Capture and Maintenance of Project Total Cost Minimization Applications' Results

Rui Moutinho
Informatics Department
University of Minho
Braga, Portugal
rumout@gmail.com

Anabela Tereso
Production and Systems Department
University of Minho
Braga, Portugal
anabelat@dps.uminho.pt

Abstract — **In the past several years, various optimization algorithms had been implemented on the project total cost minimization problem. Lately, it was developed an application that serves as a central platform that integrates all those previous implementations. Currently, such platform allows the access and execution of each one of the other utilities as modules/plugins. Each of which can be configured to execute optimizations over different projects. Although the platform already saves the optimization results, it lacks a suitable processing mechanism that would ease the cross analysis over all the utilities. Therefore, we want to include capture and analysis of results into the platform in order to properly aggregate them in a results database. Such database could then be queried for numerous purposes being the performance evaluation, across the optimization utilities, one of the first. In this project we analyze a total of five heterogeneous optimization utilities, all coded in JAVA, using algorithms based on dynamic programming and global optimization. The heterogeneity poses a challenge which we overcame by establishing a common results storage language, by means of XML files.**

Project Scheduling; Optimization Applications;* XML *Processing

I. INTRODUCTION

We start by summarize the context of the developments leading to all the optimization utilities to be analyzed. Then, a brief presentation of the specific purpose and method used of each one of those applications.

A. Context

Over the past years, several contributions were made improving the early developments on project cost optimization [1].

The goal is to determine the resource quantity allocation table that minimizes the total – allocation plus tardiness – project cost. Specifically, models were developed to address that with multimodal project activities under stochastic conditions whose precedence relation is fully described by an AOA (Activity-on-Arc) network. With multimodal, we mean each activity having its total duration as a function of the allocated quantity of each of its resources. Moreover, this relation is non-discrete and stochastic in nature. In fact, all models use the exponential distribution. The following major approaches have been used:

DP (Dynamic Programming) based: This relies on the structure of the AOA network in order to establish a set of stages by evaluating its UDC (Uniformly Directed Cutset) [1]. The method is recursive and evaluates on each stage a policy table based on the possible state configurations. Due to its complexity, a differentiation on the variable set (the activities) is enforced: fixed activities and decision activities. This allows to proceed in order to one decision variable per stage decreasing the complexity. The other variables are evaluated by testing each combination of its possible values.

Global optimization: Two algorithms are used: EMA (Electromagnetic Algorithm) [2] and EVA (Evolutionary Algorithm) [3]. These simulate the dynamic of a set of elements – possible solutions – by applying electromagnetic laws on ions or evolution laws on individuals, respectively. On each iteration the elements are randomly disturbed and when the stoping criteria are meet, the best one is taken as the solution. Usually, these algorithms are replicated several times to ensure better results.

B. Optimization Applications

All the applications share the same goal. However they do so with different strategies and algorithms which we describe, in chronological order:

AAREC: Is the migration of the initial implementation using DP (Dynamic Programming) in MATLAB [4] to JAVA [5]. It allows the optimization to process on either distributed mode or non-distributed mode with a basic level of concurrent programming.

AEM & AEV: The AEM [6] is the migration of the EMA implementation in MATLAB [2] to JAVA. The AEV is the first to implement EVA [3]. Both applications can execute on distributed and non-distributed modes.

COMURA: Implements all the three optimization algorithms applied to an extension of the previous models introducing the multiple resources scenario [7][8][9][10]. The COMURA does not provide the execution in distributed mode, rather it uses concurrent programming.

CENTRAL PLATFORM: The purpose of it is to aggregate all the previous optimization applications under a plug-in platform allowing easier execution of those utilities [11].

C. Objective

Given an extensible aggregating platform, we want to develop a suitable utility that will be responsible for the capture, processing and storage of the results obtained from the optimization applications on a common database. The immediate purpose would be the performance comparison between the several implementations but the system should be more flexible allowing a wider range of informative potential.

II. APPLICATIONS ANALYSIS

In this section we analyze the plug-in system of the central platform and the information flow and quality of each application.

A. The Central Platform Plug-In System

The central platform allows a single interface point for the plug-in modules: a panel, within a panel slider. In such panel, each module should provide to the user whatever configuration elements needed. The graphical interface of the platform also provides a text area, to which the standard output stream is redirected; and a control area where the user has additional control over the selection of the input project and over the results whereabouts (see Fig. 1).

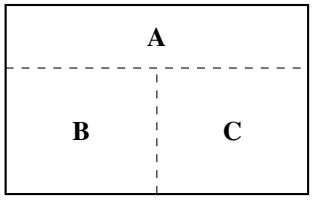


Figure 1. Central platform main view scheme. **A** Control area; **B** Plug-in panels zone; **C** Text area for output.

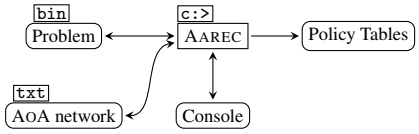
Despite its plug-in design, the communication between the platform and the modules barely goes beyond a "load-start-get" vocabulary and there is no way to the loaded modules to communicate directly with each other.

The platform already enables the storage of results. However, it only includes the final results and if a module is executed several times for the same project, only the last results are preserved. This is specially unacceptable for those applications using the global optimization algorithms which are prone to give different results for the same initial conditions.

B. Information Flow

All the optimization applications have the same objective. Therefore, all require information about the project and after execution, the result is composed of the resources allocation table and the project total cost. All the applications are console-based in their stand-alone versions. However, regarding the information flow, each one accepts input and delivers output from and to different vessels: command line, console I/O and files.

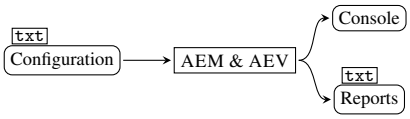
The AAREC (see Fig. 2) is configurable from command line, console and external files. The project information required can be from user input via console or from external files with previous saved data. The progress and produced information are printed to the console and the policy tables saved to external files.



c:> command line use bin binary file txt text file

Figure 2. AAREC information flow scheme.

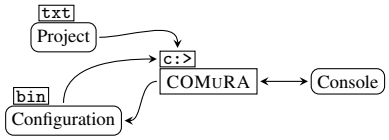
The AEM and the AEV (on Fig. 3) are only configurable through external files. The console is used only to give feedback with execution configuration and produced data. The same information is recorded to an external file.



txt text file

Figure 3. AEM and AEV information flow scheme.

The COMURA (see Fig. 4) is configured through both command line and console input. It allows to save a binary file with the algorithm configuration used for import on subsequent executions. All the runtime information is printed to the console and varies according to the user defined command line arguments: enabling and disabling progress indication or partial results output, for example.



c:> command line use bin binary file txt text file

Figure 4. COMURA information flow scheme.

C. Information Quality

Given the divergences on the data organization, we had to address the quality and completeness of the information required and produced by all the applications towards the creation of a database. Such database must contain enough information to allow performance and results comparison while providing enough data per execution in order to prevent future and expensive re-executions. The Tab. I summarizes all the information available through each application whether externally (files), internally (console) or produced.

TABLE I. INFORMATION AVAILABILITY ON EACH APPLICATION

Data	AAREC	AEM & AEV	COMURA
Execution environment (host machine)	none	none	none
Project	partial (no project name)	partial (no project name)	all (both single and multiple resource)
DP algorithm configuration	enabled (two parameters)	non-applicable	enabled (four parameters)
EMA configuration	non-applicable	partial (mostly hard-coded)	enabled (all parameters)
EVA configuration	non-applicable	partial (mostly hard-coded)	enabled (all parameters)
Partial results	always (policy tables)	always (initial and final populations)	user request
Final results	yes	yes	yes
Other algorithm specific data	non-distributed mode (UDC (Uniformly Directed Cutset))	always (work content sample tables)	user request (doesn't show UDC)
Application configuration	yes (threads count)	yes (additional stop criteria)	yes (used algorithm; concurrent and non-concurrent execution policies)

D. Decisions

The current results storage system does not meet the requirements. A natural thinking is to make a plug-in module that will add to the platform a suitable results processing mechanism. However, the interface available induces a bubble-like environment where the communication between platform and modules and between the modules themselves is very restricted. Thus, the capture of the results by another plug-in is very difficult, if not impossible. So, a new central platform design must be implemented.

Looking at the Tab. I we notice that the information is heterogeneous both in quality and completeness. If we only wanted to process the information regarding the input project and the results, that should not be a problem. However, we want to retain as much information as possible. Some information can be discarded at early studies but, with further developments, such data may become useful. Therefore, the database should be both complete and flexible enough to allow a wide spectrum of studies. On this regard we decided to use XML files for the information storage.

III. IMPLEMENTATION

Finally, we present our current developments. Like all the optimization applications, the implementations were made in JAVA.

A. The New Central Platform

Due to scheduling reasons it was impossible to us to produce a complete new platform with a rich plug-in system. Instead, we developed a prototype – PROCOOL – that integrates the applications without that feature. The PROCOOL is specially focused on the information processing: capture, extraction and translation. This is explained in section III–B.

The PROCOOL is logically divided in three internal units, all encapsulated in a graphical interface: configuration, execution and information processing units. The first maintains all the configurable parameters per application. Then, the execution unit uses that context when calling an instance of a selected application. This execution may produce additional files alongside with the console output – the execution result environment. Finally, the information processing unit gathers and transforms the data on both configuration and execution result environment. The interface provides all externally configurable information in one panel per application, emulates a console (with input and output) and provides views of the resulting XML and log (see Fig. 5).

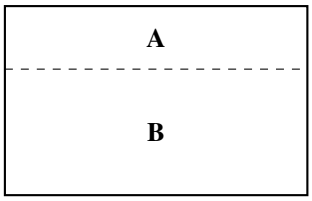


Figure 5. PROCOOL – Graphical interface. **A:** Configuration panel; **B:** Console emulation, log and XML preview

B. Information Processing

In this section we begin by explaining the general procedure followed by the details concerning the XML export format.

1) General Procedure

The all process is composed of three sequential stages: capture; extraction and translation.

On the first stage – capture – all input and output text written on the console is grabbed. This text together with any text of external files produced/used constitutes the source information. This raw data is parsed in the extraction stage. To ease the coding of this stage we used JAVACC[1]. The raw data (mostly

[1] Information and download at `https://javacc.dev.java.net/`

the one from the console) has many redundant or user-oriented only information like progress indications and input error messages, etc[2]. Such details proved to be tricky to circumvent using a direct "by hand" approach, hence the tool use. Finally, the translation phase gets the parsed information and completes it as needed using the configuration panel and the java machine properties; producing the final XML file.

2) Structure

The structure of the XML file was decided after identifying the common information to all applications. Below, we describe each area of the XML file[3].

Header: The execution context of the java virtual machine: name, version and type of the operating system; processor count and total available memory.

Project: The information relative to the targeted project. For completeness, a name based on the characteristics of the project has to be generated for the AAREC, AEM and AEV applications due to their lack of such name on their specifications.

Execution: The results and specific context of the optimization execution: the allocation table; the obtained project cost and the elapsed time. Also, there are stored the designations of the optimization application and algorithm used; start date; host machine name (a weaker alternative to the header section) and the name and version of the central platform used (here named as launcher).

Configuration: It is divided in two areas: optimization application and optimization algorithm. The algorithm configuration is stored with the maximum available parameters. Consequently, for the applications do not providing full algorithm configuration, all non-configurable parameter values are determined by using the hard-coded ones.

Extras: The remaining extracted information is stored here.

3) Database

The set of all the individual XML files produced be each optimization makes the source of our database. In order to begin the first tests, we created a simplistic application that makes a simple index of all those files in one folder – PROCORE. It allows us to add new XML files, dealing with eventual conflicts, and enables the extraction of reports. These reports are XML files with the gathered allocation tables and costs obtained for a specific project using the several applications and configurations. Those reports are a basic structure with which to compare the performance of the applications. Furthermore, we use XSLT[4] to produce human-readable versions of the reports.

[2]Due to limitations, it is not possible to include some printed executions. You may ask for them by email to the leading author at `rumout@gmail.com` with subject "CISTI'10" (to avoid undesired filtering).

[3]The actual XML schema is too big to be included in this article. However, if you desire you may send a request for the file by email to the leading author at `rumout@gmail.com`. Please use "CISTI'10" as subject to avoid eventual filtering by the anti-spam engine.

[4]`http://www.w3.org/TR/xslt`

IV. RESULTS

Due to limitation rules we only show one formatted output (see it in Fig. 6) and one report over the database so far created.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<XOU tag="1243331935962">
  <HEADER>
    <OS_NAME>Windows Vista</OS_NAME><OS_VERSION>6.0</OS_VERSION>
    <OS_ARCH>amd64</OS_ARCH><N_CPU>2</N_CPU>
    <AVAIL_RAM>477233152</AVAIL_RAM>
  </HEADER>
  <PROJECT>
    <NAME>Net3_OldFormat</NAME>
    <SCHEDULE_TIME>16.0</SCHEDULE_TIME>
    <TARDINESS_COST>8.0</TARDINESS_COST>
    <PROJECT_RESOURCES>
      <PROJECT_RESOURCE id="r1">
        <MINIMUM_QUANTITY>0.5</MINIMUM_QUANTITY>
        <MAXIMUM_QUANTITY>1.5</MAXIMUM_QUANTITY>
        <QUANTITY_COST>1.0</QUANTITY_COST>
        <MAINTENANCE_COST>0.0</MAINTENANCE_COST>
      </PROJECT_RESOURCE>
    </PROJECT_RESOURCES>
    <PROJECT_ACTIVITIES>
      <PROJECT_ACTIVITY id="a1">
        <START_NODE>n1</START_NODE><END_NODE>n2</END_NODE>
        <ACTIVITY_RESOURCES>
          <ACTIVITY_RESOURCE id="r1"><LAMBDA>0.2</LAMBDA></ACTIVITY_RESOURCE>
        </ACTIVITY_RESOURCES>
      </PROJECT_ACTIVITY>
      <PROJECT_ACTIVITY id="a3">
        <START_NODE>n1</START_NODE><END_NODE>n3</END_NODE>
        <ACTIVITY_RESOURCES>
          <ACTIVITY_RESOURCE id="r1"><LAMBDA>0.07</LAMBDA></ACTIVITY_RESOURCE>
        </ACTIVITY_RESOURCES>
      </PROJECT_ACTIVITY>
      <PROJECT_ACTIVITY id="a2">
        <START_NODE>n2</START_NODE><END_NODE>n3</END_NODE>
        <ACTIVITY_RESOURCES>
          <ACTIVITY_RESOURCE id="r1"><LAMBDA>0.1</LAMBDA></ACTIVITY_RESOURCE>
        </ACTIVITY_RESOURCES>
      </PROJECT_ACTIVITY>
    </PROJECT_ACTIVITIES>
  </PROJECT>
  <RUN>
    <START>2009.05.26 10:58:55.962 BST</START>
    <APPLICATION>COMuRA</APPLICATION>
    <ALGORITHM>DPA</ALGORITHM>
    <LAUNCHER version="1.0">ProCoOL</LAUNCHER>
    <HOST>Selenius</HOST>
    <RESULT>
      <ALLOCS>
        <ACTIVITY id="a1" type="decision">
            <RESOURCE id="r1">1.5</RESOURCE>
        </ACTIVITY>
        <ACTIVITY id="a3" type="fixed">
            <RESOURCE id="r1">1.5</RESOURCE>
        </ACTIVITY>
      </ALLOCS>
      <COST>62.387</COST>
      <RUNTIME>130</RUNTIME>
    </RESULT>
  </RUN>
  <CONFIGURATION>
    <COMURA>
      <CONCURRENCY>
        <EXECUTION_POLICY>Single Task</EXECUTION_POLICY>
        <OF_EVALUATION_POLICY>Single Threaded</OF_EVALUATION_POLICY>
      </CONCURRENCY>
    </COMURA>
    <DPA>
      <NSQF>3</NSQF>
      <NSQD>5</NSQD>
      <NSWF>4</NSWF>
      <NSWD>4</NSWD>
    </DPA>
  </CONFIGURATION>
  <EXTRAS>
    <STATISTICS>
        <PROCESSED_TASKS>3</PROCESSED_TASKS>
        <MISSED_TASKS>0</MISSED_TASKS>
    </STATISTICS>
  </EXTRAS>
</XOU>
```

Figure 6. Formatted output of a COMURA execution.

On Fig. 7 on the next page we show a report that collects all the registered executions of the COMURA application on the host named *Selenius* for a specific project; made by PROCORE and transformed with a XSLT file[5].

[5]The source XML file of the presented report may be requested by email to the leading author at `rumout@gmail.com`. Please specify "CISTI'10" as subject.

Results-by-Project Report

Project name: Mures 1.1 - 3a3n
Host name: Selenius

COMuRA@Selenius

Algorithm	Allocation			Cost	Runtime
DPA	[1.50, 1.00, 1.00]	n.a.	[0.50]	113.147	0:00:00.130
	[1.50, 1.00, 1.00]	n.a.	[0.50]	113.147	0:00:00.147
EMA	[0.99, 0.69, 0.50]	[1.08]	[0.50]	72.425	0:00:00.418
	[1.06, 0.74, 0.50]	[1.11]	[0.50]	81.261	0:00:00.432
	[1.20, 0.84, 0.50]	[1.08]	[0.50]	84.690	0:00:00.316
	[1.12, 0.78, 0.50]	[1.10]	[0.50]	79.742	0:00:00.407
EVA	[1.28, 0.90, 0.50]	[1.08]	[0.50]	91.041	0:00:00.387
	[1.23, 0.86, 0.50]	[1.10]	[0.50]	88.163	0:00:00.365
	[1.27, 0.89, 0.50]	[1.09]	[0.50]	90.734	0:00:00.378
	[1.15, 0.80, 0.50]	[1.12]	[0.50]	93.135	0:00:00.407
	[1.17, 0.82, 0.50]	[1.08]	[0.50]	85.810	0:00:00.282

Figure 7. A XML report from PROCORE over the database transformed via XSLT in HTML as viewed on a browser.

V. DISCUSSION

Despite the same objective, the information extracted from each optimization application varies both in quantity and quality. For example, in order to characterize a project is not only needed to specify its AOA network, tardiness unitary cost and schedule completion time. A simple name is important for future reference: it is far easier to compare names than to compare all the contents!

The information is, therefore, incomplete and poorly specified. This happens because the applications were made independently from each other and in different points of project management development. Thus, a central platform was created trying to homogenize those applications into a plug-in system. However, such platform failed to provide a suitable processing of the execution results. A new central platform is, then, created to fill in the gap.

The information was easy to complete once we established a common base to all applications. Such basis concerns on retaining as much information as possible in order to avoid future remakes of the same expensive executions. Thus, whereas the information is absent, we get it by external means – the execution context of the virtual machine – or by generation through other properties – project names. Since we have access to the source code, it was simple to fill the hard-coded values used as configurations on some applications. But having such a homogenous basis is not sufficient.

In order to accommodate future applications we want a format that is also flexible. Some information that is regarded as secondary today may be useful tomorrow. We chose to create XML format, thus, forming a XML database. Perhaps the same purposes could be fulfilled by a relational database but the XML allows the storage and specification of the information in a single file. This brings better support for portability and interchange. These are required because the team working on the subject from which this article stems has changed over the years with several contributions build one over another and is more likely to continue this way. Of course we are well aware of the traditional benefit of the relational databases concerning performance. Specifically when dealing with queries involving extensive amounts of data. It is possible in a near future to have a huge collection of data. However, that is not our main concern. The research is far from concluded and so, more applications with new models and algorithms are likely to be developed. These will bring more information which we will need to insert. By using XML is possible to do some changes on the structure while ensuring retro-compatibility quite easily. The same is not true with relational databases where a few changes may imply an expensive overhead when migrating from one database version to another.

VI. CONCLUSION

The need for a new central platform increased our insights about each optimization application demands. This, together with the knowledge of the weaker and stronger points of the other platform allows us to establish a set of guidelines for the future applications and newer versions of the existent. Thus, the optimization applications should:

- be console applications. Since they are optimization applications, it makes little sense to expend useful resources with graphical interfaces;
- operate by consuming a complete configuration file and producing a pre-formatted result file. This will minimize the user interaction and feedback leaving more resources available to the applications.

A new version of the central platform should provide graphical (more intuitive) interfaces to edit the configuration files together with a suitable plug-in system with a translating module for processing of the result files.

The actual version of the PROCORE is very basic and limited. By using a well developed XML database library we can easily enhance the PROCORE to a more stable and scalable version. For this purpose the BERKELEY XML DB[6] seems to be a good candidate because not only its background maturity but also its zero administration feature. That database library provides a rich application interface requiring only a couple of files to be distributed with the main application. Thus, a PROCORE version powered by it will also retain its portability feature.

[6] `http://www.oracle.com/database/berkeley-db/xml/index.html`

REFERENCES

[1] A. Tereso, Gestão de projectos – alocação adaptativa de recursos em redes de actividades multimodais, Ph.D. thesis, Production and Systems Department, University of Minho, 2002.

[2] A. Tereso, M. Araújo, and S. Elmaghraby, The Optimal Resource Allocation in Stochastic Activity Networks via the Electromagnetism Approach, Nancy, France: Ninth International Conference on Project Management and Scheduling, April 2004.

[3] A. Tereso, L. Costa, R. Novais, and M. Araújo, The Optimal Resource Allocation in Stochastic Activity Networks via the Evolutionary Approach: a platform implementation in java, Beijing, China: International Conference on Industrial Engineering and Systems Management, May 30 – June 2 2007.

[4] A. Tereso, M. Araújo, and S. Elmaghraby, Adaptive resource allocation in multimodal activity networks, International Journal of Production Economics, vol. 92(1), (2004), pp. 1–10.

[5] A. Tereso, J. R. Mota, and R. J. Lameiro, Adaptive resource allocation to stochastic multimodal projects: a distributed platform implementation in java, Control and Cybernetics, vol. 35(3), (2006), pp. 661–686.

[6] A. Tereso, M. Araújo, and R. Novais, The Optimal Resource Allocation in Stochastic Activity Networks via the Electromagnetism Approach: A Platform Implementation in Java, Reykjavíc, Iceland: 21st European Conference on Operational Research, July 2006.

[7] R. Moutinho, Gestão de Projectos – Alocação de Múltiplos Recursos, Tech. rep., University of Minho, Braga – Portugal, December 2007.

[8] A. Tereso, M. Araújo, R. Moutinho, and S. Elmaghraby, Project management: multiple resources allocation, Rio de Janeiro, Brazil: International Conference on Engineering Optimization, 2008.

[9] A. Tereso, M. Araújo, R. Moutinho, and S. Elmaghraby, Duration oriented resource allocation strategy on multiple resources projects under stochastic conditions, Montreal, Canada: International Conference on Industrial Engineering and Systems Management, 2009.

[10] A. Tereso, M. Araújo, R. Moutinho, and S. Elmaghraby, Quantity oriented resource allocation strategy on multiple resources projects under stochastic conditions, Montreal, Canada: International Conference on Industrial Engineering and Systems Management, 2009.

[11] J. P. Cardoso, Gestão de Projectos – AoA Manager, Tech. rep., University of Minho, Braga – Portugal, 2008.