STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XXXIX, 3, 1994

# COMPLEXITY METRICS FOR DISTRIBUTED PROGRAMS

**Monica CIACA***

Dedicated to Professor Sever Groze on his 65th anniversary

*Received: March 15, 1995*
*AMS subject classification: 68Q15, 65Y05*

**REZUMAT. - Metrici ale complexităţii programelor distribuite.** În lucrare se propun câteva metrici ale complexităţii programelor distribuite, metrici bazate pe diversele dependenţe de execuţie ce apar la nivelul unui program. Fiind bazate pe mai multe tipuri de dependenţă, metricile pot exprima complexitatea unui program distribuit din mai multe puncte de vedere.

## 1. Introduction

Metrics for measuring software complexity have many applications in software engineering activities including analysis, testing, debugging, and maintenance of programs, and management of project.

When we intend to measure some attributes of some entities, we must be able to capture information about the attributes and therefore we must have some representation and model of the entities such that the attributes can be explicitly described in the representation or the model. To measure the complexity of a concurrent program we must capture information about not only control structure and data flow in every process but also syncronization structure and interprocess communication among processes.

This paper presents some graph theoretical representations for distributed programs and defines some metrics for measuring complexity of distributed programs.

* *"Babeş-Bolyai" University, Faculty of Economics, 3400 Cluj-Napoca, Romania*

## 2. Definitions

A *nondeterministic parallel control flow net* (CFN) is a 10-tuple $(V,N,P_F,P_J,A_C,A_N,A_{PF},A_{PJ},s,t)$ where $(V,A_C,A_N,A_{PF},A_{PJ})$ is a simple arc-classified digraph such that $A_C \subseteq V \times V$, $A_N \subseteq N \times V$, $A_{PF} \subseteq P_F \times V$, $A_{PJ} \subseteq V \times P_J$, $N \subset V$ is a set of elements, called *nondeterministic selection vertices*, $P_F \subset V$ $(N \cap P_F = \Phi)$ is a set of elements, called *parallel execution fork vertices*, $P_J \subset V$ $(N \cap P_J = \Phi$, and $P_F \cap P_J = \Phi)$ is a set of elements, called *parallel execution join vertices*, $s \in V$ is a unique vertex, called *start vertex*, such that in-degree(s)=0, $t \in V$ is a unique vertex, called *termination vertex*, such that out-degree(t)=0 and $t \neq s$, and for any $v \in V$ $(v \neq s, v \neq t)$, there exists at least one path from s to v and at least one path from v to t. Any arc $(v_1,v_2) \in A_C$ is called *sequential control arc*, any arc $(v_1,v_2) \in A_N$ is called *nondeterministic selection arc*, and any arc $(v_1,v_2) \in A_{PF} \cup A_{PJ}$ is called *parallel execution arc*.

A usual (deterministic and sequential) control-flow graph can be regarded as a special case of nondeterministic parallel control-flow nets such that $N$, $P_F$, $P_J$, $A_N$, $A_{PF}$, and $A_{PJ}$ are the empty set.

A *nondeterministic parallel definition-use net* (DUN) is a 7-tuple $(N_C, \Sigma_V, D, U, \Sigma_C, S, R)$, where $N_C=(V,N,P_F,P_J,A_C,A_N,A_{PF},A_{PJ},s, t)$ is a control flow-net, $\Sigma_V$ is a finite set of symbols, called *variables*, $D:V \to P(\Sigma_V)$ and $P:V \to P(\Sigma_V)$ are two partial functions from V to the power set of $\Sigma_V$, $\Sigma_C$ is a finite set of symbols, called *channels*, and $S:V \to \Sigma_C$ and $S:V \to \Sigma_C$ are two partial functions from V to $\Sigma_C$.

A DUN can be regarded as a CFN with the information concerning definitions and uses of variables and communication channels. A usual (deterministic and sequential) definition-use graph can be regarded as a special case of nondeterministic parallel definition-use nets such that $N,P_F,P_J,A_C,A_N,A_{PF},A_{PJ},\Sigma_C,S$ and R are the empty set.

The above definitions of CFN and DUN are graph-theoretical, and therefore, they are independent of any programming language.

### 3. Process dependence net

Program dependences are dependence relationships holding between statements in a program [Ferrante91]. Based on the DUN of a distributed program, we can formally define five kinds of primary program dependences: *control dependence, data dependence, selection dependence, synchronization dependence, and communication dependence* [Cheng92], [Cheng93].

Informally, a statement u is directly control-dependent on the control predicate v of a conditional branch statement (e.g., an if statement or while statement) if whether u is executed or not is directly determinated by the evaluation result of v; a statement u is directly data-dependent on a statement v if the value of a variable computed at v has a direct influence on the value of a variable computed at u; a statement u is directly selection-dependent on a nondeterministic selection statement v if whether u is executed or not is directly determined by the selection result of v; a statement u is directly synchronization-dependent on another statement v if the start and/or termination of execution of v directly determines whether or not the execution of u starts and/or terminates; a statement u in a process is directly communication-dependent on another statement v in another process if the value of a variable computed at v has a direct influence on the value of a variable computed at u by an interprocess communication.

If we represent all five kinds of primary program dependences in a distributed program within an arc-classified digraph such that each type of arcs represents a kind of primary

program dependences, then we can obtain an explicit dependence-based representation of the program. Such a representation is called process dependence net [Cheng92], [Cheng93].

The *process dependence net* (PDN) of a program is an arc-classified digraph **(V,Con,Sel,Syn,Com)** where **V** is the vertex set of the CFN of the program, **Con** is the set of control dependence arcs such that any **(u,v)∈Con** if and only if u is directly weakly control-dependent on v, **Sel** is the set of selection dependence arcs such that any **(u,v)∈Sel** if and only if u is directly selection-dependent on v, **Dat** is the set of data dependence arcs such that any **(u,v)∈Dat** if and only if u is directly data-dependent on v, **Syn** is the set of synchronization dependence arcs such that any **(u,v)∈Syn** if and only if u is directly synchronization-dependent on v and **Com** is the set of communication dependence arcs such that any **(u,v)∈Com** if u is directly communication-dependent on v.

## 4. Dependence-based complexity metrics

We will further use the following notations of relational algebra:

$\mathbf{R}^+$ : the transitive closure of binary relation **R**.

$\sigma_{[1]=v}(\mathbf{R})$ : the selection of binary relation **R** such that $\sigma_{[1]=v}(\mathbf{R})=\{(v_1,v_2)|(v_1,v_2)\in\mathbf{R} \text{ and } v_1=v\}$.

$\sigma_{[1]\in S}(\mathbf{R})$ : the selection of binary relation **R** such that $\sigma_{[1]\in S}(\mathbf{R})=\{(v_1,v_2)|(v_1,v_2)\in\mathbf{R} \text{ and } v_1\in S\}$.

$\sigma_{[2]\in S}(\mathbf{R})$ : the selection of binary relation **R** such that $\sigma_{[2]\in S}(\mathbf{R})=\{(v_1,v_2)|(v_1,v_2)\in\mathbf{R} \text{ and } v_2\in S\}$.

Based on the **PDN=(V,Con,Sel,Dat,Com)** of a distributed program, we can define the following dependence-metrics for measuring the complexity of the program, where $|A|$ is the cardinality of set A, $D_p\in\{\mathbf{Con,Sel,Dat,Syn,Com}\}$, $D_U=\mathbf{Con}\cup\mathbf{Sel}\cup\mathbf{Dat}\cup\mathbf{Syn}\cup\mathbf{Com}$, and P is the set of all statements of a process p:

$|D_P|/|D_U|$ is the proportion of a special primary program dependence to all primary program dependences in a program, and therefore, it can be used to measure the degree of concurrency of the program for a special viewpoint. For example, the larger are $|Con|/|D_U|$ and $|Dat|/|D_U|$ of a program, the less concurrent is the program. The larger is $|Sel|/|D_U|$ of a program, the more nondeterministic is the program. The larger are $|Syn|/|D_U|$ and $|Com|/|D_U|$ of a program, the more concurrent is the program.

$|Sel \cup Syn \cup Com|/|D_U|$ is the proportion of those primary program dependences concerning concurrency to all primary program dependences in a program, and therefore, it can be used to measure the degree of concurrency of the program from a general viewpoint.

$|D_P^+|/|D_U^+|$ is a metric different from $|D_P|/|D_U|$ in that $|D_P|/|D_U|$ only concerns direct dependences while $|D_P^+|/|D_U^+|$ concerns not only direct dependences but also indirect dependences.

$|(Sel \cup Syn \cup Com)^+|/|D_U^+|$ : the difference between this metric and $|Sel \cup Syn \cup Com|/|D_U|$ is similar to the difference between $|D_P|/|D_U|$ and $|D_P^+|/|D_U^+|$.

$max/min\{|\sigma_{[i]\sim v}(D_P)|v \in V\}/|V|$ expresses the proportions of the maximal/minimal number of statements on which a statement is directly control, data, nondeterministic selection, synchronization, or communication dependent, respectively to the total number of statements in a program. The larger is the metric of a program, the more complex is the program. In particular, the larger is $max/min\{|\sigma_{[i]\sim v}(D_{CP})|v \in V\}/|V|$, where $D_{PC} \in \{Sel, Syn,$

Com}, of a program, the more complex is the concurrency in the program.

**max/min**$\{|\sigma_{[i]=v}(D_U)|v\in V\}/|V|$ expresses the proportions of the maximal/minimal number of statements on which a statement is somehow directly dependent, to the total number of statements in a program.

**max/min**$\{|\sigma_{[i]=v}(D_P^+)|v\in V\}/|V|$ expresses the proportions of the maximal/minimal number of statements on which a statement is directly and indirectly control, data, nondeterministic selection, synchronization, or communication dependent, respectively, to the total number of statements in a program.

**max/min**$\{|\sigma_{[i]=v}(D_U^+)|v\in V\}/|V|$ expresses the proportions of the maximal/minimal number of statements, on which a statement is somehow directly and indirectly dependent, to the total number of statements in a program.

The following metrics can be used to measure the degree of interaction among processes in a program. The larger is the maximal/minimal value of such a metric of a program, the more complex is the concurrency in the program:

$|\sigma_{[i]\in P}(Syn\cup Com)|/|V|$ is the proportion of the number of statements of other processes on which a process is directly dependent, to the total number of statements in a program.

$|\sigma_{[1]\in P}(Syn \cup Com)^+|/|V|$ is the proportion of the number of statements of other processes, which a process is directly and indirectly dependent on, to the total number of statements in a program.

$|\sigma_{[1]\in P1}(Syn \cup Com) \cap \sigma_{[2]\in P2}(Syn \cup Com)|/|V|$ is the proportion of the number of statements such that two processes directly depends on each other, to the total number of statements in a program.

$|\sigma_{[1]\in P1}(Syn \cup Com)^+ \cap \sigma_{[2]\in P2}(Syn \cup Com)^+|/|V|$ is the proportion of the number of statements such that two processes directly and indirectly depends on each other, to the total number of statements in a program.

Some of the above dependence-based complexity metrics can be used to measure the concurrency complexity of a distributed program in various aspects and some of them can be used to measure overall complexity of the program. Some other complexity metrics similar to the above metrics can also be considered. For a further discussion we refer the reader to [Conte86] and [Fenton91].

## REFERENCES

**[Cheng92]** J.Cheng - Task Dependence Net as a Representation for Concurrent ADA Programs, in J. van Katwijk (ed.) "Ada: Moving towards 2000", *Lecture Notes in Computer Science*, Vol.603, Springer-Verlag, June 1992, pp.150-164.

**[Cheng93]** J.Cheng - Slicing Concurrent Programs, *Proc 1st International Workshop on Automated and Algorithmic Debugging*, May 1993.

**[Conte86]** S.D.Conte, H.E.Dunsmore and V.Y.Shen - *Software Engineering Metrics and Models*, Benjamin/Cummings, 1986, 396 p.

**[Fenton91]** N.E.Fenton - *Software Metrics: A Rigorous Approach*, Chapman&Hall, 1991, 337p.

**[Ferrante87]** J. Ferrante, K.J. Ottenstein and J.D. Warren - The Program Dependence Graph and its use in Optimization, *ACM TOPLAS*, Vol.9, No.3, July 1987, pp.319-349.