On the Configuration of Optimization Algorithms by Using XML Files

E. Alba, J.M. García-Nieto, A.J. Nebro

12th March 2003

Abstract. There exist different ways of controlling and configuring an algorithm. One way is to hard-code the values of its parameters in the program implementing it. This approach is quite inflexible because we need to recompile the program when we want to change the value of a parameter. Thus, most programmers prefer to use another solution, consisting in defining configuration files which contain the values governing the behavior of the algorithm. However, many programmers use ASCII documents in such a way that the parameters and their values are written using ad-hoc rules, which lead to non-standard, difficult to understand and error prone information. In this paper we discuss different approaches to configure an algorithm by means of a text file. This is by no means a minor issue, since a proper configuration could lead to standardization, safety and algorithm interaction (e.g. cooperation). We raise this discussion in the context of optimization algorithms and in the aim at developing a web optimization service. Such proposal must accommodate heterogeneous algorithms, that should be configured with our proposal by using XML files following a common DTD document.

Key Words. Configuration Files, Optimization and Search Algorithms, XML, DTD, Internet.

1 Introduction

When faced to the problem of controlling an algorithm[1], an option is to use configuration files to define the parameters and the techniques that the algorithm must apply during its execution; typically, these files are read by the algorithm when it starts its execution. The other choice consists in including the parameter values inside the own code of the algorithm. This second approach has obvious problems when changing values, because recompilation is required, which leads to a waste of time. However, this second scheme is used in some parallel environments to avoid access to a common configuration file through the network.

In this paper we address the first approach, namely configuring an algorithm by defining values in a configuration file. This method has the advantage of not needing recompilation, and it is easier for users to set the parameters through, for example, a simple text editor. The access to the configuration file is not considered a problem and is thought to be granted for all the processes.

The motivation of this work is related to our interest in achieving an ambitious goal: we want to develop an optimization web service that incorporates different *existing* algorithms in such a way that a user can navigate using a web browser through all them, select the algorithm, set the parameters and run the algorithm from a remote computer in order to solve an optimization problem. Take a look to the simplified vision of the architecture in Figure 1. We can notice in this figure that we define a hierarchy of machines, namely the client side, the server side and the workers side. The final user is sit to the client side and selects the algorithm and parameters to solve the selected problem. The user also attaches a file defining the optimization problem in a preferred programming language (assuming that there is at least one algorithm written in this language in the worker side). The workers are in charge for the execution of the algorithm by fol-

[1]For the sake of simplicity, we use the term "algorithm" to refer to the algorithm itself as well as to the program implementing it.

lowing the parameters chosen by the user. There exist some servers allowing a dynamic addition of workers and algorithms to the whole system, and also a primary server to have a minimum Internet infrastructure to guide and execute algorithms considered indispensable for the whole system. In order to create such a service, we are forced to deal with existing algorithms, most of them having different ways of being configured. This is the reason for using an extensible way of configuring and exchanging information inside the system.

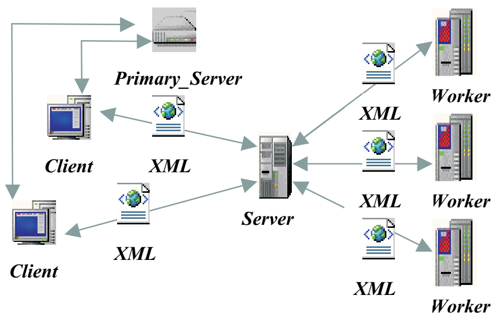


Figure 1: Simplified architecture of our target optimization service in Internet.

Thus, we are facing a twofold objective: first, dealing with different configuration files that already exist, and, second, the need of defining a "standard" for configuration files. The first goal can be accomplished by developing some kind of *wrapper* to adapt the existing file to the standard definitions of the optimization web service we want to build. The difficulty of this task depends on the structure of the concrete file and how it can processed to fit into the requirements imposed by the web service. To achieve the second goal, i.e, defining a standard, we have necessarily to define the context of the standard, since a clear solution exists in Internet: XML (the eXtended Markup Language) [1]. Our context is "optimization".

In this work we begin by discussing different possibilities for dealing with plain ASCII configuration files. As mentioned before, if we use them in our future web optimization system, we will need a wrapper code written in some language such as C, C++ or Java, to be integrated in the system. The second part of this paper will focus on proposing a concrete *Data Type Document* or DTD that would serve as our "standard" for defining future compatible configuration files. This will clearly remove the necessity of using wrappers as long as the algorithms conform to this standard. The highly promising extensions and applications of such an optimization system configured and manipulated through XML would have a large impact when put in contact to other XML related standards, such us *XML-RPC* and *SOAP* [2]. These new standards include a APIs for manipulating XML files, and they would permit our system to be used by a wide audience of potential users, thus having a deep impact in the optimization and search research community.

The structure of this document is as follows. In the following section (Section 2) we address some different approaches to use a configuration file written in ASCII code, without considering XML as the language to be used. In Section 3, we present some general ideas about XML, some related technologies, and the requirements that a web optimization service will impose on the XML files. In Section 4, we discuss our proposal of a concrete DTD which will allow us to define XML files in order to establish the standard behavior for new algorithms. Some explained examples are included in Section 5. Finally, we summarize our contributions, conclusions and future work in Section 6.

2 Non-standard ASCII Configuration Files

In this section we discuss several approaches to configure an algorithm by using ASCII plain files. The problem to solve is how to write the the parameters and/or their values inside a file. There are a number of approaches than can be used. For example, we can write all the values in the file, identifying them using some kind some kind of *separator*. Some examples of separators are blank spaces, tabulators or carriage returns. This configuration assigns to each attribute a fixed position in the file, and it makes no explicit assignment of values to attributes, since only values are placed inside the file. An example in depicted in Figure 2.

Although processing this kind of files is sim-

```
1.0
0.1
if_better
100
EvalFunc.MOD
```

Figure 2: Simple ASCII configuration file. Each line represents a value for an implicit attribute.

ple, it is error-prone, because the missing of a single attribute can lead to assign wrong values to the rest of parameters. These errors can be difficult to detect. Furthermore, the understanding of the parameters can be difficult to the user. A small improvement is to use comments, which can appear in any part of the document, like it is shown in the example of Figure 3. Comments are used to explain individual values, and they can also be used to add some meta-information like author, version, problem or algorithm being used (i.e, the comment style used by the language Java). Comments usually span through one line beginning with a given indicator, such as `//` or `#`. Explanatory comments greatly helps the user, but the document still remains an ad-hoc and a non-standard input mechanism. Reading comments is quite simple, so this approach implies only a little overhead for the programmer and the impact in the efficiency of the algorithm is negligible.

```
5       // number of independent runs
500     // number of evaluations
300     // Markov chain length
0.88    // temperature decay
0       // display state, LAN-configuration
200     // the global state is updated in this number of evaluations
0       // 0: asynchronous mode // 1: synchronous mode
500     // interval of iterations to cooperate (0 if no cooperation)
```

Figure 3: ASCII file with comments.

The clear improvement on all the precedent formats is to include explicitly the name of the parameters and their values in the configuration file, in a similar way as environment variables are set in the UNIX shells. This scheme furnishes the user with a larger freedom to allocate the values in any place of the file (see an example in Fig-

```
##########################################################
## CONFIGURATION FILE : Steady State Genetic Algorithm  ##
## Use the '#' character at the                         ##
## beginning of a line for comments                     ##
##########################################################
##           Parameters of Algorithm                    ##
##########################################################

number_of_genes         =       10
#
gene_length             =        1
#
population_size         =       100
#
crossover_prob          =       0.8
#
mutation_prob           =       0.00123
#
max_n_evals             =       100
#
fitness_function        =       ProblemOneMax.java
#
```

Figure 4: Commented ASCII configuration file with pairs attribute-value.

ure 4). This approach leads to readable configuration files, in which we can also define sections of similar values, not only the attribute names of such values. Including comments in this file is the final requirement to have a flexible, clear and, at the same time, easy to read configuration file.

```
[NUMBER_OF_OPERATORS]
      5
[OPERATORS]
      RW
      RANDOM
      RECOMBINE_ES 1
      MUTA_ES 1.0
      REP_LEAST_FIT
[OUTPUT_DISPLAY]
      NOTHING
[PROBLEM_CONFIG_FILE]
      bc_train.net
[OUTPUT_FILE]
      foo.out
```

Figure 5: Commented ASCII configuration file with pairs variable-value and sections.

Al last, we can use labels, which can be defined like words locked up in brackets, as shown in Figure 5. In this example, the label `[NUMBER_OF_OPERATORS]` indicates that 5 is the number of parameters of the algorithm, and the label `[OUTPUT_DISPLAY]` is used to indicate to the algorithm the kind of verbose information to be printed on the standard output during its execution.

As we use format independent features, comments, and labels, we progress towards configuration files that are more understandable for users and easy to be processed. Our last label example can be considered as a good solution, but a problem still remains cached: it is an ad-hoc mechanism, the sections are problem-dependent, and it is difficult to deduce issues as valid data types or attribute values from the file itself. In next section we make a proposal to fix all these problems using XML.

3 XML to the Rescue

In order to define standard, compatible and flexible configuration files we focus on new Internet technologies. In this context, XML appears as the most promising markup language to be considered for such a goal. XML seems to be the best choice if we want to use a technology with good future perspectives; besides, XML will allow us to configure algorithms as well as to exchange information between them or even inside our target web optimization system. Finally, XML is the gate to use other modern technologies, like SOAP for remote object access and procedure call, as well as other initiatives that can appear in the future, because they will relay probably also on XML.

XML will endow our system with rich features like an improved configuration, readability, connectivity with other systems, on-line cooperation between different algorithms, and access to other emerging Internet technologies. However, using XML will require parsing such files with specialized APIs like DOM or SAX, which requires an extra effort to the programmer and, perhaps, a small overhead when reading the parameters. If XML is used only for configuration, the overhead is negligible; however, if XML is used also for information exchange between the components of a parallel algorithm, then the overhead could be appreciable. In summary, we can state that the advantages clearly outweigh the drawbacks, and hence we will undergo the step of defining an appropriate sub-set of attributes and types for our target optimization/search algorithms.

Many examples of languages developed after XML exist in different fields of application. Some of them are the following ones:

- CDF (*Channel Definition Format*). It defines channels by which to send information periodically.
- CML (*Chemical Markup Language*). Chemical equations and data.
- MathML (*Mathematical Markup Language*). Mathematical equations and expressions.
- OSD (*Open Software Description*). Software packages to install by remote way.
- AML (*Algebraic Modelling Languages*). Specification of algebraic models.

We face this task from the point of view of achieving a light definition of XML specification in the form of a DTD (*Document Type Definition*). In a DTD file we declare the format of the sub-language and the type of the elements, leading to a hierarchy of labels and sub-labels that will hold the information and values to configure or even to run our algorithms. A DTD can be embedded inside modern XML schemas, which are more powerful but more complicated than DTD, and we pursuit an easy scheme to control our algorithms through well conformed XML files.

A more ambitious approach will specify not only the parameters rather the algorithm itself entirely in an XML document. However, we do not want to design a full new language for specifying algorithms, like those mentioned before (CDF, CML, etc); instead, we want to control existing (or new, in the future) algorithms. The desired functionality will account for several target objectives:

- Configure an algorithm, in a uniform way for all the available algorithms.
- Identify the users of the algorithm, in order to interact with them safely.
- Return the results of the algorithms when they finish.
- Define the details about how the algorithm must be compiled, linked and/or run.

Thus, we want a DTD definition capable of supporting information for the algorithms, that allows to differentiate among users to return the results to them, and with the ability of admitting existing algorithms, probably written in different languages on different operating systems. In fact, it is likely that the user send the function to optimize encapsulated in such an XML file that conforms the DTD we are about to propose.

4 DTD Design

A DTD is implemented by following a hierarchic structure, so we begin by dividing our documents in four main information blocks which are the children of a root element called `optimization_algorithm`. In Figure 6, we can see the relationship among elements in a well-conformed XML document.

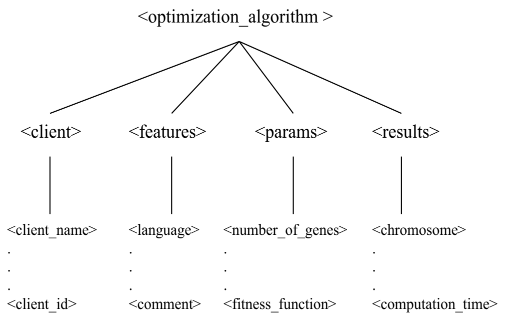


Figure 6: Proposed hierarchy of DTD elements.

From `<optimization_algorithm>` down we give structure to our DTD:

- `<client>`. Information about the client for the system.
- `<features>`. Features of the algorithm to be run.
- `<params>`. Input parameters for the algorithm coming from the client side.
- `<results>`. Results from the algorithm going to the client side.

Within each main element we define a series of sub-elements with their corresponding data types (i.e. labels and contents):

1. `<client>`. It contains elements with information about the client machine and user, and it must appear once in each XML document. We can see these elements in Table 1.

Element	Description
`<client_name>`	Name of the client who uses the service. It must appear once.
`<client_ip>`	IP address of the client machine. It must appear once.
`<client_id>`	Client identifier. It can appear zero or once.

Table 1: Table of `<client>` elements.

2. `<features>`. It contains elements concerning data of the implementation of the algorithms and the commands required for their manipulation. It must appear once in each document. We can see the elements in Table 2.

Element	Description
`<language>`	Programming language in which the algorithm is implemented. It must appear once.
`<compilation>`	Algorithm compilation command. It must appear once.
`<execution>`	Algorithm execution command. It must appear once.
`<online>`	Returned from/during the execution of the algorithm. It can appear once or not.
`<comment>`	Additional comments on the algorithm. It can appear once or not.

Table 2: Table of `<features>` elements.

3. `<params>`. It contains elements with information of the parameters that the algorithms are going to interpret. It must appear once in each document with the added

attribute `<type>` of algorithm. See the list of allowable types of algorithm available in Figure 11). In Table 3 we include the list of sub-elements of the `<params>` element.

Each one of these elements can appear zero or once in the XML document. They can also contain other sub-elements or attributes specified in the DTD, which are shown in Figures 12, 13 and 14.

4. `<results>`. It contains elements that give information of the results after the execution of the algorithms. Their appearance is optional in the XML document. Their elements are included in Table 4, where each element can appear as well zero or once in XML document.

5 Examples

In this section we proceed to show some examples of using the DTD. In Figure 7 we present an XML document defining the parameters for a heuristic genetic algorithm. The reader can distinguish the DTD blocks of information, namely type of the algorithm, input and output parameters, information of the problem to solve and a final block of details about the implementation and running requirements. In the `<features>` section the reader can find the details on its implementation, e.g. the programming language was `Modula 2` for this algorithm.

In Figure 8 we describe an XML document including a section of the result data returned by the execution of a *Steady State Genetic Algorithm* [6]. In the `<features>` section we can notice that the language was Java, as well as the compiling and execution commands to be used for executing the algorithm.

In Figure 9 we show an example evolution strategy having different parameters inside the same block sections that showed in the previous example. We now mark these sections to make them obvious for the reader. The `<features>` section marks the programming language as being `C++` and details the make command and the subsequent execution of the algorithm.

Finally, in Figure 10 we give an example of a simulated annealing heuristic with attributes not

Element	Description
`number_of_genes`	Number of genes in the chromosome
`length_of_gene`	Gene length
`population_size`	Number of individuals
`population_width`	Width of the population if using a grid
`population_height`	Height of the population if using a grid
`population_number`	Number of (sub)populations
`crossover_prob`	Crossover probablility
`mutation_prob`	Mutation probability
`max_n_evals`	Maximum number of evaluations
`migration`	Migration parameters
`fitness_function`	Name of the fitness function code file
`elitism`	Parameters for elitist selection
`replacement`	Replacement policy
`selection`	Parameters of selection
`neighb`	Neighborhood structure
`cost_function`	Name of the cost function
`min_temperature`	Minimum temperature
`max_temperature`	Maximum temperature
`temperature_decay`	% Decay of temperature
`markov_chain _length`	Markov chain length
`lower_bound`	Lower bound
`upper_bound`	Upper bound
`long_term_ memory_length`	Long memory length
`short_term_ memory_length`	Short memory length
`candidate_list_ length`	Candidate list length
`max_distance`	Intermediate distance between two solutions
`strategy`	Node strategy
`others`	It describes other parameters (for miscellaneous elements)

Table 3: Table of `<params>` elements.

Element	Description
allele	Allele
individual	Individuals
chromosome	Chromosome
fitness	Fitness
solution	Solution (generic element)
avg_fitness	Average of fitness
stddev_fitness	Standard deviation of the futness
best_fitness	Best fitness
worst_fitness	Worst fitness
migrations	Number of migrations
computation_time	Computation time
communication_time	Communication time
run_time	Total time of execution
error	Error message

Table 4: Table of `<results>` elements.

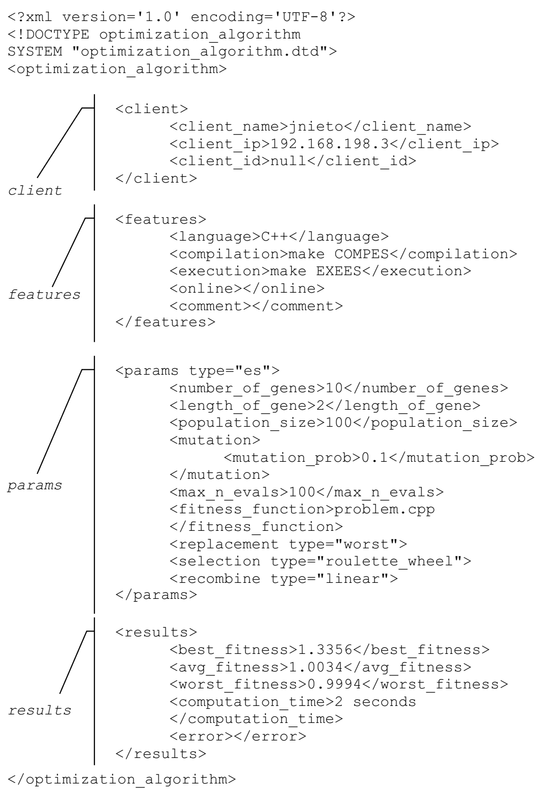
```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE optimization_algorithm
SYSTEM "optimization_algorithm.dtd">
<optimization_algorithm>
      <client>
            <client_name>jnieto</client_name>
            <client_ip>192.168.198.160</client_ip>
            <client_id>null</client_id>
      </client>
      <features>
            <language>Modula 2</language>
            <compilation>compile.bat</compilation>
            <execution>exe_cfxx.bat</execution>
            <online></online>
            <comment></comment>
      </features>
      <params type="cea">
            <number_of_genes>10</number_of_genes>
            <length_of_gene>2</length_of_gene>
            <population_width>6</population_width>
            <population_height>4</population_height>
            <crossover>
                  <crossover_prob>1.0</crossover_prob>
            </crossover>
            <mutation>
                  <mutation_prob>0.1</mutation_prob>
            </mutation>
            <max_n_evals>100</max_n_evals>
            <fitness_function>EvalFunc.MOD
            </fitness_function>
            <replacement type="if_better" />
            <neighb>4</neighb>
      </params>
</optimization_algorithm>
```

Figure 7: XML Document for a *Cellular Genetic Algorithm.*

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE optimzation_algorithm
SYSTEM "optimzation_algorithm.dtd">
<optimzation_algorithm>
      <client>
            <client_name>pmarmol</client_name>
            <client_ip>192.168.198.3</client_ip>
            <client_id>12</client_id>
      </client>
      <features>
            <language>Java</language>
            <compilation>javac SSea.java</compilation>
            <execution>java SSea</execution>
            <online></online>
            <comment></comment>
      </features>
      <params type="ssea">
            <number_of_genes>10</number_of_genes>
            <length_of_gene>2</length_of_gene>
            <population_size>100</population_size>
            <crossover>
                  <crossover_prob>0.8</crossover_prob>
            </crossover>
            <mutation>
                  <mutation_prob>0.00123</mutation_prob>
            </mutation>
            <max_n_evals>1000</max_n_evals>
            <fitness_function>ProblemOneMax.java
            </fitness_function>
      </params>
      <results>
            <chromosome>11111101110001111111</chromosome>
            <fitness>125.1</fitness>
            <computation_time>2 seconds</computation_time>
            <error></error>
      </results>
</optimzation_algorithm>
```

Figure 8: XML Document for a *Steady State Genetic Algorithm.*

already included in the previous XML file, since SA is a different kind of heuristic (not belonging to any family of evolutionary algorithms like it does occur with the preceding examples). Again, similar sections exist. The `<features>` section and `<results>` section contain our final exemplification of the kind of contents that our DTD allows using.



```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE optimization_algorithm
SYSTEM "optimization_algorithm.dtd">
<optimization_algorithm>
      <client>
            <client_name>jnieto</client_name>
            <client_ip>192.168.198.3</client_ip>
            <client_id>null</client_id>
      </client>
      <features>
            <language>C++</language>
            <compilation>make COMPES</compilation>
            <execution>make EXEES</execution>
            <online></online>
            <comment></comment>
      </features>
      <params type="es">
            <number_of_genes>10</number_of_genes>
            <length_of_gene>2</length_of_gene>
            <population_size>100</population_size>
            <mutation>
                  <mutation_prob>0.1</mutation_prob>
            </mutation>
            <max_n_evals>100</max_n_evals>
            <fitness_function>problem.cpp
            </fitness_function>
            <replacement type="worst">
            <selection type="roulette_wheel">
            <recombine type="linear">
      </params>
      <results>
            <best_fitness>1.3356</best_fitness>
            <avg_fitness>1.0034</avg_fitness>
            <worst_fitness>0.9994</worst_fitness>
            <computation_time>2 seconds
            </computation_time>
            <error></error>
      </results>
</optimization_algorithm>
```

Figure 9: XML Document for an *Evolution Strategy*.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE optimization_algorithm
SYSTEM "optimization_algorithm.dtd">
<optimization_algorithm>
      <client>
            <client_name>ealba</client_name>
            <client_ip>192.168.198.3</client_ip>
            <client_id>6</client_id>
      </client>
      <features>
            <language>C++</language>
            <compilation>make COMPONEMAX</compilation>
            <execution>make EXEONEMAX</execution>
            <online></online>
            <comment></comment>
      </features>
      <params type="sa">
            <max_n_evals>500</max_n_evals>
            <fitness_function>onemax</fitness_function>
            <markov_chain_length>300</markov_chain_length>
            <temperature_decay>0.88</temperature_decay>
      </params>
      <results>
            <chromosome>1111111111</chromosome>
            <fitness>127.5</fitness>
            <computation_time>1 second</computation_time>
            <error></error>
      </results>
</optimization_algorithm>
```

Figure 10: XML Document for a *Simulated Annealing*.

6 Conclusions and Future Work

In this document we have discussed different plain ascii approaches to configuring optimization algorithms. Our goal has been to define how different algorithms could be managed from a common web optimization service that will serve as a front end for users to interact with them.

Since we intend to send code files containing the optimized function, parameters for configuring the algorithms, returned results to the user, and effectively run the algorithm in remote computers, we have selected XML to define our small language in the form of a DTD document. We intend to improve on this initial definition in the future, for example to allow algorithms to interact each other by exchanging such XML documents to guide the search.

The potential advantages of XML, its flexibility and its relationship with other technologies and standards support our initial guess of making research about an Internet system that could be implemented and managed with a proper DTD or XML schema definition.

Appendix: DTD Internals

```
<?xml version='1.0' encoding='us-ascii'?>

<!-- Name         optimization_algorithm.dtd          -->
<!-- Version      1.0                                  -->
<!-- Author       Jose Manuel Garcia Nieto            -->
<!-- Description  DTD for optimization_algorithm.xml  -->

<!-- root element -->
<!ELEMENT optimization_algorithm (client, features, params, results?)>

<!-- client specifications -->
<!ELEMENT client (client_name,client_ip,client_id?)>
<!ELEMENT client_name (#PCDATA)>
<!ELEMENT client_ip   (#PCDATA)>
<!ELEMENT client_id   (#PCDATA)>

<!-- main features of the algorithm -->
<!ELEMENT features (language, compilation, execution, online?, comment?)>
<!ELEMENT language    (#PCDATA)>
<!ELEMENT compilation (#PCDATA)>
<!ELEMENT execution   (#PCDATA)>
<!ELEMENT online      (#PCDATA)>
<!ELEMENT comment     (#PCDATA)>

<!-- language      (language of programming of the algorithm) -->
<!-- compilation   (compilation instruction/s)                -->
<!-- execution     (execution instruction/s)                  -->
<!-- online        (online results)                           -->
<!-- comment       (comment about the algorithm)              -->

<!-- all kinds of parameters -->
<!ELEMENT params (number_of_genes?, length_of_gene?, population_size?,
population_width?, population_height?, population_number? ,crossover?,
mutation?, max_n_evals?, migration?, fitness_function?, elitism?,
replacement?, selection?, recombine?, neighb?, max_temperature?,
min_temperature?, markov_chain_length?, temperature_decay?, upper_bound?,
lower_bound?, short_term_memory_length?, long_term_memory_length?,
candidate_list_length?, max_distance?, strategy?, cost_function?, others?)>

<!ATTLIST params
             type
(ssea|mea|genea|cea|dea|pea|cgpea|cfpea|sa|ts|grasp|bb|dp|simpleea|ee)
#REQUIRED >

<!-- EA     (Evolutionary Algorithm)                       -->
<!-- ssea   (Steady State EA)                              -->
<!-- mea    (Messy EA)                                     -->
<!-- genea  (Genetational EA)                              -->
<!-- cea    (Celular EA)                                   -->
<!-- pea    (Parallel EA)                                  -->
<!-- cgpea  (Coarse Grain PEA)                             -->
<!-- cfpea  (Fine Grain PEA)                               -->
<!-- sa     (Simulated Annealing)                          -->
<!-- ts     (Tabu Search)                                  -->
<!-- grasp  (Greedy Randomized Adaptive Search Procedures) -->
<!-- bb     (Branch and Bound)                             -->
<!-- dp     (Dynamic Programing)                           -->
<!-- simpleea     (simple EA)                              -->
```

Figure 11: DTD part 1.

```
<!-- parameter entity to define the range in numeric type-->
<!ENTITY % range
     "type (int|double|float|long|short|byte|boolean) #IMPLIED
     min CDATA #IMPLIED
     max CDATA #IMPLIED">

<!ELEMENT number_of_genes (#PCDATA)>
<!ATTLIST number_of_genes %range;>

<!ELEMENT length_of_gene (#PCDATA)>
<!ATTLIST length_of_gene %range;>

<!ELEMENT population_size (#PCDATA)>
<!ATTLIST population_size %range;>

<!-- only for cea -->
<!ELEMENT population_width (#PCDATA)>
<!ATTLIST population_width %range;>

<!-- only for cea -->
<!ELEMENT population_height (#PCDATA)>
<!ATTLIST population_height %range;>

<!ELEMENT population_number (#PCDATA)>
<!ATTLIST population_number %range;>

<!ELEMENT crossover (crossover_prob)>
<!ATTLIST crossover
                    type (spx|dpx|upx|apx) #IMPLIED>

<!ELEMENT crossover_prob (#PCDATA)>
<!ATTLIST crossover_prob %range;>

<!ELEMENT mutation (mutation_prob)>


<!ELEMENT mutation_prob (#PCDATA)>
<!ATTLIST mutation_prob %range;>

<!ELEMENT max_n_evals (#PCDATA)>
<!ATTLIST max_n_evals %range;>


<!ELEMENT migration (migration_freq?,migration_rate?)>
<!ATTLIST migration
                type (best|worst|random|null) #IMPLIED>

<!-- migration_freq    (frequency of migration regarding  number of
evaluations)-->
<!-- migration_rate    (number of individuals that emigrate)-->

<!ELEMENT migration_freq (#PCDATA)>
<!ATTLIST migration_freq %range;>

<!ELEMENT migration_rate (#PCDATA)>
<!ATTLIST migration_rate %range;>
```

Figure 12: DTD part 2.

References

[1] E R Harold, XML Bible, IDG Books, 1999.

[2] B McLaughlin, Java and XML, O'Relly, 2001.

[3] A Pew, Instant Java, The Sunsoft Press - Pretince Hall, 1996.

[4] A. Díaz, F. Glover, H. M. Ghaziri, otros. Optimización Heuristica y Redes Neuronales. Paraninfo S.A, 1996.

[5] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 1992.

[6] E. Alba, J. M. Troya. A Survey of Parallel Distributed Genetic Algorithms, Complexity, 4(4):31-52 1999.

```
<!-- for  generational EA -->
<!-- K best individuals to next generation -->
<!ELEMENT elitism (#PCDATA)>


<!ELEMENT replacement EMPTY>
<!ATTLIST replacement
          type (worst|random|if_better|always|null) #IMPLIED>

<!ELEMENT selection EMPTY>
<!ATTLIST selection
          type
(roulette_wheel|ranking|random|best_individual|tournament|null) #IMPLIED>

<!--- for Evolutive Strategy -->
<!ELEMENT recombine EMPTY>
<!ATTLIST recombine
          type (ee|linear|null) #IMPLIED>


<!-- neighborhood type and number of neighbors, only for cea -->
<!ELEMENT neighb (#PCDATA)>
<!ATTLIST neighb
          type (linear|compact|null) #IMPLIED>

<!-- maximum start temperature for simulated annealing -->
<!ELEMENT max_temperature (#PCDATA)>
<!ATTLIST max_temperature %range;>

<!-- minimum allowed temperature for simulated annealing -->
<!ELEMENT min_temperature (#PCDATA)>
<!ATTLIST min_temperature %range;>

<!ELEMENT markov_chain_length (#PCDATA)>
<!ATTLIST markov_chain_length %range;>

<!ELEMENT temperature_decay (#PCDATA)>
<!ATTLIST temperature_decay %range;>


<!-- for tabu search -->
<!ELEMENT short_term_memory_length (#PCDATA)>
<!ATTLIST short_term_memory_length %range;>

<!-- for tabu search -->
<!ELEMENT long_term_memory_length (#PCDATA)>
<!ATTLIST long_term_memory_length %range;>

<!-- for tabu search -->
<!ELEMENT  max_distance (#PCDATA)>
<!ATTLIST  max_distance %range;>

<!-- for branch and bound -->
<!ELEMENT upper_bound (#PCDATA)>
<!ATTLIST upper_bound %range;>

<!-- for branch and bound -->
<!ELEMENT lower_bound (#PCDATA)>
<!ATTLIST lower_bound %range;>
```

Figure 13: DTD part 3.

```
<!-- for grasp -->
<!ELEMENT candidate_list_length (#PCDATA)>
<!ATTLIST candidate_list_length %range;>


<!-- for branch and bound -->
<!ELEMENT strategy EMPTY>
<!ATTLIST strategy
          type (fifo|lifo|other) #IMPLIED>

<!ELEMENT fitness_function (#PCDATA)>
<!ELEMENT cost_function (#PCDATA)>

<!-- generic element -->
<!ELEMENT others (others_id,others_value)>
<!ELEMENT others_id (#PCDATA)>
<!ELEMENT others_value (#PCDATA)>
<!ATTLIST others_value %range;>

<!-- others_id      (id of other possible parameter)                  -->
<!-- others_value   (value of other possible parameter)               -->

<!--  results of the algorithm  -->
<!ELEMENT results (allele?, chromosome?,individual?, fitness?,
best_fitness?, worst_fitness?, migrations?,
                    solution? ,avg_fitness?, stddev_fitness?,
computation_time?, communication_time?,
                    run_time?, error?)>

<!ELEMENT allele (#PCDATA)>

<!ELEMENT individual (#PCDATA)>
<!ELEMENT chromosome (#PCDATA)>
<!ELEMENT fitness (#PCDATA)>
<!-- generic solution -->
<!ELEMENT solution (#PCDATA)>

<!ELEMENT avg_fitness (#PCDATA)>
<!ELEMENT stddev_fitness (#PCDATA)>
<!ELEMENT best_fitness (#PCDATA)>
<!ELEMENT worst_fitness (#PCDATA)>

<!ELEMENT migrations (#PCDATA)>

<!ELEMENT computation_time (#PCDATA)>
<!ELEMENT communication_time (#PCDATA)>
<!ELEMENT run_time (#PCDATA)>
<!ELEMENT error (#PCDATA)>
```

Figure 14: DTD part 4.