Solving Combinatorial Problems: An XML-based Software Development Infrastructure *

Rui Barbosa Martins[1,2], Maria Antónia Carravilla[1,2], Cristina Ribeiro[1,2]

[1] FEUP—Faculdade de Engenharia da Universidade do Porto
[2] INESC—Porto
Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
{ei01018,mac,mcr}@fe.up.pt

Abstract. The resolution of combinatorial problems typically requires the articulation of tools that range from modeling languages to dedicated solvers, including processing input data sets and visualizing results.
This work concerns the improvement of the software development environment for a research project using a custom-designed XML dialect. NestingXML has been designed to capture one kind of combinatorial problems in what concerns their input and output data. The dialect is used for storing problem and solution descriptions in a flexible way. In a project context, data formatted according to the dialect are imported into a Java API used for developing solvers and associated tools. The problem description is enriched with both preprocessing data and solution descriptions.
We describe the NestingXML dialect and the Java API used in the project and illustrate their use in the problem-solving process. The resulting environment demonstrates increased flexibility in data representations and will become an easy integration medium for new team members.

Keywords: XML, Cutting and Packing, Constraint Programming.

1 Introduction

The resolution of combinatorial problems typically requires the articulation of tools that range from modeling languages to dedicated solvers, and also include the processing of input data sets and the visualization of results.

A research team involved in developing solutions for a class of combinatorial problems faces the significant overhead of maintaining consistent representations for problem instances and solution descriptions that can be shared among researchers who are working on different approaches.

The GLOBALNest project (Global Constraints for Nesting Problems) is focused on developing constraint programming solutions for nesting problems,

* Supported by FCT under project POSI/SRI/45379/2002 (GLOBALNest)

a category of cutting and packing problems where polygon-shaped pieces are placed over a board and the goal is to minimize the length of the resulting layout. The development and test of new solution methods requires the use of known problem instances and the execution of test suits under different program configurations.

In the context of the project, data for the problem instances is available in the form of text files. The resolution methods make use of previously defined modules for geometric manipulations; *ad hoc* formats have been specified for their output. In the course of some of the project developments, extra information has been required for some of the modules and the data formats were modified accordingly. It soon became clear that extending and modifying the data format would be the rule in projects of this nature, and that a more flexible representation was needed.

An XML data format has been defined covering the expected complexity of the descriptions for this class of problems and their solutions. The existing tools for parsing input data and presenting results have been kept as "legacy data" transformers and new tools are being developed for manipulating this more expressive data format.

The project development platform has a Java API where the concepts of the problem domain are represented. Several tools that deal with the geometry and some output format generators were developed based on the API.

The paper describes an infrastructure for software development and solver evaluation comprising the XML data format (NestingXML), the Java API and the set of tools that support the preparation of inputs for the solvers and the evaluation of results.

The paper is organized as follows. Section 2 briefly describes the project development environment. The formats used for data input and output are detailed in Section 3. The Java API used in the project to capture the main concepts of the problem domain is presented in Section 4. The XML data format developed for this domain is the subject of Section 5, followed by the update of the API in Section 6. Section 7 illustrates the use of the dialect and the API and some conclusions follow.

2 Combinatorial Problems and Solutions

Combinatorial problems typically require modeling a problem domain, characterizing feasible solutions and using a search strategy to explore the solution space and find solutions that best satisfy a problem goal.

Research in combinatorial problems involves developing new representations for the problem domains, to ease the expression of the constraints that define feasibility, and devising new search strategies that lead to the exploration of either larger or more promising regions of the search space.

In the case of cutting and packing problems, the goal is to minimize waste when packing items in a container or cutting out small pieces from a surface of

raw material. Fig. 1illustrates a solution layout, with several polygonal pieces positioned on a plate.

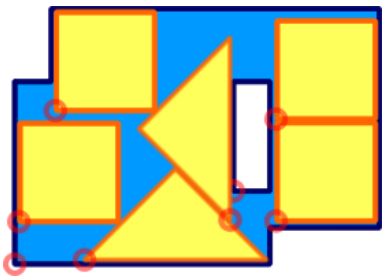

Fig. 1. A solution for a nesting problem

The solution of a nesting problem is the choice of a position for each one of a set of polygon-shaped pieces over a plate—a layout. It is an intrinsically 2D problem where the central constraints must express the fact that any 2 pieces may not overlap. The problem is being addressed in the Operations Research community with Mixed Integer Programming and heuristic methods.

To represent "non-overlappedness" in 2D requires some geometrical preprocessing of the input. Moreover, any development aimed at improving solution quality tends to require the computation of new values to characterize either the input or some intermediate solution configuration.

The testbed for the work described is the GLOBALNest project (Global Constraints for Nesting Problems), a research project where the main concern is the development of specialized constraints to handle the solution of these problems.

In a typical solution process workflow the problem description is read from a file into a memory representation used by the solver software. When modules in several languages are used, intermediate files may be required to pass-on information. If preprocessing of the input data is required, as is the case with geometric problems, then intermediate steps add to the characterization of the input data.

In a test setup, several input sets must be run through selected solvers in order to compare solutions. It is interesting in this case to be able to configure complete tests, assemble input data and collect results.

The problems tackled so far by the research team consider the board as a rectangular stripe whose length is to be minimized. The first versions of the solver were implemented using the SICStus-Prolog CLP-FD module [1]. In [2] and in [3], a CLP approach to the resolution of nesting problems is presented, using reified constraints and pieces represented by respectively convex and non-convex polygons. In the following step a special global constraint named "outside" [4] was developed to handle specifically the non-overlapping of a pair of general

polygons, leading to improvement of the domain reduction during search. During this period the Web application "CortaBem" [5] was used as a tool to draw the pieces, build the input files for the solver and draw the layouts.

In a more recent stage of the work, a solver using the global constraint "outside" has been implemented in ILOG with the purpose of testing another type of tool that would allow the use of a hybrid approach, combining CP techniques with Mixed Linear Programming.

3 Data Formats

The solution process for nesting problems is therefore a workflow with various intermediate files and solver modules. The way test examples are stored, can be manipulated, extended and put to some use may improve or strongly reduce researcher's productivity.

Initially the problem descriptions were stored in *ad hoc* file formats. Those formats were conceived with the purpose of making it easy for researchers to parse the data and feed them into their solver and there was no concern with the extensibility of the format. But today's ideas and data requirements for a solver may not be tomorrow's. During research different approaches are tried in order to get better solutions and the need to enrich previous formats arises. The lack of an easy way to extend format contents usually leads either to complicating even more an existing format by hacking in new required data, or to the creation of one more file format.

In our case study there are up to seven different formats taking an active role for each solver iteration:

1. *jfo* – contains a description of the pieces and their quantities
2. *brd* – contains board information
3. *prb* – contains problem information
4. *dat* – contains polygon descriptions
5. *nfp* – contains intermediate polygons built from the polygons on the *dat* format
6. *si* – contains input data to be directly read by the solver
7. *so* – contains nesting problem solutions

Some formats (*dat* and *nfp*) store intermediate results, are active for very short periods and do not get stored after the solution process is finished. All the other formats must be stored as each one contains unique information about the problem or the solution.

In Listing 1 we can see some of the content of an example *dat* file. The language used in some of the fields is portuguese, making it difficult to share the format among the research community. One thing common to all formats is the not so intuitive way data is stored. The file begins with the number of pieces to be described (7). It then proceeds enumerating each of the polygons assigning them a string identifier (e.g., *PECA 1*, defining the pieces quantity in the lot,

the number of vertices (e.g., 5) and their enumeration. Each vertex is described by its X and Y coordinates.

To understand the format there is a need for additional information, such as the one given above. Without that information, the meaning of the format will be lost and the format itself becomes useless.

```
NUMERO  DE        PECAS
        7

PECA              1  QUANTIDADE
        5
NUMERO  DE        VERTICES
        6
VERTICES(X,Y)
        0.0       0.0       2
        2.0       -1.0       2
        4.0       0.0       2
        4.0       3.0       2
        2.0       4.0       2
        0.0       3.0       2

(  ...  )
```

Listing 1: *dat* file content

Coping with all the scattered information of these files is not an easy task and it tends do get harder as new problem approaches are explored.

4 The Java API

To ease the manipulation of the formats described in the previous section, we chose to create a Java Application Programming Interface (API). The API provides an easy way to read each format into Java data structures, and to write the files back with added content.

These basic functionalities provided by the API allowed us to build a set of tools to manipulate the nesting problem's data. The first tools were built to compute some geometric properties of the problem's pieces. Some of these properties could then be used to generate sorted sequences of pieces to be positioned. An example is a sequence of pieces sorted by ascending convex hull area [6].

Graphic tools have also been created on top of the API. They provide an easy way to visualize both the problem's lot and its solutions. The renderings are then stored to jpeg files, allowing the representation of a problem and its solution to be easily rendered in a web browser.

5 NestingXML

The ideas for a XML format that could fully describe a nesting problem and its solutions first came about early 2005. In order to develop a format that would be used by all the researchers dealing with similar problems, the format was first developed in-house and then fully accepted by the European Research Group on Cutting and Packing Problems [7]. We currently have a stable first version of the XML format, for which an XML Schema is available [8]. NestingXML accomplishes the following objectives:

- Ability to represent both very simple and very complex problem instances in a common format.
- Ability to store both the nesting problem, its solutions and intermediate computations in a self contained file.
- A clear, human readable format.
- An easily extensible format (due to the use of XML).

The NestingXML format is structured in five sections:

1. Information about the author and a brief description of the problem.
2. Description of the problem instance.
3. Heuristics, intermediate computations and extra information.
4. Geometrical description of all the polygons used in the problem.
5. Solutions of the problem.

The first section includes some metadata for the problem described in the file, such as name and contacts of the author. A brief and informal description of the problem may also be added to this section. In this description, the author can point out interesting features of the problem that couldn't be easily spotted out by its formal description. A simple example of this first section can be seen in List. 2.

```
<name>Nesting problem example</name>
<author>Rui Martins</author>
<date>2005/11/04</date>
<description>
    A brief informal description about the nesting problem ↩
should be placed here.
</description>
<verticesOrientation>clockwise</verticesOrientation>
<coordinatesOrigin>up-left</coordinatesOrigin>
```

Listing 2: Example of the first section of a NestingXML file

The formal description of the nesting problem is presented in the second section of the NestingXML format. The section begins with the description of

the boards, which represent simple or complex geometrical 2D figures where the pieces are positioned. A board can be described by several polygons, representing materials of different qualities or holes, e.g., a simple way of describing a window frame could be by defining two superimposed rectangles of different sizes, where the smaller one is a hole. The description of the pieces to be positioned follows. Pieces and boards use similar structures, since both represent geometrical shapes. A small example of this section is present in List. 3. Fig. 2 has the corresponding graphical representation.

```
    <problem>
        <boards>
            <piece id="board1" quantity="1">
                <component type="0" idPolygon="poly0" ↔
xOffset="0.0" yOffset="0.0" />
                <component type="-1" idPolygon="hole0" ↔
xOffset="6.0" yOffset="-3.0" />
            </piece>
        </boards>
        <lot>
            <piece id="piece1" quantity="1">
                ...
            </piece>
            ...
        </lot>
    </problem>
```

Listing 3: Example of the second section of a NestingXML file

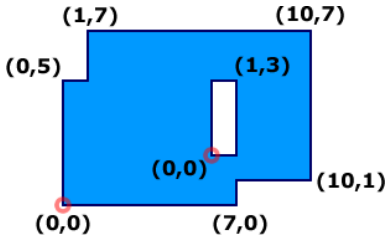

Fig. 2. Graphical representation of the board described in List. 3

The third section is meant for researchers to store extra information on the problem, such as time-consuming intermediate computations and information on heuristics. Listing 4 has a short example of the XML describing *Nofit Polygons*. Nofit polygons are a geometric transformation that makes the requirement

of "non-overlappedness" of two polygons easier to handle. Overlapping of two polygons can be decided based on the relative position of a polygon (the nofit) and a point. The visual representation of the computation of the nofit polygon can be seen in Fig. 3, where polygon R is the result of the orbit movement of polygon O around static polygon S.

```
<nfps>
    <nfp>
        <staticPolygon idPolygon="poly1" angle="0.0" ↩
mirror="none" />
        <orbitingPolygon  idPolygon="poly1" angle="0.0" ↩
mirror="none" />
        <resultingPolygon idPolygon="nfp_p1_p1" />
    </nfp>
     ...
</nfps>
```

Listing 4: Example of the third section of a NestingXML file

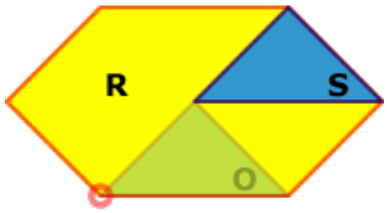

Fig. 3. Graphical representation of the No Fit Polygon information described in List. 4

Section four of the NestingXML format takes advantage of the fact that nesting problems use several geometrical data. To reduce the verbosity and avoid duplication of information in the file, this section describes polygon line segments and other relevant geometrical information. Each polygon represented here has a unique id, enabling it to be used as components of the boards, pieces and other geometrical structures. Consider for example a piece to be a square of 2×2 and a board with a 2×2, both of them could use a 2×2 square polygon described in this section avoiding the duplication of information. A small example containing the description of a triangle can be seen in List. 5. Going back to Fig. 3 one can see two triangles which correspond to the polygon described here.

The fifth and final section of the NestingXML format is reserved to describe the solutions of the problem, which are represented by the X and Y coordinates of

```
<polygons>
    <polygon id="poly1" nVertices="3">
        <lines>
            <segment n="1" x0="0.0" y0="0.0" x1="1.0" ↩
y1="1.0" />
            <segment n="2" x0="1.0" y0="1.0" x1="2.0" ↩
y1="0.0" />
            <segment n="3" x0="2.0" y0="0.0" x1="0.0" ↩
y1="0.0" />
        </lines>
        <xMin>0.0</xMin>
        <xMax>2.0</xMax>
        <yMin>0.0</yMin>
        <yMax>1.0</yMax>
        <perimeter>4.8284</perimeter>
        <area>1</area>
    </polygon>
</polygons>
```

Listing 5: Example of the fourth section of a NestingXML file

the positioning points (placement vertex) of the pieces. An example of a solution is presented in List. 6.

For the time being we chose not to create a specific element to store a numeric value representing the "solution's quality" because there is no clear way of generalizing this. Different approaches tend to use different measures of quality. There is room here for future extensions of NestingXML.

```
<solutions>
    <solution>
        <placement idBoard="board1" idPiece="piece1" x="0.0" ↩
y="0.0" angle="0.0" mirror="none" />
        <placement idBoard="board1" idPiece="piece2" x="1.0" ↩
y="1.0" angle="0.0" mirror="none" />
        <placement idBoard="board1" idPiece="piece2" x="4.0" ↩
y="0.0" angle="0.0" mirror="none" />
        <usagePercentage>15</usagePercentage>
    </solution>
    ...
</solutions>
```

Listing 6: Example of the fifth section of a NestingXML file

6 An extended API

As with the older file formats described in section 3, we have also developed a Java API for the NestingXML format. Because we intend to maintain backward compatibility, some code from the previous API has been integrated to enable an easy transformation between formats. The tools developed for the previous API have been enhanced for the new one.

In the future, it may be necessary to add new features to the XML format. To use the new features in tools and solvers, they will have to be available in the API. We expect that such extensions to the API can be smoothly added, as the main concepts of the application domain are already captured in the current API. New features will account for parsing and handling new elements or attributes. This allows a steady evolution of the development environment.

As already stated, a NestingXML file may contain complex information regarding the problem, such as rotation of the pieces and multi-holed boards. While some research teams might be using such information, we are focusing our research on simpler problems. Therefore our API does not account for features we are not currently handling.

7 The Software Development Infrastructure

We are now going to take a look at an example solving process and the way the NestingXML file is transformed. Fig. 4 should be used along the description, as a way of getting a visual picture of the whole process.

In a first step we create a new NestingXML file where the problem is stored, including metadata for author and problem in the *nesting* root tag. To turn our file into valid XML we should define the header describing the version and namespace used for the document.

After creating the file's skeleton, we need to insert the formal description of the nesting problem. This information is inserted right after the information added in the previous step. Because both boards and pieces are sets of polygons, we should also add those to a specific section near the bottom of the NestingXML file. In Fig. 4 we can see arrows coming off this step's preview and pointing to NestingXML sections 2 and 4. Section 2 represents the place where the boards and the pieces are declared in the NestingXML file. Section 4 refers to the polygon description section.

In the third step of the process, we do some intermediate computations which will be used by some of our solvers. These data should also be stored for later usage. The intermediate computations section in the NestingXML file is represented by number 3 in Fig. 4. Note also that section 4 is also pointed by this process because some of the computation generate new polygon information.

The fourth and final step of our example process is the resolution of the generated problem instance. The solver retrieves the problem's data and intermediate computations from the proper location in the input NestingXML file. It appends the computed solution to the last section of the file where other solutions may already be present.

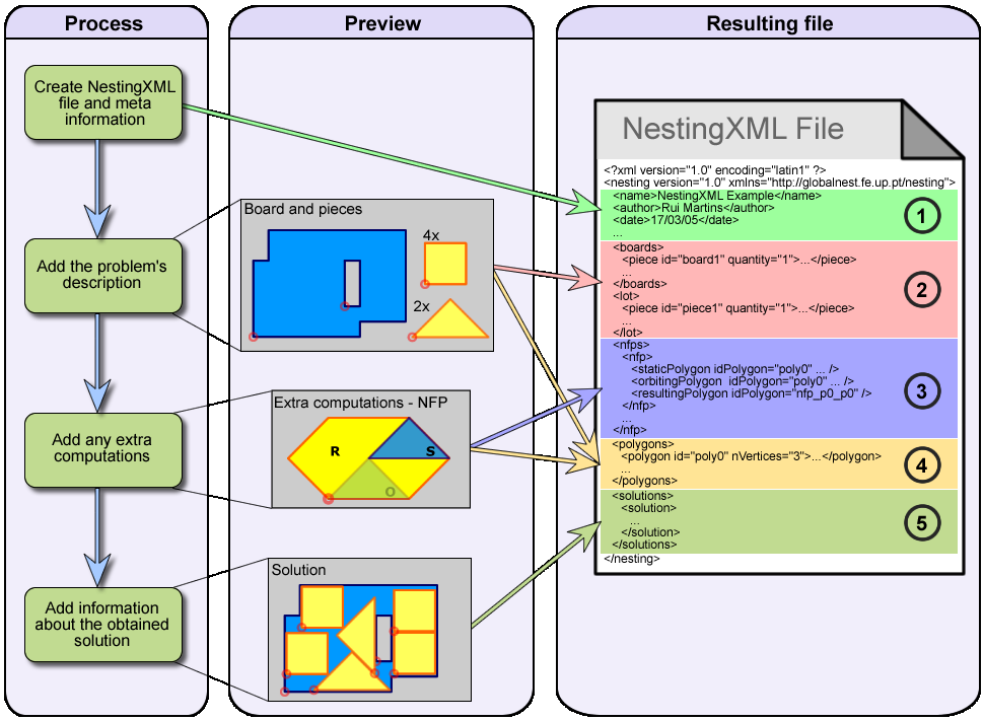

Fig. 4. Process example

8 Conclusions and Ongoing Work

The work reported here concerns part of the infrastructure put together to support a research project on Constraint Programming methods and solutions for a class of combinatorial problems. Lines of research such as this one extend for significant periods of time—the current project builds on results from work done since 1997. This kind of work typically involves the collaboration of several people, the consolidation of project results for building new approaches and the inclusion of new collaborators at a regular pace. In this context a dependable development environment is an important issue.

Researchers working on cutting and packing problems have collected a significant number of benchmark problems, and new ones are proposed when new techniques require illustration. Our project environment has shown that developing new approaches leads to the production of intermediate results that we must organize to be used in the problem-solving process. The CortaBem application [5] has served to test a resolution environment and provide an easy interface for colleagues who would like to get acquainted with our work, but no easy way to interchange problem instances and solutions was available.

The flexible representation of the NestingXML dialect has proven essential for exploring the structure of the problem domain and for avoiding the custom parsing and systematic migration that would otherwise be required.

The process infrastructure illustrated in Section 7 is being further developed in the sense of automating the repeated execution of sequences of operations on the input data to solve a problem. Data representation with NestingXML makes some other useful products simple to define. We are currently exploring the simplification of complex input shapes and the transformation of the output representations into standard graphic formats that can be used for rendering purposes [9].

9 Acknowledgments

We thank our colleagues José Fernando Oliveira and António Miguel Gomes for contributing with their expertise on the problem domain for defining the NestingXML dialect and for many useful discussions. João Paulo Mendes and Hugo Almeida, members of the GLOBALNest team, have contributed the first Java API and were also strongly engaged in the definition of NestingXML.

We thank Helmut Simonis, consultant for the GLOBALNest project, for the emphasis given to the project development environment and for his insights on the goals of the whole line of research.

References

1. Carlsson, M., Ottosson, G., Carlson, B.: An Open-Ended Finite Domain Constraint Solver. In Glaser, H., Hartel, P., Kucken, H., eds.: Programming Languages: Implementations, Logics, and Programming. Volume 1292 of Lecture Notes in Computer Science., Southampton, Springer-Verlag (1997) 191–206
2. Ribeiro, C., Carravilla, M.A., Oliveira, J.F.: Applying constraint logic programming to the resolution of nesting problems. Pesquisa Operacional **19** (1999) 239–247
3. Carravilla, M.A., Ribeiro, C., Oliveira, J.F.: Solving nesting problems with non-convex polygons by constraint logic programming. International Transactions in Operational Research **10** (2003) 651–663
4. Ribeiro, C., Carravilla, M.A.: A global constraint for nesting problems. In Régin, J.C., Rueher, M., eds.: CPAIOR. Volume 3011 of Lecture Notes in Computer Science., Springer (2004) 256–270
5. GLOBALNest: GlobalNest Website (2005) `http://192.168.102.63/~globalnest/`
6. Mendes, J.P., Almeida, H., Ribeiro, C., Carravilla, M.A.: An Evaluation Infrastructure for Nesting Problems. Technical report, INESC-Porto (2005)
7. ESICUP: EURO Special Interest Group on Cutting and Packing (2005) `http://www.apdio.pt/sicup/`
8. GLOBALNest Project Team: NestingXML Schema (2005) Disponível em `http://globalnest.fe.up.pt/nesting/nesting.xsd`
9. Gonçalves, B., Baldaia, N., Cerqueira, N., Martins, R.: NestingXML Visualization and Geometric Manipulation Tools. Technical report, INESC—Porto (2006)