# A Test Bed for Manufacturing Planning and Scheduling: Discussion of Design Principles

**Claude Le Pape**

ILOG S.A.
9, rue de Verdun
F-94253 Gentilly Cedex
clepape@ilog.fr

**Abstract**

The development of a public test bed for manufacturing planning and scheduling brings a lot of design issues, along several dimensions: the content of the test bed, the testing protocol, the format of the data, and the management of the results obtained. The objective of the present paper is to discuss these issues and explain the rationale behind the choices that have been made in *MaScLib*, a library of combined manufacturing planning, batching, and scheduling problem instances.

## Introduction

Manufacturing rough plans and detailed schedules must establish a compromise between antagonistic optimization criteria: meet as much as possible of the established and expected customer demand; deliver customer demands on time; reduce production costs (processing costs, resource usage costs, setup costs, cleaning costs); keep little value in inventories without risking starvation, etc. Usual practice is often very unsatisfactory in this respect. For example, rough plans are often built by averaging setup times and costs, even when variations are significant; production batch sizes are determined by economical considerations, with no respect to the distribution of demand due dates; detailed schedules are built and maintained under fixed resource availability and production-order-to-demand pegging assumptions. In many cases, globally sub-optimal results are obtained, due to the fact that at some step in the process an important optimization criterion and the associated constraints have been ignored.

New integrated planning, batching, and scheduling models, aimed at ensuring greater consistency between the planning, batching, and scheduling steps, and, as a result, at delivering better plans and schedules, must be developed. Focusing on such integrated models has three advantages: the approximations needed at each problem-solving step are made explicit and the related issues clearly identified; global problem-solving methods can be designed and implemented towards a unique well-defined objective; these methods can be evaluated against global problem instances. The underlying optimization problems are, however, extremely difficult, and necessitate both theoretical and experimental research. To motivate and guide such research, the *ILOG Plant PowerOps* development team decided to introduce and maintain an organized set of problem instances, *MaScLib*, first focused on manufacturing scheduling [Nuijten et al., 2004], and later extended to the combination of planning (deciding which products to manufacture and which demands to deliver), batching (creating production orders and organizing the flow of materials within the factory), and scheduling (allocating time and resources to the activities composing each production order) [Le Pape & Robert, 2007].

The objective of the present paper is to discuss the main issues encountered during the design and development of *MaScLib*, and explain the rationale behind the choices that have been made. The issues are classified in four categories:

- **Content:** the optimization problem proposed to the scientific community.
- **Protocol:** the ways in which problem-solving methods shall be applied to the proposed problem instances and evaluated.
- **Format:** the type and organization of files describing the problem instances.
- **Result:** the exposure of results obtained by different researchers on different instances.

Each of these categories is discussed in turn, in an attempt to explain how the overall goal behind the development of *MaScLib* influenced the choices we have made.

## Content

In the design and development of industrial optimization applications, one major concern is that the optimization algorithm must be robust. By "robust," we mean not only that the algorithm must provide "good" solutions to problem instances of different size and numerical characteristics, but also that the algorithm must continue to work well when constraints are added or removed. As mentioned in [Chabrier et al., 2004], this expectation is heightened in constraint programming as its inherent flexibility is often put forward as its main advantage over other optimization techniques. In practice, it has important effects on the reinforcement of problem formulation, search management, the advantages of parallel search, the applicability of different optimization techniques including hybrid combinations, etc.

The benchmark problem suites that are used by the academic community generally do not reflect this requirement for robustness. They are on the contrary typically focused on simple (to state) complex (to solve) problems, e.g., the Job Shop Scheduling Problem, as defined in its decision form in [Garey & Johnson, 1979]. Such simple but complex problems precisely present two advantages from an academic perspective:

- They are easy to state, to reason about, to teach and talk about. As a result, their fundamental nature is understood by many potential contributors.
- Their complexity is due to a small set of features that are often encountered in real life. As a result, progress on these features has potential impact on many applications.

Yet there is no such thing as, for example, the Job Shop Scheduling Problem, so the impact of new research results remains hard to evaluate, as long as these results are not actually applied to real cases.

For a benchmark problem suite to be the most profitable, some equilibrium must be established between the simplicity of the proposed problem and the presence of side constraints. A too detailed real application is not likely to grasp the interest of the research community and lead to widely applicable results. A too simple problem is more likely to lead to an abundant scientific literature, but in the end the really important results still have to be sorted out, at their expense, by application developers.

Several industrial researchers have tried to provide benchmark problems based on a simple core problem and extensions. For example, Caseau and Kökény [1998] provide both "B" and "G" instances of an inventory management problem. Compared to the "B" instances, the "G" instances include a side constraint: the pieces of equipment in inventory must be regularly maintained in a maintenance shop of limited capacity. In addition, the purchase of new equipment and the substitution of different types of equipment may or may not be allowed. Similarly, the ROCOCO benchmark problem suite [Bernhard et al., 2002] [Chabrier et al., 2004] includes six optional constraints, leading to 64 different formulations of 21 basic instances. Even though a researcher can, in the beginning, focus on the simplest version of the problem, the ultimate goal is to develop problem-solving methods that are robust to the introduction of the side constraints.

In a similar but systematized spirit, *MaScLib* proposes different categories of instances. The simplest category, "NCOS", contains instances in which resources are subjected to "No Calendar", i.e., are available at all times, and production recipes are limited to "One Step". In addition, all the production recipes require a unique resource of capacity 1. More complex instances are gathered in other categories:

- NCGS: No Calendar, General Shop, in which several activities, linked by temporal constraints, are required to manufacture final products.
- BROS: BReaks, One Step, in which the resource is subjected to breaks and production activities can start before and end after breaks under given conditions.
- BPOS: Breaks and Productivity, One Step, in which the resource is not only subjected to breaks but also to variations of its productivity over time.
- PM_NCOS: Parallel Machines, No Calendar, One Step, in which the unique resource is a group of equivalent machines, enabling the performance of several activities in parallel.
- STC_*, where "STC" stands for "Setup Times and Costs", in which sequence-dependent setup times and costs must be considered.
- MM_ STC_*, where "MM" stands for "Multiple Modes", in which activities can be performed in different conditions on different resources.
- UNP_*, where "UNP" stands for "UNPerformed", in which it is possible to (i) leave some customer demands unsatisfied and (ii) leave some activities unassigned, both at a given cost.

In addition, instances are classified according to the optimization criteria under consideration:

- Beside production (and possibly setup, non-delivery, and non-performance) cost, the "a" and "b" instances include tardiness costs. The "a" instances are such that different production orders have the same importance in terms of tardiness, while in "b" instances some production orders are more important than others.

- The “c”, “d”, and “e” instances include earliness and storage costs as well. The “c” and “d” instances are such that it is always better to deliver a demand early than to store the corresponding product. In addition, in “c” instances, earliness and tardiness costs are such that different production orders have the same importance in terms of earliness and tardiness. The “e” instances correspond to the more general case, in which a compromise between storing products and delivering customers early must be established.

Finally, each instance can be used either for scheduling only (relying on a set of production orders provided in the data file) or for planning, batching, and scheduling (by ignoring these production orders). In the end, the proposed problem varies from a very simple but complex scheduling problem “$1 \mid r_i,d_i \mid \Sigma_i T_i$” to a planning, batching, and scheduling problem, with multiple modes per activity, setup times and costs, arbitrary temporal constraints, and/or calendars. The hope is that researchers, starting from the most classical problem, will get interested in extending their problem-solving methods to side constraints that are often met in industry.

## Protocol

Once the content and overall organization of the instances is decided, it is necessary to define how the instances shall be used, depending on the main purpose of the problem suite. Different objectives lead to different sets of instances, different constraints on the testing protocol, and different ways to evaluate the results.

When the goal is to test how different methods behave in the case of a particular manufacturing plant, the set of instances shall be designed to converge on the problems actually met **in this plant**, the execution constraints (e.g., limited CPU time) shall be dictated by the use cases of planning and scheduling software **in this plant**, and the overall evaluation shall be representative of the money that will be gained by using the software **in this plant**. This does not imply that robustness to differences in problem size, numeric characteristics, and side constraints, is not an issue. Indeed, customer demand may be seasonal, some constraints may apply only in night shifts, etc., entailing important differences from a problem instance to another.

On the opposite, when the objective is directed towards the evaluation of generic software, aimed to be applicable in many contexts, the set of instances shall be as diverse as possible, the execution constraints shall be defined according to a general feeling of what is acceptable in practice, and the overall evaluation shall be directed towards robustness rather than towards pure performance.

In the case of *MaScLib*, this led to the following choices:

- The set of instances includes both simplifications of real problems encountered by ILOG customers, random instances provided by academic researchers, and instances built to be difficult.
- For each instance, the goal is to provide the best possible solution in a given time limit, defined with respect to the problem size. For a given instance and a given problem-solving method, the result is normalized by dividing the cost of the solution obtained in the given CPU time by the cost of the best known solution.
- The overall evaluation over a set of instances is obtained as the average of these normalized results minus 1. This evaluation is often called Mean Relative Error MRE = average(cost-of-solution-found / cost-of-best-solution-known) – 1.

As said, these choices are really geared toward evaluating robustness: the instances are very different one from the other; the normalized result on a given instance is high whenever it is known that the solution found is much more costly than it could have been; and very bad solutions have a high impact on the MRE. The drawback of this protocol, however, is that it relies on the cost of best known solutions, which change all the time as new results are reported. Ideally, it would of course be better to rely on the cost of optimal solutions, but for large instances these costs are unknown. An idea would be to rely on reference costs found once and for all by a particular problem-solving method on a particular computer, but then instances on which the reference solutions are bad would contribute less to the overall evaluation, which would be very unsatisfactory.

## Format

Providing a “good” problem and a “good” protocol is not sufficient. The format in which problem instances are provided shall be simple enough to read; otherwise the instances will simply not be used. Needless to say, it is quite difficult to design a simple format enabling the description of problems with complex side constraints.

For *MaScLib*, a relational format was adopted. Each instance is described by a set of tables: a table for global characteristics of the instance, a table for resources, a table for production recipes, a table for activities composing recipes, a table for customer demands, a table for due dates, etc. Some tables, or some columns in some tables, are optional, depending on the instance category. For example, in the NCOS category, all resources have capacity 1: the CAPACITY column of the RESOURCE table is omitted.

The advantage of such a format is its natural extensibility. A researcher can write a reader for NCOS instances and then extend it to PM_NCOS instances. Globally, it however appeared that the first reader takes long to develop. To overcome this drawback, the simplest instances are also provided in simpler textual format, but then the researcher has a gap to fill to move from these simple instances to more complex instances. It is unclear how a better solution to this issue could be reached.

## Results

Once the data sets are distributed, results arrive. It is then necessary to decide which results to keep. *MaScLib* currently consists of about 500 instances. Should twenty researchers provide results, each with ten variants of a given problem-solving method, we get about 100.000 figures to manage. Either some infrastructure must be built to manage these results, or dominated results shall not be kept and distributed with the problem set.

For many benchmark problem suites, only the best-known lower and upper bounds are maintained on a publicly available page. This is indeed the strict minimum. However, when the main objective is to test the robustness of problem-solving methods, best-known lower and upper bounds are clearly insufficient. Indeed, it could be that a given problem-solving method M provides a lot of best-known solutions, but also provides catastrophic solutions on a significant number of instances; while another method N is not so good at reaching the best-known solutions, but is on average much better than method M.

For *MaScLib*, the following policy is currently adopted:

- The results of a problem-solving method that is proven to be among the three best methods on a significant subset of instances are systematically kept.
- The results of any other problem-solving method are eliminated unless this elimination would make a best-known lower or upper bound disappear from the result base.

Unfortunately, the outcome of this policy depends on the order in which results are received and eliminated, which is not really satisfactory. For example, if method M provides the best solution for instances A and B but not for instance C, method N provides the best solutions for instances A and C but not for instance B, and method P provides the best solutions for instances B and C but not for instance A, either M or N or P can be eliminated. At a given time, ties can be broken by looking at the overall robustness of the different methods or giving a preference to the methods documented in the literature, but in general there is no fully satisfactory policy to make decisions in this respect.

## Conclusion

The development of a public test bed for manufacturing planning and scheduling brings a lot of design issues, along several dimensions: the content of the test bed, the testing protocol, the format of the data, and the management of the results obtained.

Even when the objective of the test bed is clearly defined, usability and complexity issues along these four dimensions are encountered. Academic researchers and application developers should discuss these issues, with the common objective of improving the benchmark problem suites made available to the community and, as a result, the relevance of the research conducted toward a better resolution of these problems.

## Acknowledgements

I wish to thank:

- My colleagues engaged in the development of the optimization engines of the *ILOG Plant PowerOps* tool — Thomas Bousonville, Julien Briton, Filippo Focacci, Daniel Godard, Emmanuel Guere, Arnaud Leforestier, Xavier Nodet, Wim Nuijten, Frédéric Paulin, and Anna Robert — for their contribution to the gathering of instances, the design of the data format, and the organization of the overall test set.

- The developers of the *ILOG Plant PowerOps* graphical user interface — Dominique Bréheret and Ali Sadeghin — and testing framework — Philippe Charman, Virginie Grandhaye, and Marie-Laure Leroux. These components have been used hundreds of time to check and register solutions, thereby contributing to the elimination of bugged solutions and to the maintenance of accurate (and so precious!) tables of results.

- Philippe Baptiste and Francis Sourd for the early discussions we had on the *MaScLib* topic, as well as all the researchers who, in addition to the above-mentioned, already provided us with results or indirectly contributed — Emilie Danna, Renaud Dumeur, Antoine Jouglet, Safia Kedad-Sidhoum, Philippe Laborie, Laurent Perron, Jérôme Rogerie, Ed Rothberg, and Ruslan Sadykov.

## References


Bernhard, R.; Chambon, J.; Le Pape, C.; Perron, L.; and Régin, J.-C. 2002. Résolution d'un problème de conception de réseau avec Parallel Solver. In *Onzièmes Journées Francophones de Programmation Logique et Programmation par Contraintes*, Nice, France (in French).

Caseau, Y.; and Kökény, T. 1998. An Inventory Management Problem. *Constraints* 3(4):363-373.

Chabrier, A.; Danna, E.; Le Pape, C.; and Perron, L. 2004. Solving a Network Design Problem. *Annals of Operations Research* 130:217-239.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company.

Le Pape, C.; and Robert, A. 2007. Jeux de données pour l'évaluation d'algorithmes de planification et ordonnancement. In *Cinquièmes Journées Francophones de Recherche Opérationnelle (FRANCORO) et Huitième Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF)*, Grenoble, France (in French).

Nuijten, W.; Bousonville, T.; Focacci, F.; Godard, D.; and Le Pape, C. 2004. Towards an Industrial Manufacturing Scheduling Problem and Test Bed. In *Proceedings of the Ninth International Conference on Project Management and Scheduling*, Nancy, France.