International Journal of Foundations of Computer Science,


# Vulverability of Complex Systems and Critical Vertices in AND/OR Graphs*

Yvo Desmedt
*Computer Science, Florida State University, Tallahassee, Florida, FL 32306-4530, USA, desmedt@cs.fsu.edu*

and

Yongge Wang
*Department of Software and Information Systems, University of North Carolina at Charlotte, 9201 University City Blvd., Charlotte, NC 28223, yonwang@uncc.edu*

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

**Abstract**

AND/OR graphs and minimum-cost solution graphs have been studied extensively in artificial intelligence (see, e.g., Nilsson [14]). Generally, the AND/OR graphs are used to model problem solving processes. The minimum-cost solution graph can be used to attack the problem with the least resource. However, in many cases we want to solve the problem within the shortest time period and we assume that we have as many concurrent resources as we need to run all concurrent processes. In this paper, we will study this problem and present an algorithm for finding the minimum-time-cost solution graph in an AND/OR graph. We will also study the following problems which often appear in industry when using AND/OR graphs to model manufacturing processes or to model problem solving processes: finding maximum (additive and non-additive) flows and critical vertices in an AND/OR graph. A detailed study of these problems provide insight into the vulnerability of complex systems such as cyber-infrastructures and energy infrastructures (these infrastructures could be modeled with AND/OR graphs). For an infrastructure modeled by an AND/OR graph, the protection of critical vertices should have highest priority since terrorists could defeat the whole infrastructure with the least effort by destroying these critical points. Though there are well known polynomial time algorithms for the corresponding problems in the traditional graph theory, we will show that generally it is **NP**-hard to find a non-additive maximum flow in an AND/OR graph, and it is both **NP**-hard and co**NP**-hard to find a set of critical vertices in an AND/OR graph. We will also present a polynomial time algorithm for finding a maximum additive flow in an AND/OR graph, and discuss the relative complexity of these problems.

*Research supported by DARPA F30602-97-1-0205.

## 1. Introduction

Structures called AND/OR graphs are useful for depicting the activity of production systems (see, e.g., Nilsson [14]). Wang, Desmedt, and Burmester [19] used AND/OR graphs to make a critical analysis of the use of redundancy to achieve network survivability in the presence of malicious attacks. That is, they used AND/OR graphs to model redundant computation systems consisting of components which are based on computations with multiple inputs. Roughly speaking, an AND/OR graph is a directed graph with two types of vertices, labeled $\wedge$-*vertices* and $\vee$-*vertices*. The graph must have at least one input (source) vertex and one output (sink) vertex. In this case, processors which need all their inputs in order to operate could be represented by $\wedge$-vertices, whereas processors which can choose (using some kind of voting procedure) one of their "redundant" inputs could be represented by $\vee$-vertices. It should be noted that our following definition is different from the standard definitions in artificial intelligence (see, e.g., [14]). That is, the directions of the edges are opposite. The reason is that we want to use the AND/OR graphs to model redundant computation systems too.

**Definition 1** *An* AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$ *is a graph with a set* $V_\wedge$ *of* $\wedge$-vertices*, a set* $V_\vee$ *of* $\vee$-vertices*, a set* $INPUT \subset V_\wedge$ *of* input vertices*, an* output vertex $output \in V_\vee$*, and a set of* directed edges $E$*. The input vertices have no incoming edges and the output vertex has no outgoing edges.*

Assume that an AND/OR graph is used to model a redundant computation system or a problem solving process. Then, information (for example, mobile codes) must flow from the input vertices to the output vertex. And a valid computation in an AND/OR graph can be described by a *solution graph*.

**Definition 2** *Let* $G(V_\wedge, V_\vee, INPUT, output; E)$ *be an AND/OR graph. A* solution graph $P = (V_P, E_P)$ *is a minimum subgraph of* $G$ *satisfying the following conditions.*

1. $output \in V_P$.
2. *For each* $\wedge$*-vertex* $v \in V_P$*, all incoming edges of* $v$ *in* $E$ *belong to* $E_P$.
3. *For each* $\vee$*-vertex* $v \in V_P$*, there is exactly one incoming edge of* $v$ *in* $E_P$.
4. *There is a sequence of vertices* $v_1, \ldots, v_n \in V_P$ *such that* $v_1 \in INPUT$*,* $v_n = output$*, and* $(v_i \rightarrow v_{i+1}) \in E_P$ *for each* $i < n$.

Wang, Desmedt, and Burmester [19] have studied the problem of finding vertex disjoint solution graphs in an AND/OR graph. Specifically, they have showed that it is **NP**-hard to find vertex disjoint solution graphs in an AND/OR graph. These problems are mainly related to that of achieving dependable computation using redundancy.

Minimum-cost solution graphs have been studied extensively in artificial intelligence and many heuristic algorithms for finding minimum-cost solution graphs have been presented (see, e.g., [1,4,5,10,11,12,13,14,15]). When a problem solving process is modeled by an AND/OR graph, the minimum-cost solution graph can be used to attack the problem with the least resource. However, in many cases we want to solve the problem within the shortest time period and we assume that we have as many concurrent resources as we need to run all concurrent processes. In Section 2, we will study this problem and present an

algorithm for finding the minimum-time-cost solution graph in an AND/OR graph.

Maximum-flow minimum-cut theorem (see, e.g., [2,6,7,9,18]) has played an important role in the study of networks. For example, it is used to direct the traffic in networks such as the Internet (see, e.g., [2]). However, this theorem is only for networks that have one kind of vertices: $\vee$-vertices (that is, directed graphs). And it is not applicable for networks which can be modeled by AND/OR graphs. Indeed, in artificial intelligence and distributed computation systems, the realization of many important projects (such as the construction of a dam, of a shopping center, of a housing estate or of an aircraft; the carrying out of a sequence of manufacturing steps; the programming of a test flight of an aircraft; etc.) are dependent on the computations with multiple inputs and many problems in the realization are that of finding maximum flows in AND/OR graphs.

A detailed study of maximum flows and critical vertices in AND/OR graphs provide insight into the vulnerability of complex systems such as cyber-infrastructures and energy infrastructures (these infrastructures could be modeled with AND/OR graphs). For an infrastructure modeled by an AND/OR graph, the protection of critical vertices should have highest priority since terrorists could defeat the whole infrastructure with the least effort by destroying these critical points.

In this paper, we will consider the following problem: is there an equivalent theorem of maximum-flow minimum-cut theorem for AND/OR graphs? That is, does there exist a polynomial time heuristic algorithm for finding maximum flows in AND/OR graphs? We will show that this problem is **NP**-hard. We will also consider the problems of finding critical vertices in AND/OR graphs and show these problems are even "harder", that is, they lie in the second level of the polynomial time hierarchy, which is believed to be harder than **NP**-complete problems. However, if we modify the flow structure and make it additive, as Martelli and Montanari [12,13] did for cost structures, then we will have a polynomial time (heuristic) algorithm for finding maximum additive flows in ADN/OR graphs.

The organization of the paper is as follows. We first present in Section 2 a polynomial time algorithm for finding a minimum-time-cost solution graph in an AND/OR graph. In Section 3 we discuss the problem of finding maximum flows in AND/OR graphs and prove the **NP**-hardness of several problems. Section 4 is devoted to the problem of finding critical vertices in an AND/OR graph. Several problems related to critical vertices are shown to be "harder", that is, they lie in the second level of the polynomial time hierarchy. In Section 5 we present a polynomial time algorithm for finding maximum additive flows in AND/OR graphs.

We will use (fairly standard) notions of complexity theory. We refer the reader to [8] for definitions of these. Here we only give an informal description of a few notions. A *polynomial time many one reduction* (denoted by $\leq_m^p$) from a problem $A$ to another problem $B$ is a polynomial time computable function $f$ with the property that $f(x) \in B$ if and only if $x \in A$ for all inputs $x$. A *polynomial time Turing reduction* (denoted by $\leq_T^p$) from a problem $A$ to another problem $B$ is a polynomial time oracle Turing machine $M$ with the property that, for any input $x$, the Turing machine $M$ with access to the oracle $B$ will decide in polynomial time whether $x \in A$. For a complexity class $\mathcal{C}$ and a problem $A \in \mathcal{C}$, by saying that $A$ is $\mathcal{C}$-complete we mean that every

problem in $\mathcal{C}$ can be reduced to $A$ by a polynomial time many one reduction.

## 2. Minimum-time-cost solution graphs and PERT graphs

The shortest path problem and maximum flow problem are among the oldest problems in graph theory (see, e.g., [6,9,18]). They appear either directly, or as subsidiary problems, in many applications. Amongst others, we can mention the following: vehicle routing problems, some problems of investment and of stock control, many problems of dynamic programming with discrete states and discrete time, network optimization problems, and the problem of a continuous electrical current through a network of dipoles. However, as showed in [19], traditional graphs do not present a model for all problems in practice, e.g., it is always the case that a processor needs more than one type of inputs. AND/OR graphs seem to be a possible candidate for modeling these problems with multiple inputs. For a given AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$, if we associate with each edge $e \in E$ a rational number $l(e)$ called the length of the edge, then we can define a minimum-cost solution graph of $G$ to be a solution graph $P(V_P, E_P)$ in $G$ whose total length

$$l(P) = \sum_{e \in E_P} l(e)$$

is a minimum. In addition to its important applications in artificial intelligence (see, e.g., [14,15]), minimum-cost solution graphs have many practical applications in distributed computation systems with multiple inputs, because length $l(e)$ may equally well be interpreted as being a cost of transportation along $e$, the time through $e$, and so on. Chang and Slagle [5] have proposed a heuristic search algorithm for finding minimum-cost solutions in an AND/OR graph, but it was subsequently shown by Sahni [16] that with above definition of the cost, this problem is **NP**-hard. Thus their algorithm cannot be implemented efficiently in practice. By modifying the cost structure and making it additive, Martelli and Montanari [12,13] were able to formulate a polynomial time "marking" (with or without heuristic functions) algorithm AO$^*$ for AND/OR graphs (see also [1,4,10,11] for more discussions on the heuristic algorithm AO$^*$). Roughly speaking, in the new cost structure, the cost of one edge may be counted as many times as it will be used in the unfolded AND/OR tree of the solution graph.

In the practice of distributed computation systems, many systems can be modeled by AND/OR graphs with only one $\vee$-vertex, that is, the output vertex is the only $\vee$-vertex. For such kind of AND/OR graphs, it is easy to find minimum-cost solutions in them with respect to the above "non-additive" cost definition.

**Theorem 1** *There is a polynomial time algorithm to find a minimum-cost solution graph in an AND/OR graph with only one $\vee$-vertex.*

**Proof.** Given an AND/OR graph $G$ with only one $\vee$-vertex, it is straightforward that there are at most $k$ solution graphs in it, where $k$ is the number of incoming edges of the $\vee$-vertex *output*. Whence it is easy to find a minimum-cost solution graph in it by an exhaustive search. □.

For a problem solving process modeled by an AND/OR graph, the minimum-cost solution graph can be used to attack the problem with the least resource. However, in many cases we want to solve the problem within the shortest time

period and we assume that we have as many concurrent resources as we need to run all concurrent processes. In the following, we present an algorithm for finding minimum-time-cost solution graphs in acyclic AND/OR graphs.

First we present the definition of *PERT digraphs* (**Program Evaluation and Review Technique**). A *PERT digraph* is an AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$ with the following properties:

1. $INPUT = \{in\}$ has only one element.
2. $G$ has no directed circuits.
3. $G$ has only one $\vee$-vertex *output* and *output* has only one incoming edge.
4. Every vertex $v \in V_\wedge$ is on some directed path from *in* to *output*.

PERT digraphs have been used to model the central scheduling problems (see, e.g., [2,6,9]). A PERT digraph has the following interpretation. Every edge represents a process. All the processes which are represented by edges of $in^+$, can be started right away. For every vertex $v$, the processes represented by edges of $v^+$ can be started when all the processes represented by edges of $v^-$ are completed. Note that we use $v^-$ and $v^+$ to denote the incoming and outgoing edges of $v$ respectively. For a given PERT digraph, we want to know how soon the whole project can be completed; that is, what is the shortest time, from the moment the processes represented by $in^+$ are started, until the process represented by $output^-$ is completed. We assume that the resources for running the processes are unlimited. For this problem to be well defined let us assume that each $e \in E$ has an assigned *length* $l(e)$, which specifies the time it takes to execute the process represented by $e$. The minimum completion time can be found by the following algorithm:

1. Assign *in* the label $O(\lambda(in) \leftarrow 0)$. All other vertices are "unlabeled".
2. Find a vertex, $v$, such that $v$ is unlabeled and all edges of $v^-$ emanate from labeled vertices. Assign
$$\lambda(v) \leftarrow \max_{e=(u \rightarrow v)} \{\lambda(u) + l(e)\}.$$
3. If $v = output$, halt; $\lambda(output)$ is the minimum completion time. Otherwise, go to Step 2.

For more discussions on the above algorithm it is referred to [2,6,9].

We define a *redundant PERT digraph* to be an AND/OR graph with the following properties:

1. $INPUT = \{in\}$ has only one element.
2. $G$ has no directed circuits.
3. Every vertex $v \in V_\wedge \cup V_\vee$ is on some directed path from *in* to *output*.

As in a PERT digraph graph, every edge in a redundant PERT digraph represents a process. All the processes which are represented by edges of $in^+$, can be started right away. For every $\vee$-vertex $v$, the processes represented by edges of $v^+$ can be started when any one of the processes represented by edges of $v^-$ is completed. And for every $\wedge$-vertex $v$, the processes represented by edges of $v^+$ can be started when all the processes represented by edges of $v^-$ are completed.

Our problem deals with the question of finding a solution graph in a redundant PERT digraph such that the minimum completion time of the solution

graph is a minimum. That is, if we have enough resources to run these processes concurrently, then we can solve the problem within the shortest time period. Such kind of minimum-time-cost solution graphs can be found by the following algorithm.

1. Assign *in* the label $O(\lambda(in) \leftarrow 0)$. All other vertices are "unlabeled".
2. Find a vertex, $v$, such that $v$ is unlabeled and all edges of $v^-$ emanate from labeled vertices. If $v$ is an $\wedge$-vertex, then assign
$$\lambda(v) \leftarrow \max_{e=(u \rightarrowtail v)} \{\lambda(u) + l(e)\};$$
Otherwise assign
$$\lambda(v) \leftarrow \min_{e=(u \rightarrowtail v)} \{\lambda(u) + l(e)\};$$
3. If $v = output$, halt; $\lambda(output)$ is the minimum completion time of the minimum-time-cost solution graph. Otherwise, go to Step 2.

In Step 2, the existence of a vertex $v$, such that all the edges of $v^-$ emanate from labeled vertices is guaranteed by Condition (2) and (3) of the definition for a redundant PERT digraph: If no unlabeled vertex satisfies the condition then for every unlabeled vertex, $v$, there is an incoming edge which emanates from another unlabeled vertex. By repeatedly tracing back these edges, one finds a directed circuits. Thus if such vertex is not found then we conclude that either Condition (2) or (3) does not hold.

It is easy to prove, by induction on the order of the labeling, that $\lambda(output)$ is the minimum completion time of the minimum-time-cost solution graph.

Once the algorithm terminates, by going back from *output* to *in*, via the edge which determined the label of the vertex, we can trace the minimum-time-cost solution graph. Clearly, they may be more than one minimum-time-cost solution graph.

## 3. The maximum flow problem in an AND/OR graph

Given an AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$, a *capacity function* $c$ associated with $G$ is a positive integral function defined on edges of $G$. A *flow* $f$ in $G$ is a positive integral function defined on edges of $G$ with the following properties: for all $e \in E$,

$$0 \leq f(e) \leq c(e),$$

for all $v \in V_\vee$,

$$\sum_{e \in v^-} f(e) = \sum_{e \in v^+} f(e) \tag{1}$$

where $v^-$ is the set of incoming edges of $v$ and $v^+$ is the set of the outgoing edges of $v$, and for all $v \in V_\wedge$,

$$\forall e_1, e_2 \in v^-(f(e_1) = f(e_2)) \text{ and } \forall e_1 \in v^- \forall e_2 \in v^+(f(e_2) \leq f(e_1)). \tag{2}$$

For a vertex $v \in V_\vee \cup V_\wedge$, the *amount of flow* into $v$ is defined to be the value $\sum_{e \in v^-} f(e)$. For a flow $f$ in the AND/OR graph $G$, the *total flow* $F_f(G)$ is defined to be the amount of flow into the output vertex *output*. And we will use $F_c(G)$ to denote the maximum of $F_f(G)$ for all flows $f$ in $G$, that is, $F_c(G) = \max\{F_f(G) : f \text{ is a flow in } G\}$.

Applications of the theory of flows in AND/OR graphs are extremely numerous and varied. For example, the optimal design and expansion of computation systems with multiple inputs, and the optimal design of a production manufacturing process. Though there are polynomial time algorithms for finding maximum flows in traditional graphs, we will show that the equivalent problem for AND/OR graphs is **NP**-hard. Specifically, we will show that the following problem MFAO is **NP**-hard.

MFAO (i.e., Maximum Flows for AND/OR Graphs).
*Instance:* An AND/OR graph $G$, a capacity function $c$ associated with $G$, and a positive integer $k$.
*Question:* Does there exist a flow $f$ in $G$ such that the total flow $F_f(G)$ is at least $k$?

**Theorem 2** *MFAO is* **NP***-complete.*

**Proof.** It is clear that MFAO $\in$ **NP**, whence it suffices to reduce the following **NP**-complete problem MI to MFAO.

MI (i.e., Maximum Independent set).
*Instance:* A graph $G(V, E)$, and a positive integer $k$.
*Question:* Does there exist an independent set $V' \subseteq V$ of $G$ such that $|V'| \geq k$, that is, any two vertices of $V'$ is not connected by an edge in $E$?

The input $G = (V_G, E_G)$, to MI, consists of a set of vertices $V_G = \{v_1, \ldots, v_n\}$ and a set of edges $E_G$. In the following we construct an AND/OR graph $MG(V_\wedge, V_\vee, INPUT, output; E)$ and a capacity function $c(e) = 1$ $(e \in E)$ (the input to MFAO) such that there is an independent set of size $k$ in $G$ if and only if there is a flow $f$ in $MG$ such that the total flow $F_f(MG)$ is at least $k$.

Let $INPUT = \{I_i, I_{i,j} : i, j = 1, \ldots n\}$, $V_\vee = \{output\} \cup \{u_{i,j} : i, j = 1, \ldots n\}$, $V_\wedge = INPUT \cup \{u_i : i = 1, \ldots, n\}$, and $E$ be the set of the following edges.

1. For each $i = 1, \ldots, n$, there is an edge $I_i \rightarrow u_i$.
2. For each pair $i, j = 1, \ldots, n$, there is an edge $I_{i,j} \rightarrow u_{i,j}$.
3. For each pair $i, j = 1, \ldots, n$, such that the unordered pair $(v_i, v_j) \in E_G$, there are four edges $u_{i,j} \rightarrow u_i$, $u_{i,j} \rightarrow u_j$, $u_{j,i} \rightarrow u_i$, and $u_{j,i} \rightarrow u_j$.
4. For each $i$, there is an edge $u_i \rightarrow output$.

And the capacity function is defined by letting $c(e) = 1$ for all $e \in E$. Now it is easy to see that, for any flow $f$ in $MG$, if both $f(u_i \rightarrow output) = 1$ and $f(u_j \rightarrow output) = 1$ $(i \neq j \leq n)$, then $(v_i, v_j) \notin E_G$. Whence there is a flow $f$ in $MG$ such that the total flow $F_f(MG)$ equals to $k$ if and only if there is an independent set of size $k$ in $G$. $\square$.

Indeed, MFAO is **NP**-complete for $k = 1$ (i.e., $k$ is not a part of the input).

**Theorem 3** *MFAO is* **NP***-complete for* $k = 1$.

**Proof.** It is clear that MFAO $\in$ **NP**. In order to prove that MFAO is **NP**-hard for $k = 1$, we first define the **NP**-complete problem $3SAT$ as follows. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a finite set of *variables*. A *literal* is either a variable $x_i$ or its *complement* $\bar{x}_i$. Thus, the set of literals is $L = \{x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$. A *clause* $C$ is a 3-element subset of $L$. We

are given a set of clauses $C_1, C_2, \ldots, C_m$, each of which consists of 3 literals. The question is whether the set of variables can be assigned values $T$ (true) or $F$ (false), so that each clause contains at least one literal with a $T$ value. A clause is satisfied under an assignment if the clause contains at least one literal with a $T$ value. The concise statement of $3SAT$ is, therefore, the following:

*Instance:* A set of clauses.
*Question:* Is there an assignment of the literals such that all the clauses are satisfied?

Now we reduce the **NP**-complete problem $3SAT$ to MFAO with $k = 1$.

The input $C$, to $3SAT$, consists of clauses $C_1, C_2, \ldots, C_m$, each a 3-element subset of the set of literals $L = \{x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$. In the following we construct an AND/OR graph $g(C)$ and a capacity function $c$ with the property that: $C$ is satisfiable if and only if there is a flow $f$ in $g(C)$ such that the total flow $F_f(g(C))$ in $g(C)$ is 1.

For each variable $x_i$ we construct two $\wedge$-vertices $v_i$ and $\bar{v}_i$ and one $\vee$-vertex $u_i$, as shown in Figure 1. For the reason of convenience, we use hexagons to denote $\wedge$-vertices and rectangles to denote $\vee$-vertices. There is an input vertex *in* which is connected by an edge to $u_0$ which is connected again by two edges to the vertices $v_1$ and $\bar{v}_1$ respectively. The vertices for variables are connected in series: for $i < n$, both $v_i$ and $\bar{v}_i$ are connected by edges to $u_i$, and $u_i$ is connected by edges to $v_{i+1}$ and to $\bar{v}_{i+1}$. $u_n$ is connected by an edge to the $\wedge$-vertex $u_c$. In addition, there are $\vee$-vertices $c_1, c_2, \ldots, c_m$ and an edge from each to the $\wedge$-vertex $u_c$. For each occurrence of $x_i$ ($\bar{x}_i$), there is an edge from $v_i$ ($\bar{v}_i$) to the vertex $c_j$, the clause in which it occurs. Lastly, there is one edge from $u_c$ to the output vertex *output*. The capacity function $c$ is defined by letting $c(e) = 1$ for all $e \in E_{g(C)}$.

It is easy to see that any flow $f$ in $g(C)$ will go through either $v_i$ or $\bar{v}_i$ (but not both) for each $i \leq n$, and will go through each edge from $c_j$ ($j \leq m$) to $u_c$ since $u_c$ is an $\wedge$-vertex. Also, for each $j \leq m$, $f$ goes through exactly one edge from some vertex $v_i$ or $\bar{v}_i$ to the $\vee$-vertex $c_j$.

Thus, if the answer to $g(C)$, with respect to MFAO for $k = 1$, is positive, then we can use the flow $f$ to assign a satisfying assignment of the literals as follows: if $f$ goes through $v_i$, assign $x_i = T$, and if through $\bar{v}_i$, $x_i = F$. In this case, the answer to $C$, with respect to $3SAT$, is also positive.

Conversely, assume that there is a satisfying assignment of the variables. If $x_i = T$, let $f$ use $v_i$; if $x_i = F$, use $\bar{v}_i$. Now, let $\xi$ be a 'true' literal in $C_j$. If $\xi = x_i$ then let $f$ uses the edge from $v_i$ to $c_j$; if $\xi = \bar{x}_i$, use the edge from $\bar{v}_i$ to $c_j$. Finally, use the $m+1$ edges entering $u_c$ and the edge from $u_c$ to *output* to form the flow $f$. □.

In traditional graph theory, the problem of finding maximum flows in a graph is closely related to the problem of deciding the connectivity of the graph. However, Theorem 3 shows that there is a big difference between these two corresponding problems in AND/OR graphs. Obviously, there is at least one solution graph in the AND/OR graph $g(C)$ (see Figure 1), but it may be the case that there is no nonzero flow in it. The following example presents an AND/OR graph which has a nonzero flow but does not have a solution graph in it.

**Example 1** *Let $G$ be the AND/OR graph in Figure 2 and $c$ be the capacity function defined by letting $c(e) = 1$ for all $e \in E$. Define a flow $f$ by letting*

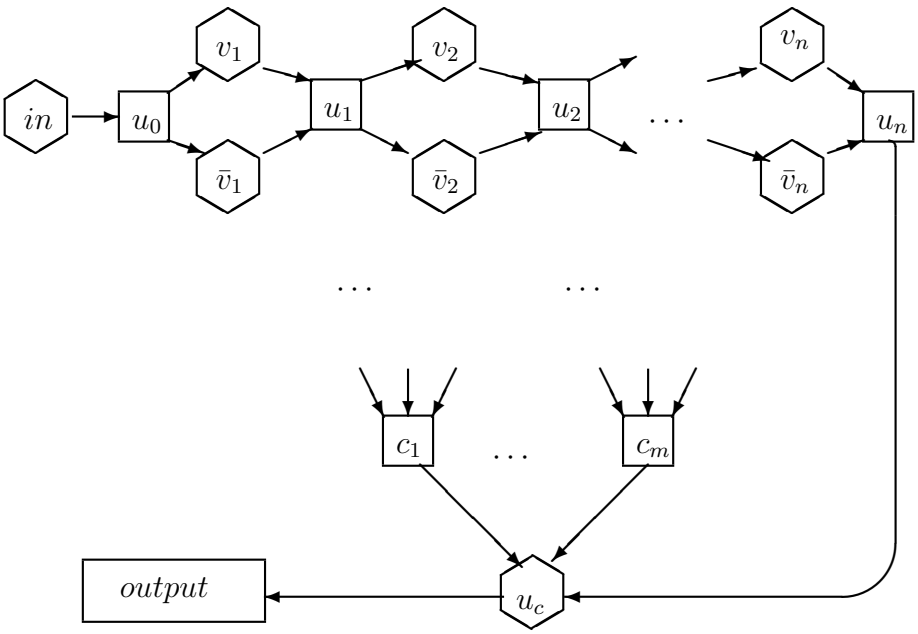


Figure 1: The AND/OR graph $g(C)$

$f(v_2 \to v_3) = 1, f(v_3 \to v_2) = 1, f(v_2 \to output) = 1, f(v_3 \to output) = 1$, $f(v_3 \to v_1) = 0$, *and* $f(in \to v_1) = 0$. *Then* $F_f(G) = 2$ *and it is clear that there is no solution graph in* $G$.

In Section 5, we will show that there is a polynomial time algorithm for finding maximum flows in AND/OR graphs with only one $\vee$-vertex.

## 4. The problem of finding critical vertices in an AND/OR graph

In this section, we assume the familiarity with the complexity classes within the Polynomial time Hierarchy like $\Sigma_n^p$ and $\Pi_n^p$. For more details, it is referred to Stockmeyer [17].

Let us consider the following scenarios: A redundant computation system (or a problem solving process) with multiple inputs (e.g., the electrical power distribution systems, the air traffic control system, etc.) is modeled by an AND/OR graph $G$ with a capacity function $c$ associated with it. And an adversary has the power to destroy $k$ processors (that is, $k$ vertices of the graph $G$) of the system. Then the adversary wants to know how to choose $k$ vertices in the graph such that the destruction of these vertices results in the largest damage to the system. In another words, he wants to remove $k$ vertices from the AND/OR graph such that the maximum flows of the resulting AND/OR graphs (in this case, the flows coming from the corrupted vertices are all 0) is a minimum. The designer of the system is also concerned with this scenario because he wants to know how robust his system is.

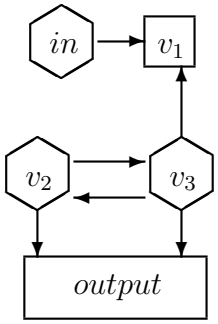


Figure 2: The AND/OR graph $G$

In order to state our problem more precisely, we first give two definitions. Given an AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$ with a capacity function $c$ and a vertex set $U \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$, the capacity function $c_U$ is defined by

$$c_U(e) = \begin{cases} 0 & \text{if } e \text{ is an outgoing edge of some vertex in } U, \\ c(e) & \text{otherwise.} \end{cases}$$

And for a number $k > 0$, a *set of critical vertices with respect to both $c$ and $k$* is a vertex set $U \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$ with the following properties:

- $|U| \leq k$.
- If $F_{c_U}(G)$ is the maximum (for the definition see the previous section) of all total flows in $G$ with respect to the capacity function $c_U$ then, for any other vertex set $U' \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$ with $|U'| = k$, $F_{c_U}(G) \leq F_{c_{U'}}(G)$, where $F_{c_{U'}}(G)$ is the maximum of all total flows in $G$ with respect to the capacity function $c_{U'}$.

Whence the concise statement of our problem is the following: Given an AND/OR graph $G$ with a capacity function $c$ and a positive integer $k$, how can one find a set of critical vertices with respect to both $c$ and $k$? We can show that this problem is **NP**-hard. Indeed, we can prove that the problem of deciding whether a given set of vertices is critical is in $\Pi_2^p$ and is both **NP**-hard and co**NP**-hard.

CV (i.e., Critical Vertices).
*Instance:* An AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$ with a capacity function $c$, and a vertex set $U \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$.
*Question:* Is $U$ a set of critical vertices with respect to both $c$ and $|U|$?

It is clear that $U$ is a set of critical vertices if and only if for all flows $f_U$ in $G$ (with respect to the capacity function $c_U$) and for all $U' \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$ of size $|U|$, there is a flow $f_{U'}$ in $G$ (with respect to the capacity function $c_{U'}$) such that $F_{f_U}(G) \leq F_{f_{U'}}(G)$. Whence CV is in $\Pi_2^p = \text{co}\Sigma_2^p$, that is, the second level of the polynomial time hierarchy.

**Theorem 4** *CV is* **NP***-hard.*

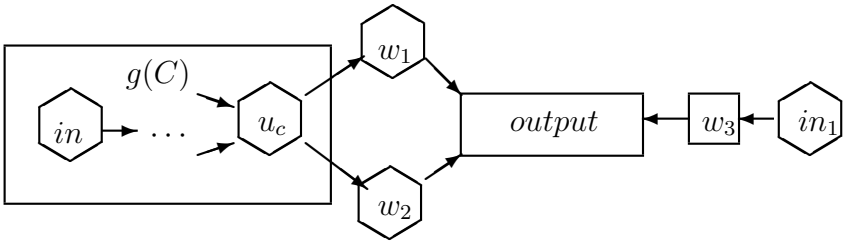


Figure 3: The AND/OR graph $g_1(C)$

**Proof.** It suffices to reduce the **NP**-complete problem $3SAT$ to CV. For each input $C$ to $3SAT$, we construct an AND/OR graph $g_1(C)$ as in Figure 3, where the box $g(C)$ represents the AND/OR graph $g(C)$ without the output vertex *output* constructed in Theorem 3 (that is, the AND/OR graph $g(C)$ in Figure 1). Define the capacity function $c$ for $g_1(C)$ by letting $c(e) = 1$ for all edges $e \in E_{g_1(C)}$. And let $U = \{u_c\}$.

It is clear that $F_{c_U}(g_1(C)) = 1$. And if we let $U' = \{w_3\}$, then a similar argument as in the proof of Theorem 3 shows that $C$ is satisfiable if and only if $F_{c_{U'}}(g_1(C)) = 2$, and $C$ is not satisfiable if and only if $F_{c_{U'}}(g_1(C)) = 0$.

Thus, if $U$ is a set of critical vertices in $g_1(C)$, then $C$ is satisfiable. Since if $C$ is not satisfiable, then $F_{c_{U'}}(g_1(C)) = 0 < F_{c_U}(g_1(C)) = 1$ and $U$ is not a set of critical vertices.

Conversely, assume that $C$ is satisfiable, then it is straightforward that $U$ is a set of critical vertices. This completes our proof of the theorem. □.

**Theorem 5** *CV is co***NP***-hard.*

**Proof.** It suffices to reduce the co**NP**-complete problem $\overline{3SAT}$ to CV. For each input $C$ to $\overline{3SAT}$, let $g_1(C)$ be the AND/OR graph constructed in Theorem 4 (see Figure 3), define the capacity function $c$ for $g_1(C)$ by letting $c(e) = 1$ for all edges $e \in E_{g_1(C)}$. And let $U = \{w_3\}$.

Now a similar argument as in the proof of Theorem 3 shows that $C$ is satisfiable if and only if $F_{c_U}(g_1(C)) = 2$, and $C$ is not satisfiable if and only if $F_{c_U}(g_1(C)) = 0$.

Thus, if $U$ is a set of critical vertices in $g_1(C)$, then $F_{c_{U'}}(g_1(C)) = 1 \geq F_{c_U}(g_1(C))$, where $U' = \{u_c\}$. Whence $F_{c_U}(g_1(C)) = 0$ and $C$ is not satisfiable.

Conversely, assume that $C$ is not satisfiable, then it is straightforward that $U$ is a set of critical vertices. This completes our proof of the theorem. □.

**Corollary 1** *If* **P**≠**NP**, *then CV belongs neither to* **NP** *nor to co***NP**.

**Proof.** This follows from Theorems 4 and 5. □.

Applications of critical vertices are varied. For example, in order to attack a computation system modeled by an AND/OR graph with the least resource, an adversary wants to choose a minimal set of critical vertices to corrupt (e.g., to bomb). And in order for a system designer to make the computation system dependable, he should pay more attention to the processors corresponding to the critical vertices.

It is often the case that a system designer wants to know how many faults a system can tolerate, that is, he is interested in the following problem CVB.

CVB (i.e., Critical Vertices with a given Bound).
*Instance:* An AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$ with a capacity function $c$, two positive integers $k$ and $p$.
*Question:* Does there exist a vertex set $U \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$ in $G$ such that $|U| \leq k$ and $F_{c_U}(G) \leq p$?

It is clear that $F_{c_U}(G) \leq p$ if and only if for all flow $f_U$ with respect to the capacity function $c_U$ we have $F_{f_U}(G) \leq p$. Whence CVB belongs to the complexity class $\Sigma_2^p$. In the following we will show that CVB is both **NP**-hard and co**NP**-hard. We first introduce a restricted version SCV of CVB and show that the problem SCV is **NP**-complete. Given an AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$, a set $U \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$ is called a set of *strictly critical vertices* of $G$ if, for any solution graph $P$ in $G$, $P$ passes through at least one vertex of $U$. Note that a set of strictly critical vertices is different from a vertex separator (though related) defined in [3].

SCV (i.e., Strictly Critical Vertices).
*Instance:* An AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$ and a positive integer $k \leq |(V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})|$.
*Question:* Does there exist a size $k$ set of strictly critical vertices?

**Theorem 6** *SCV is* **NP***-complete.*

**Proof.** We first show that SCV $\in$ **NP**. It suffices to show that there is a polynomial time algorithm to decide whether a given set $U$ of vertices is a set of strictly critical vertices.

For a given set $U$ of vertices, we define another vertex set $U'$ by the following procedure. For the reason of simplicity, in the following we will denote by $outgoing(U')$ the set of all outgoing edges of vertices in $U'$.

1. Let $U' = U$.
2. For each vertex $v \in V_\vee \setminus U'$ such that all incoming edges of $v$ are in $outgoing(U')$, let $U' = U' \cup \{v\}$. Go to Step 3.
3. If there is a vertex $v \in V_\wedge \setminus U'$ such that at least one of the incoming edges of $v$ is in $outgoing(U')$, then let $U' = U' \cup \{v\}$ and go to Step 2, otherwise output $U'$ and halt.

It is straightforward to check that $U$ is a set of strictly critical vertices if and only if the following condition does not hold:

- There is a sequence of vertices $v_1, \ldots, v_n \notin U'$ such that $v_1 \in INPUT, v_n = output$, and $(v_i \rightarrow v_{i+1}) \in E_P$ for each $i < n$.

This shows that SCV $\in$ **NP**. Whence it suffices to reduce the following **NP**-complete problem VC to SCV.

VC (i.e., Vertex Cover problem).
*Instance:* A graph $G(V, E)$ and a positive integer $k$.
*Question:* Does there exist a vertex cover $V'$ of $G$ with $|V'| \leq k$, that is, a vertex set $V' \subseteq V$ such that any edge in $E$ is incident to at least one vertex in $V'$?

The input $G = (V_G, E_G)$, to VC, consists of a set of vertices $V_G = \{v_1, \ldots, v_n\}$ and a set of edges $E_G = \{e_1, \ldots, e_m\}$ . In the following we construct an AND/OR graph $MG(V_\wedge, V_\vee, INPUT, output; E)$ such that there is a vertex cover of size $k$ in $G$ if and only if there is a size $k$ set of critical vertices in $MG$.

Let $INPUT = \{I_i : i = 1, \ldots n\}$, $V_\vee = \{output\}$, $V_\wedge = INPUT \cup \{u_i : i = 1, \ldots, n\} \cup \{w_i : i = 1, \ldots, m\}$, and $E$ be the set of the following edges.

1. For each $i = 1, \ldots, n$, there is an edge $I_i \rightarrow u_i$.
2. For each $i = 1, \ldots, m$, there are two edges $u_{i_1} \rightarrow w_i$ and $u_{i_2} \rightarrow w_i$, where $e_i = (v_{i_1}, v_{i_2})$
3. For each $i = 1, \ldots, m$, there is an edge $w_i \rightarrow output$.

Now assume that $\{v_{i_1}, \ldots, v_{i_k}\}$ is a vertex cover in $G$, then it is straightforward that $\{u_{i_1}, \ldots, u_{i_k}\}$ is a set of critical vertices in $MG$.

Conversely, assume that $U$ is a set of critical vertices in $MG$. Let $U' \subseteq V_G$ be a set of vertices defined by the following procedure.

1. Let $U' = \emptyset$.
2. For each $i \leq n$ such that $u_i \in U$, let $U' = U' \cup \{v_i\}$.
3. For each $i \leq m$ such that $w_i \in U$, let $U' = U' \cup \{v_j\}$, where $v_j$ is any vertex in $V_G$ which is incident to $e_i$.

And it is easy to see that $U'$ is a vertex cover in $G$. □.

Actually, the proof of Theorem 6 also shows that the problem CVB is **NP**-hard.

**Theorem 7** *CVB is* **NP***-hard.*

**Proof.** Let $G$ and $MG$ be the graph and the AND/OR graph in the proof of Theorem 6. Then it is straightforward to check that there is a vertex cover of size $k$ in $G$ if and only if there is a set $U \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$ such that $|U| \leq k$ and $F_{c_U}(MG) = 0$. This completes our proof of the theorem. □.

**Theorem 8** *CVB is co***NP***-hard.*

**Proof.** It suffices to reduce the co**NP**-complete problem $\overline{3SAT}$ to CV. For each input $C$ to $\overline{3SAT}$, let $g_1(C)$ be the AND/OR graph constructed in Theorem 4 (see Figure 3). Define a capacity function $c$ for $g_1(C)$ by letting $c(e) = 1$ for all edges $e \in E_{g_1(C)}$, and let $k = 1$ and $p = 0$.

Now a similar argument as in the proof of Theorem 5 shows that the following conditions are equivalent.

1. $C$ is not satisfiable.
2. There exists a vertex set $U \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$ in $g_1(C)$ such that $|U| \leq 1$ and $F_{c_U}(g_1(G)) \leq 0$ (that is, $U = \{w_3\}$ satisfies this property).

This completes our proof of the theorem. □.

**Corollary 2** *If* **P**≠**NP***, then CVB belongs neither to* **NP** *nor to co***NP***.*

**Proof.** This follows from Theorems 7 and 8. □.

The complexity of problems can be compared by reductions. If one problem can be reduced with respect to some reduction to another problem, then the latter problem has at least the complexity of the former problem, and the comparability increases when the reduction becomes more restrictive. As an example, in the polynomial time Turing reduction, an **NP**-complete optimization problem can be reduced to the corresponding decision problem while this *might* not be possible in the polynomial time many one reduction. Up to now in this paper, we have only considered polynomial time many one reductions. We close this section by showing that $\overline{\text{CV}}$ can be reduced to CVB by a polynomial time Turing reduction (we do not know whether there is a polynomial time many one reduction between them).

**Theorem 9** *There is a polynomial time Turing reduction from* $\overline{\text{CV}}$ *to CVB.*

**Proof.** The input to $\overline{\text{CV}}$ consists of an AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$ with a capacity function $c$, and a vertex set $U \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$. By Theorem 7, CVB is **NP**-hard, whence using a standard polynomial time Turing reduction from an optimization problem to its corresponding decision problem, we can compute in polynomial time the value of $F_{c_U}(G)$ by an adaptive oracle access to CVB. Now we distinguish the following two cases:

1. $F_{c_U}(G) = 0$. Then $U$ is a set of critical vertices.
2. $F_{c_U}(G) \geq 1$. Then $U$ is not a set of critical vertices if and only if there is a set of vertices $U' \subseteq (V_\wedge \cup V_\vee) \setminus (INPUT \cup \{output\})$ with the property $|U'| = |U|$ and $F_{c_{U'}}(G) \leq F_{c_U}(G) - 1$. For this case, with an adaptive access to the oracle CVB, we can decide in polynomial time whether $U$ is a set of critical vertices.

This completes our proof of the theorem. □.

## 5. Maximum additive flows in an AND/OR graph

We have mentioned in Section  that after Sahni [16] proved the **NP**-hardness of finding minimum-cost solution graphs in AND/OR graphs, Martelli and Montanari [12,13] modified the cost structure to be additive and got a (heuristic) polynomial time algorithm AO$^*$. Based on the similar idea, we can make our definition of flows in an AND/OR graph "additive" and then get a polynomial time algorithm for finding maximum additive flows in AND/OR graphs. A careful analysis of the proofs in Section 3 shows that the **NP**-hardness of the problems related to maximum flows in AND/OR graphs is mainly due to the nondeterminism of $\vee$-vertices, that is, each unit flow into an $\vee$-vertex leaves that vertex from only one outgoing edge (cannot leave that vertex from several edges at the same time). Like the definition of the additive cost of edges by Martelli and Montanari [12,13], we can unfold each $\vee$-vertex. That is, let the $\vee$-vertex have the copy function. More precisely, an *additive flow* $f$ in an ADN/OR graph $G$ is a flow function $f$ defined on edges of $G$ with the equation (1) replaced by the following equation (3):

$$f(e') \leq \sum_{e \in v^-} f(e) \tag{3}$$

for all $v \in V_\vee$ and $e' \in v^+$. For an additive flow $f$ in the AND/OR graph $G$, the *total additive flow* $F'_f(G)$ is defined to be the amount of additive flow into the output vertex *output*. And we will use $F'_c(G)$ to denote the maximum of $F'_f(G)$ for all additive flow $f$ in $G$, that is, $F'_c(G) = \max\{F'_f(G) : f \text{ is an additive flow in } G\}$.

Applications of the theory of additive flows in AND/OR graphs are varied. For example, data in computer systems are easy to copy. The additive flow may be interpreted that $\vee$-vertices in an AND/OR graph can "copy" data (note that in some production process, "hardwares" can not be easily "copied", whence can only be modeled by non-additive flows).

**Theorem 10** *There is an efficient algorithm to compute the maximum additive flows in AND/OR graphs.*

**Proof.** For a given AND/OR graph $G$ and a capacity function $c$, we first recursively define a function $f$ on edges of $G$ by the following procedure.

1. Let $f(e) = c(e)$ for all $e \in E$.
2. If $f(e)$ is an additive flow in $G$, then halt, otherwise go to Step 3.
3. For each vertex $v \in V_\vee$ such that the equation (3) does not hold, let $m_v = \sum_{e \in v^-} f(e)$. For each vertex $v \in V_\wedge$ such that the equation (2) does not hold, let $m_v = \min\{f(e) : e \in v^-\}$. And for all other vertices $v$ let $m_v = \infty$.
4. Let $v_0 \in V_\vee \cup V_\wedge$ be the vertex such that $m_{v_0}$ (defined in Step 3) is a minimum (resolve ties arbitrarily).
5. We distinguish the following two cases.
   - $v_0 \in V_\vee$. For each $e \in v_0^+$, let $f(e) = \min\{f(e), m_{v_0}\}$. Go to Step 2.
   - $v_0 \in V_\wedge$. For each $e \in v_0^+ \cup v_0^-$, let $f(e) = \min\{f(e), m_{v_0}\}$. Go to Step 2.

By the choice of $v_0$ in Step 4, it is easy to see that for each edge $e$, the values of $f(e)$ will change for at most once. Whence the above algorithm will stop in at most $(|E| + |V_\vee| + |V_\wedge|)^2$ steps with an additive flow function $f$. By the construction of the additive flow $f$, it is straightforward to check that for any other additive flow $f'$, we have $f'(e) \leq f(e)$ for all $e \in E$. It follows that $F'_c(G) = \sum_{e \in output^-} f(e)$. This completes our proof of the Theorem. □.

**Corollary 3** *For an AND/OR graph $G$ with only one $\vee$-vertex and a capacity function $c$, there is a polynomial time algorithm for finding the maximum (non-additive) flow in it.*

**Proof.** It is straightforward to check that, for an AND/OR graph $G$ with only one $\vee$-vertex and a capacity function $c$, the maximum additive flow and the maximum non-additive flow coincide. Whence the corollary follows from Theorem 10. □.

As in Section 4, we can also define the set of critical vertices for additive flows and we denote by A-CV, A-SCV and A-CVB the corresponding versions of CV, SCV and CVB respectively. Since SCV = A-SCV and there is a polynomial time algorithm to compute the maximum additive flow in an AND/OR graph, we have the following theorem.

**Theorem 11** *Both A-SCV and A-CVB are* **NP***-complete, and A-CV* $\in$ *co***NP**.

**Proof.** It follows directly from Theorem 10 and the definitions of these complexity classes that both A-SCV and A-CVB belong to **NP**, and A-CV belongs to co**NP**.

Since SCV = A-SCV, by Theorem 6, A-SCV is **NP**-complete. It is straightforward to check that the proof of Theorem 7 also shows that A-CVB is **NP**-hard. Whence A-CVB is **NP**-complete. □.

We close this section by showing that $\overline{\text{A-CV}}$ is polynomial time Turing complete for **NP**. Note that we still do not know whether $\overline{\text{A-CV}}$ is **NP**-complete, that is, whether $\overline{\text{A-CV}}$ is polynomial time many one complete for **NP**.

**Theorem 12** *$\overline{\text{A-CV}}$ is polynomial time Turing complete for* **NP**.

**Proof.** By Theorem 11, $\overline{\text{A-CV}} \in$ **NP**, whence it is sufficient to reduce the **NP**-complete problem VC to $\overline{\text{A-CV}}$ by a polynomial time Turing reduction.

Given an input graph $G(V_G, E_G)$ to VC, let $MG$ be the AND/OR graph constructed in Theorem 6. Define a capacity function $c$ by letting $c(e) = 1$ for all $e \in E$. For any vertex set $U \subseteq (V_\vee \cup V_\wedge) \setminus (INPUT \cup \{output\})$, let $f(U)$ be a set of vertices defined by the following procedure:

1. Let $f(U) = \emptyset$.
2. For each $i \leq n$ such that $u_i \in U$, let $f(U) = f(U) \cup \{u_i\}$.
3. For each $i \leq m$ such that $w_i \in U$, let $f(U) = f(U) \cup \{u_j\}$, where $j$ is a number such that $v_j$ is any vertex in $V_G$ which is incident to $e_i$.

It is straightforward to check that $F'_{c_{f(U)}}(MG) \leq F'_{c_U}(MG)$. Now using the following procedure we can decide in polynomial time whether $G(V_G, E_G)$ has a vertex cover of size $k$.

1. Pick up any vertex set $U \subseteq (V_\vee \cup V_\wedge) \setminus (INPUT \cup \{output\})$ with $|U| = k$.
2. With the help of the oracle $\overline{\text{A-CV}}$, decide whether $f(U)$ is a set of critical vertex with respect to the additive flow in $MG$, if the answer is *yes*, then go to Step 3, otherwise go to Step 4.
3. If $F'_{c_{f(U)}}(MG) = 0$, then $G$ has a vertex cover $\{v_i : u_i \in f(U)\}$ of size $k$, otherwise $G$ does not have such a vertex cover. Halt.
4. Use a standard polynomial time Turing reduction from an optimization problem to its corresponding decision problem to compute a size $k$ vertex set $U' \subseteq (V_\vee \cup V_\wedge) \setminus (INPUT \cup \{output\})$ such that $F'_{c_{U'}}(MG) < F'_{c_{f(U)}}(MG)$. Let $U = U'$ and go to Step 2.

This completes our proof of the Theorem. □.

## 6. Comments and open problems

In this paper, we have discussed the problem of finding maximum flows and the problem of finding critical vertices in an AND/OR graph. In particular, we showed the hardness of several problems and compared their complexity. It should be noted that though we stated our theorem for general AND/OR graphs, it is straightforward to check that all our results hold for acyclic AND/OR graphs. As a summary, Figure 4 describes the relationship among the problems we have discussed graphically. The arrows describes the knowledge of the existence of polynomial time many one reductions.

The following interesting problems remain open yet.

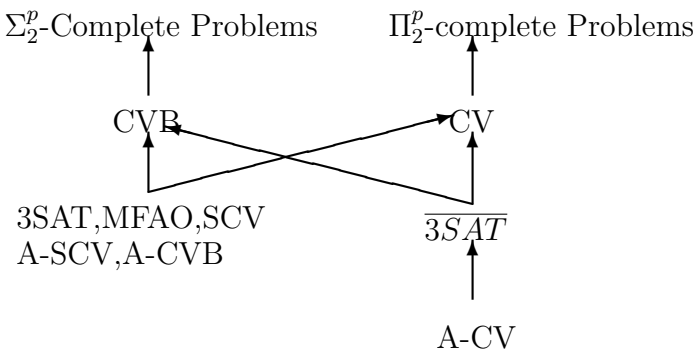


Figure 4: The relationships

1. Is CVB $\Sigma_2^p$-complete?
2. Can CVB be polynomial time Turing reduced to $\overline{\text{CV}}$?
3. Is CV $\Pi_2^p$-complete?
4. Is A-CV co**NP**-complete?
5. Can CVB be polynomial time Turing reduced to $3SAT$?

We conjecture that the answers to the above questions are all negative unless the polynomial time hierarchy collapses. In addition, it is interesting to show the exact relationship between $\overline{\text{CV}}$ and CVB?

## References

1. A. Bagchi and A. Mahanti. Admissible heuristic search in AND/OR graphs. *Theoret. Comput. Sci.,* **24**:207–219, 1983.
2. D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, Inc., 1992.
3. M. Burmester, Y. Desmedt, and Y. Wang. Using approximation hardness to achieve dependable computation. In: *Proc. the 2nd International Conference on Randomization and Approximation Techniques in Computer Science, RANDOM '98*. Lecture Notes in Computer Science, Springer Verlag, 1998.
4. P. Chakrabarti, S. Ghose, S. DeSarkar. Admissibility of AO* when heuristics overestimate. *Artificial Intelligence*, **34**:97–113, 1988.
5. C. L. Chang and J. R. Slagle. An admissible and optimal algorithm for searching AND/OR graphs. *Artificial Intelligence*, **2**:117–128, 1971.
6. S. Even. *Graph Algorithms*. Computer Science Press, 1979.
7. L.R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of* **NP***-Completeness*. W. H. Freeman and Company, San Francisco, 1979.
9. M. Gondran and M. Minoux. *Graphs and Algorithms*. John Wiley & Sons Ltd., New York, 1984.
10. D. Hvalica. Best first search algorithm in AND/OR graphs with cycles. *J. of Algorithms*, **21**:102–110, 1996.

11. A. Mahanti and A. Bagchi. AND/OR graph heuristic search methods. *J. Assoc. Comput. Mach.*, **32**(1):28–51, 1985.
12. A. Martelli and U. Montanari. Additive AND/OR graphs. In *Proceedings of the Third International Joint Conference on Artificial Intelligence,* pages 1–11, Morgan Kaufmann Publishers, Inc., 1973.
13. A. Martelli and U. Montanari. Optimizing decision trees through heuristically guided search. *Comm. of the Assoc. Comput. Mach.*, **21**(12):1025–1039, 1978.
14. N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.
15. J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.
16. S. Sahni. Computationally related problems. *SIAM J. Comput.*, **3**:262–279, 1974.
17. L. Stockmeyer. The polynomial time hierarchy. *Theoret. Comput. Sci.*, **3**:1–22, 1977.
18. M. Swamy and K. Thulasiraman. *Graphs, Networks, and Algorithms*. John Wiley & Sons Ltd., New York, 1981.
19. Y. Wang, Y. Desmedt, and M. Burmester. **NP**-Hardness of dependable computation with multiple inputs. *Fundamenta Informaticae*, **42**(1):61–73, 2000.