q}_4} \rangle$ and the minimum expected penalty is 12.06.

Generating the Solution of an AND/OR Graph

In this section we address the problem of generating the solution having minimum expected penalty for a given AND/OR graph. First we demonstrate the lack of optimal substructure property through a counter example and present an algorithm for solving the problem of computing the solution graph having *minimum expected penalty* for a given AND/OR graph.

Lack of Optimal Substructure Property

Traditional methods for finding minimum solution graphs of AND/OR graphs adopt a dynamic programming approach, where we traverse the graph bottom up marking the most favoured child at each OR node by analysing the problem represented by that OR node.

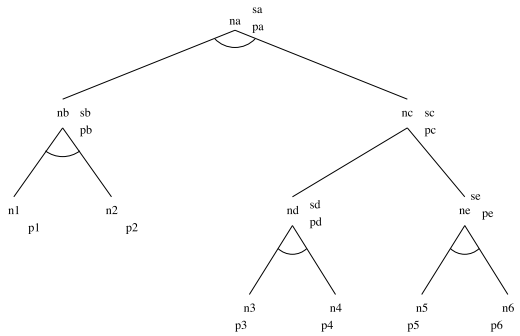

Figure 3: Example Showing the Lack of Optimal Substructure

For the problem presented in this paper we first demonstrate that the problem lacks the optimal substructure property. This is illustrated in the example shown in Figure 3. Here $n_1, \cdots, n_6$ are terminal nodes, and the $\langle probability, cost \rangle$ pair is shown below each terminal node. Suppose S_1 and S_2 are two solutions of the AND/OR tree shown in Figure 3, where $\mathcal{Q}(S_1) = \{n_1, n_2, n_3, n_4\}$ and $\mathcal{Q}(S_2) = \{n_1, n_2, n_5, n_6\}$. For solution S_1, $\pi_{min}(\mathcal{Q}(S_1)) = \langle n_3, n_1, n_4, n_2 \rangle$ and the minimum expected penalty is 133.5. Similarly, for S_2, $\pi_{min}(\mathcal{Q}(S_2)) = \langle n_5, n_6, n_1, n_2 \rangle$ and the minimum expected penalty is 101.5. The evaluation sequences with minimum expected penalty for node $n_a, \cdots, n_e$ are specified as $s_a, \cdots, s_e$ and the corresponding value of the minimum expected penalty is specified within square brackets. Here solution S_2 has the minimum expected penalty. Within S_2, the sub-solution rooted at node OR node, n_c, consists of terminal nodes n_5 and n_6 (the minimum expected penalty of this sub-solution is 19.20). However the optimal sub-solution rooted at node n_c, consists of terminal nodes n_3 and n_4 having minimum expected value 15.75. Clearly the optimal solution contains a sub-solution rooted at n_c which is not optimal. This shows the lack of optimal substructure property for this problem.

We conjecture that the problem of finding the solution with minimum expected penalty for a given AND/OR graph is NP hard, however the proof is open. Next, we present a cost recomputation based algorithm for the problem.

Overall Method

We have demonstrated that combining minimum expected penalty sub solutions may not generate the solution with minimum expected penalty. Our idea is to compute k-best solutions at every intermediate node instead of only one solution and recompute the expected penalty for every newly computed solution from the set of tasks belonging to that solution. Here k is an input parameter. Keeping k-best solutions at every child node, allows us to explore multiple combinations at the parent node. The bottom-up method of computing k-best solutions works by keeping k-best solutions at every child and then using them to compute the solutions at the parent node.

In the context of finding k-best solutions for valued sd-DNNF (Darwiche 2001) based structures that exhibit AND/OR DAG structures, a bottom-up method was proposed by Elliot (2007). The problem of finding minimum cost solution for AND/OR graphs with cycles has been well addressed by the research community (Bonet and Geffner 2005; Hansen and Zilberstein 2001; Jiménez and Torras 2000; Chakrabarti 1994). A bottom up method, REV*, for computing the minimum cost solution of explicit AND/OR graphs with cycles has been proposed by Chakrabarti.

In our proposed method CR-REV*, we use the idea of REV* for traversing the explicit AND/OR graph bottom-up while computing the k-best solutions at the intermediate nodes using cost recomputation; that is, at every intermediate AND node, while combining the solutions of the children nodes the expected penalty is computed directly from the set of tasks belonging to the new solution, instead of using the expected penalty values of the children solutions. We use the $visited$ flag for every vertex to keep track of the vertices which are processed during the bottom-up traversal. The visited flag for all vertices are initially set to false.

Algorithm 1: CR-REV*

```
   input  : An AND/OR graph G and k
   output : k solutions in the order of non-decreasing penalty
 1 Open ← ∅;
 2 foreach terminal node v_t ∈ G do Insert v_t to Open;
 3 while Open ≠ ∅ do
 4   v ← Remove from Open that element whose best
     solution has the least value for expected penalty;
 5   if v is not visited earlier then
 6     foreach parent node v_p of v do
 7       if ∀v_q ∈ S(v_p) visited(v_q) = true then
 8         if v_q is an AND node then
 9           Compute the k best combinations by combining
             the solutions of the children nodes;
10           Recompute the expected penalty of each of the
             combination;
11         else if v_q is an OR node then
12           Compute the k best selections from the solutions
             of the children nodes;
13         end
14         Insert v_p to Open;
15       else if v_q is an OR node then
16         if the expected penalty of the best solution of v is
           less than any of the current k best solutions of v_p
           then
17             Merge the solutions of v with the solutions v_p
               and keep the k best solution of v_p;
18             if v_p ∉ Open then Insert v_p to Open;
19       end
20     end
21   end
22 end
```

The computation of the k-best solution ((Elliott 2007))

has the running time of $O(|E|.k \log k + |E|. \log d + |V|.k \log d)$ and a space complexity of $O(k|E|)$ where $|V|$ and $|E|$ denotes the number of nodes and edges in the graph and every node has d children. The cost revision step requires $|V|$ steps at most. Hence, the time complexity of the proposed algorithm is $O(|E|.|V|.k \log k + |E|.|V|. \log d + |V|^2.k \log d)$.

Experimental Results

In order to demonstrate the effect of ordering to reduce the expected penalty, we have used the test cases provided for Web Services Challenge (2010). We have chosen 4 test cases (TC-1, $\cdots$, TC-4) and constructed the AND/OR graphs from the given service descriptions. The number of nodes in TC-1, TC-2, TC-3, and TC-4 are 3113, 3864, 6298, and 8279, respectively. Since these graphs contains cycles, we obtained a solution graph using REV* (Chakrabarti 1994). In the solution graph we assigned each task the following - (a) the probability of failure as a random number between 0.1 and 0.9, and (b) the rollback penalty as a random number between 200 to 20000. The details of the experimental result are presented in Figure 4 which shows the effect of execution ordering of the tasks on the expected penalty. Here we have reported the expected penalty under both best and worst ordering. It may be observed that typically the worst case expected penalty is more than two times of the best case expected penalty.

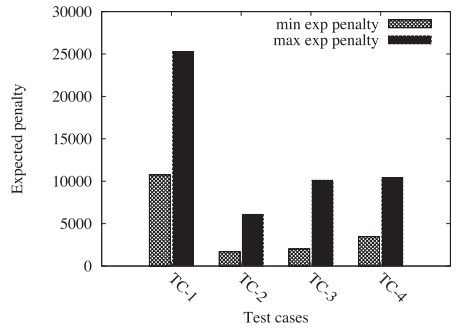

Figure 4: Min and max expected penalty for solutions of different SDGs

We have experimented with randomly constructed AND/OR trees. The trees are constructed by keeping the depth of the tree at 5 and varying the degree parameter for the intermediate OR nodes. The minimum degree parameter for the OR nodes, denoted by d_m, is varied from 3 to 7. We have created a set of 10 trees for every value of d_m by incrementing the degrees of OR nodes of the bottom most level of OR nodes.

The proposed method CR-REV* is used to generate k solutions, where k is an input to CR-REV*. We use different values of k (from the set $\{5, 10, 15, 20, 25\}$) and among the solutions generated by CR-REV* we compute the number of solutions, κ, that belong to the actual k-best solutions. In order to determine the number κ, first we construct the entire set of solutions and order them according to the non-decreasing value of expected penalty. Next we compare the top k solutions within this ordered set with the k solutions reported by CR-REV* method and count the number of matched solutions. It is important to note that the optimal solution is the best solution in the ordered set of all solutions. Since the problem lacks optimal substructure property, CR-REV* computes the minimum expected penalty of the solutions that are generated as a result of combining the sub-solutions at the AND nodes. Therefore, some solution that actually belong to the k-best solutions may be missed from the k solutions generated using CR-REV*.

In Figure 5, for each value of k in the set $\{5, 10, 15, 20, 25\}$, we report the average percentage of ordered solutions generated for each value of d_m. For a certain value of k, the average percentage of ordered solutions, θ, over $\#tc$ test cases is obtained using the following formula : $\theta = \frac{\sum_{i=1}^{\#tc} \kappa_i}{\#tc \times k} \times 100(\%)$, where κ_i denotes the number of solutions that belong to the actual k-best solutions for the i^{th} test-case. For our experimentation, $\#tc = 10$. It may be observed that for a fixed value of d_m the value of θ increases with the increase of k.

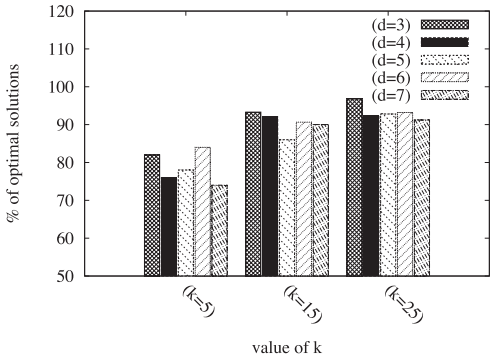

Figure 5: Average percentage of ordered solutions generated for different values of k

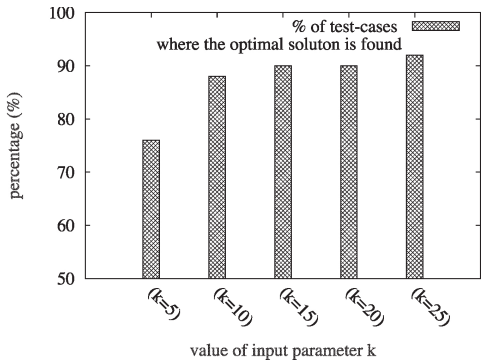

Figure 6: Percentage of test-cases for which the best solution was found for different values of k

In Figure 6, we report the percentage of test cases (among the 50 test cases used) for which the optimal solution is generated by CR-REV* for different values of k. It may be observed that for most of the cases the optimal solution is found with small value of k (less than 25).

Conclusion

This paper highlights the complexity of handling failure probabilities in AND/OR graph based problem decomposition approaches. It is not hard to envisage numerous applications in which failure probabilities must be factored into problem decomposition. However, the tractability of the problem remains an open question. We propose an approximate algorithm for finding the solution with minimum expected penalty for a given AND/OR graph and experimental results shows that the proposed algorithm performs well.

Acknowledgments

This work is partially supported by Department of Science and Technology, Govt. of India.

References

Bonet, B., and Geffner, H. 2005. An algorithm better than AO*? In *Proceedings of the 20th national conference on Artificial intelligence - Volume 3*, 1343–1347. AAAI Press.

Casali, G. L. 2010. *Ethical decision making and health care managers: developing managerial profiles based on ethical frameworks and other influencing factors*. Ph.D. Dissertation, Queensland University of Technology.

Chakrabarti, P. P. 1994. Algorithms for searching explicit AND/OR graphs and their applications to problem reduction search. *Artif. Intell.* 65(2):329–345.

Chang, C.-L., and Slagle, J. R. 1971. An admissible and optimal algorithm for searching AND/OR graphs. *Artif. Intell.* 2(2):117–128.

Darwiche, A. 2001. Decomposable negation normal form. *J. ACM* 48:608–647.

Demeulemeester, E.; Deblaere, F.; Herbots, J.; Lambrechts, O.; and de Vonder, S. V. 2007. A multi-level approach to project management under uncertainty. *Review of Business and Economics* 0(3):391–409.

Elliott, P. H. 2007. Extracting the k best solutions from a valued And-Or acyclic graph. Master's thesis, Massachusetts Institute of Technology.

Gaaloul, W.; Bhiri, S.; and Rouached, M. 2010. Event-based design and runtime verification of composite service transactional behavior. *IEEE T. Services Computing* 3(1):32–45.

Ghosh, P.; Chakrabarti, P. P.; and Dasgupta, P. 2011. Solving AND/OR graphs with failure probabilities. http://cse.iitkgp.ac.in/-prynkrg/TechReports/PWP-TR.pdf. Technical Report.

González, V.; Alarcón, L. F.; Maturana, S.; Mundaca, F.; and Bustamante, J. 2010. Improving planning reliability and project performance using the reliable commitment model. *Journal of Construction Engineering and Management* 136(10):1129–1139.

Gu, Z.; Li, J.; and Xu, B. 2008. Automatic service composition based on enhanced service dependency graph. In *Web Services, 2008. ICWS '08. IEEE International Conference on*, 246 –253.

Gu, Z.; Xu, B.; and Li, J. 2010. Service data correlation modeling and its application in data-driven service composition. *Services Computing, IEEE Transactions on* 3(4):279–291.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2):35–62.

Jiménez, P., and Torras, C. 2000. An efficient algorithm for searching implicit AND/OR graphs with cycles. *Artif. Intell.* 124:1–30.

Liu, A., and Li, Q. 2007. Ensuring consistent termination of composite web services. In *SIGMOD'07 Ph. D. Workshop*.

Liu, A.; Li, Q.; Huang, L.; Xiao, M.; and Liu, H. 2008. QoS-aware scheduling of web services. In *WAIM*, 171–178.

Liu, A.; Li, Q.; Huang, L.; and Xiao, M. 2010. Facts: A framework for fault-tolerant composition of transactional web services. *IEEE T. Services Computing* 3(1):46–59.

Martelli, A., and Montanari, U. 1973. Additive and/or graphs. In *Proceedings of the 3rd international joint conference on Artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Montagut, F., and Molva, R. 2006. Augmenting web services composition with transactional requirements. In *ICWS*, 91–98.

Mori, M., and Tseng, C. 1997. A resource constrained project scheduling problem with reattempt at failure: A heuristic approach. *Journal of the Operations Research* 40(1):33–44.

Nilsson, N. J. 1980. *Principle of artificial intelligence*. Tioga Publishing Co.

Nozik, L.; Turnquist, M.; and List, G. 2001. Project management under uncertainty with applications to new product development. In *Change Management and the New Industrial Revolution, 2001. IEMC '01 Proceedings.*, 394–399.

Pearl, J. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Pich, M. T.; Loch, C. H.; and Meyer, A. D. 2002. On uncertainty, ambiguity, and complexity in project management. *Manage. Sci.* 48(8):1008–1023.

Ren, L.; Bai, S.; and Guo, Y. 2010. Project scheduling in and-or graphs based on design structure matrix. In *Emergency Management and Management Sciences (ICEMMS), 2010 IEEE International Conference on*, 161–164.

Shin, D.-H.; Jeon, H.-B.; and Lee, K.-H. 2010. A sophisticated approach to composing services based on action dominance relation. In *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*, 164 –170.

Smith, W. E. 1956. Various optimizers for single-stage production. *Naval Research Logistics Quarterly* 3:59–66.

WSChallenge. 2010. Web services challenge 2010. http://ws-challenge.georgetown.edu/wsc10/.