How does constraint technology meet industrial constraints ?

Philippe Gérard Simon de Givry Jean Jourdan
Juliette Mattioli Nicolas Museux
Pierre Savéant
PLATON*Group
THALES Research & Technology France
Domaine de Corbeville
91404 Orsay cedex FRANCE
{philippe.gerard, simon.degivry, jean.jourdan}@thalesgroup.com
{juliette.mattioli, nicolas.museux, pierre.saveant}@thalesgroup.com

August 31, 2001

Abstract

This paper describes our experience for introducing Constraint Programming in operational systems of THALES[1]. Many of them are on-board and real-time systems. We give the requirements of a technology for developing real-time systems with combinatorial optimization capabilities. We present the constraint technology that has the ability to fulfill most of the requirements. The current limitations of state of the art Constraint Programming solvers yield to research works that are described. Finally, we give an overview of the constraint technology at THALES.

Introduction

Industrial motivations

In many areas of industry and business, the optimization of resource allocation is becoming crucial. Services and products must fulfill the increasingly specific needs of customers. What is the best way to allocate costly resources and improve competitiveness? Re-directing resources to handle shortages or rearranging the sequence of activities can lower costs, reduce waste, shorten cycle time and speed up delivery time. Planning and scheduling systems will provide a competitive advantage on the market.

Planning determines the sequence of tasks (the plan) and scheduling decides on the choice of specific resources, operations and their timing to perform the tasks (the schedule). Planning and scheduling problems are highly complex and involve making many decisions, including which set of resources are to be assigned to each demand. Each decision has a limited number of possible alternatives and must satisfy operational constraints. These problems belong to a class of problems called

*PLAnning opTimization & decisiON making
[1] Ex Thomson-CSF.

combinatorial optimization problems. Most of them are computationally difficult and require considerable development time and expertise, in both the application domain (for modeling) and algorithm design (for solving).

As on-board real-time systems evolve from hardware to software solutions, their resource management functions require more sophisticated algorithms dealing with complex planning and scheduling problems. Such capabilities are more and more required in applications areas such as C3I , sensor management, weapon allocation or deployment, telecom resource management, mission planning ...

The following section gives the requirements of a technology for developing real-time systems with combinatorial optimization capabilities.

Requirements of a suited technology

Real-line systems have a difficult operational context compared to classical off-line systems. They must react continuously to follow changes of an external environnement. The response time can be very short. A valid plan/schedule has to be provided at each deadline. And there is also a memory space limit.

Moreover, in several applications, the life cycle of the system can be very long. The system has to be maintained during a long period, over twenty years in the case of a Defense system. During this period, there will be many retrofitting of the system. The system has to be robust to the addition of new functionalities and should derive benefit from platform evolutions (faster computers).

The construction and the maintenance of complex large scale software systems, including heterogeneous, multi-components, multi-functions applications become significant in the software marketplace demand. One way to solve a combinatorial problem is to develop a specific algorithm completely from scratch. This has the advantage of exploiting all the problem-specific knowledge. The disadvantage is that it can take a long time to develop a really satisfactory solution. System maintenance is dependent on the software author's particular way of thinking. Moreover, if the problem changes in subtle ways, the algorithm must be scrapped and reinvented, that can be very costly.

The target technology must have the following properties:

- Time and space guarantees,
- Robustness to retrofitting of the system,
- Modularity property through a safe incremental programming approach (useful for complex and large systems involving several engineers and for heterogeneous systems),
- Reuse property through a high level of abstraction because reading code is a skill that not even software experts do very well. It is difficult to fully understand something which is thousand lines of code, without making a significant investment. This property allows:
  - an easy reasoning on the fundamental properties of the system,
  - the reduction of the gap between specification and code,
  - and the opportunity to capitalize at the model level,
- Traceability property through a formal approach enabling the refinement of requirements.

Note that some industrial companies, for confidentiality or market protection reasons, may prefer to keep in house the entire development process and its know-how. The technology has to be mastered in house.

In the following part, the constraint technology is presented as a key technology for modeling and solving real world combinatorial problems.

1 Constraint Programming for modeling and solving real world combinatorial problems

Traditional modeling languages are particularly strong in mathematical programming application (e.g. linear or integer programming). Some of the combinatorial problems are naturally expressed using algebraic or logic notations. However, a number of combinatorial applications, such as job-shop scheduling and resources allocation problems are out-side the scope of these languages. On the one hand, these problems are rarely expressed naturally using only algebraic constraints. On the other hand, it is often important in these applications to guide the solver toward solutions by specifying an appropriate search procedure or a domain heuristic.

In the following sections, the Constraint Programming is introduced. A clean separation between the modeling and the solving parts is shown. The underlying mechanisms are briefly presented. Finally, an industrial analysis of the fundamental properties of the constraint technology is given.

A longer introduction of Constraint Programming can be found in [Wallace, 1995; Barták, 1998]. Reference books are [Van Hentenryck, 1989; Tsang, 1993; Marriott & Stuckey, 1998].

1.1 A short history of Constraint Programming

Constraint Programming has a long tradition in Artificial Intelligence. The Constraint Satisfaction Problem (CSP) was first formalized and studied in vision research for solving line-labeling problems during the early seventies. This class of problems is important because any combinatorial optimization problem can be represented in this paradigm.

But Constraint Programming as a language comes from the integration of consistency techniques in Logic Programming in the mid-eighties. Logic Programming brought theoretical foundations and the language point of view thanks to its built-in nondeterminism. Consistency techniques and Operations Research provided efficient implementation methods mainly based on graph theory. Constraint Logic Programming (CLP) emerged in the mid 80's, from European and American labs. Most influential was the work of the CLP(R) team at IBM, who coined the word CLP and set its theoretical foundations. Prolog II was then the only instance of promising scheme. A definite interest in finite domains arose from the work on the CHIP concept (Constraints Handling in Prolog) at the joint Bull-ICL-Siemens European Computer-Industry Research Center. Both CHIP and Prolog III later turned into products.

The first startups appeared in early 90's: Cosytec, Axia, Ingenia and Ilog. Several academic solvers were also developed among them SICStus in Suede, Eclipse at the Imperial College Park (UK), Oz, CHR and IF/Prolog in Germany, B-Prolog in Japan, clp(FD) and ncl at INRIA (France) and many more. Most of corporate research laboratories of industrial companies have developed also their own solvers, for instance THALES (ex. Thomson-CSF), France-Telecom, Daimler, etc...

Cosytec is the godfather of the so-called global constraints which allow to easily specify and solve complex problems in areas like planning, scheduling, sequencing, packing, configuration and routing. Note that the Constraint solving paradigm came out from Logic Programming to integrate general purpose languages such as Ilog SOLVER, the world leader, which is not Prolog-based.

The industrial impact started in 1993 with the first operational applications. Today the technology has spread out many markets such as manufacturing, transportation, telecommunications and building.

1.2 Constraint Programming as a modeling language

"Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it." [E. Freuder, Constraints, April 1997]

The essence of Constraint Programming is based on a clean separation between the statement of the problem (the variables and the constraints), and the resolution of the problem (the algorithms).

The basic way to express and model a combinatorial problem is with:

- Variables, which are defined with a specified domain and should allow to model the possible decisions one can take for determining a solution to the problem.
- Constraints, describing all the restrictions on variables and all relations between these variables that must be satisfied for the given problem. The important feature of constraints is their declarative manner, i.e., they specify what relationship must hold without specifying a computational procedure to enforce that relationship.
- Objective function(s), specifying what must be optimized. An objective function is usually expressed as a function of (a part of) the decision variables or related to the variables in a more implicit or complex manner.

In order to partially overcome the lack of expression and deductive power of conventional Constraint Programming languages, global constraints have been introduced. One of the main advantages of global constraints is to take into account more globally a set of elementary constraints [Beldiceanu & Contejean, 1994]. It can lead to substantial speedups because they allow the use of propagation algorithms based on mathematical properties of constraints using the notions from graph theory or mathematical programming.

Most of the constraint realizations we have worked on are built from two types of models:

- Extension models. This is the usual model that most people build using constraint technology. For this type of model we can represent in extension all the elements involved in the model, for instance the set of tasks and resources of the scheduling problem.
- Comprehension models. We develop such models when it is impossible to represent all the elements of a model. This occurs in three cases for continue phenomena, for dynamic aspects and sometimes for scaling up issues. The amount of effort and the methodology required to build comprehension models is not comparable to those to build extension models.

The difficulty in building comprehension model is to preserve the interactions with other models. Otherwise, it is not worth using Constraint Programming. The way we set up comprehension models are based on an approximation process, which consists in finding out a set of sufficient conditions. In most of the case studies, this modeling approach has preserved the use of constraint technology. Sometimes the constructed sufficient conditions have led to too rough approximations. In such cases the solution lies in the combination of several resolution techniques, in particular, non linear solving methods [Hentenryck *et al.*, 1997]. Unfortunately sometimes, it is still an open issue and even worth impossible to achieve.

1.3 How does Constraint Programming work?

Computation behind Constraint Programming is based on the cooperation of two processes: local consistency maintenance on one hand and problem decomposition on the other one.

Local consistency is achieved by filtering algorithms which remove combinations of values that cannot appear in the solution under construction. This process is also called constraint propagation. If all bad combinations were discarded then solutions would be reached; but this problem is NP-hard and polynomial algorithms are available only for 2 or 3 variables, that is why consistency is not complete. With Finite Domain Constraint Programming variables range over finite domains which are integers or intervals over integers and each constraint defines its own filtering algorithm.

Decomposition is based on committing choices to divide the problem into sub-problems until they get simple enough to be solved in a straightforward way. It is usually implemented by a tree-search algorithm. For optimization problems the algorithm is extended to a so-called branch-and-bound algorithm which principle is to avoid to search in sub-trees for which a proof has been made that no better solution can appear. This way of enumerating could be sufficient but its complexity is obviously exponential. That is why in practice solutions are reached by the interleaving of the two processes: the propagation engine is requested each time a choice is committed.

1.4 Software engineering properties provided by the constraint technology

Constraint programming appears to be a very appealing technology for most of the combinatorial optimization problems. The interest of industrials in using this technology is mainly based on its software engineering capabilities.

Indeed it is well known that the constraint technology can reduce the development time of the planning/optimization applications by orders of magnitude and provides several others advantages from industrial prospects, which have not been too much highlighted and discussed so far. The analysis below does not pretend to be exhaustive but is the first attempt to summarize what we have learned from an industrial viewpoint during 10 years trying to introduce the constraint technology in operational systems of THALES:

- Thanks to its declarative nature this technology appears more to be a modeling technology than a programming technology. Indeed, application engineers don't have to care about the order the constraints are set in the system and furthermore they don't have to worry about the interactions with later added constraints. This seems trivial but provides a safe incremental programming approach, which becomes fundamental for designing complex and large systems involving several engineers.

- Thanks to its compositionality property, this technology provides a nice way of representing multiple redundant models or complementary models. The generic modeling framework, presented in [Jourdan, 1995a; 1995b] introduces the notion of redundant dual models. This modeling approach goes a step forward the notion of redundant constraints and leads to important improvements in term of scalability [Cheng *et al.*, 1999].

- Thanks to its high level of abstraction, it helps in reducing the gap between the specifications and the realizations. The debugging process is made easier, errors are conceptual errors instead of programming errors. A constraint program consists in a system that reasons on the fundamental properties of the system, which are not far from its specifications. For industrials in charge of huge programs, requirement traceability is very important and even contractually required. Furthermore, the high level of abstraction provided by Constraint Programming offers the opportunity to capitalize at the modeling level rather than at the programming level, which yields to higher gain in productivity.

- Thanks to its formal approach, it helps in refining the specifications themselves. The mathematical essence of Constraint Programming imposes to transform specifications into equations (in a broad sense). Consequently a lot of imprecise or fuzzy requirements are removed.

However the constraint technology is faced with difficulties which are explained below. Research directions are given to overcome the limitations. A French research project is presented. It aims at producing a real-time Constraint Programming framework.

2 Toward a real-time Constraint Programming framework

Current limitations of state of the art Constraint Programming generic solvers are:

- Due to the complexity of constraint reasoning, it can be difficult to debug a constraint or a model. There is a need for an automatic explanation of the algorithm working.

- There is a lack of abstraction for expressing search algorithms. In Constraint Programming, one has to program its search algorithm rather than specify it.

- Of course, combinatorial problems remain difficult to solve. In limited time, the results of a search may have a high variability in term of quality. The usual search algorithm in Constraint Programming solver, based on a complete search tree, does not fit to a limited time context. Moreover if the search algorithm is stopped before its normal end, there is no information on the quality of the results. This is a problem if we want to validate the results of a real-time optimization system.

- There is no guarantee on space and time.

- The solver does not take into account the time contract. Therefore it will not take advantage of the evolutions of the platforms.

A French project, called EOLE[2] [EOLE web, 2000], aims to overcome these limitations. This project is a RNRT[3] project supported by the French Research Office. It began on April 2000, and it will be finished at the end of 2002. The work plan is divided into six sub-projects. The first one consists in establishing foundations, concepts and perimeter of use of an optimization framework for on-line combinatorial problems. This sub-project should supply with formalisms to express algorithms the behavior of which answers the defined characteristics. The second sub-project consists in offering primitives implementing the previous formalism. Finally, it aims at realizing the integration of these tools and libraries in an object-oriented framework. The following three sub-projects allow the validation through a set of benchmarks led on three experimental applications (ATM network management, network reconfiguration and frequency assignment). And the last sub-project attempts to define an industrialization plan of the developed technology.

The proposed approach consists in combining:

- the flexibility of modeling of Constraint Programming,
- the efficiency of hybrid algorithms,
- the adequation to real-time operational constraints of anytime algorithms[4] and parameterized search algorithms,
- and the capitalization capabilities of optimization frameworks, based on high level search primitives, templates of search and code generator.

The following sections give an insight of these approaches.

2.1 Scalability to large scale combinatorial problems

They are two orthogonal approaches to face large scale combinatorial problems.

The first approach concerns the modeling phase. A refinement process of the model will enhance the constraint propagation efficiency. The goal is to find the best tradeoff between the propagation computation time and the resulting pruning of the search space. The addition of redundant models will also improve the stability and quality of the results [Jourdan, 1995a; Cheng *et al.*, 1999].

The second approach concerns the solving methods. There are two kinds of methods: tree search methods and local search methods. Tree search methods use a problem decomposition scheme for producing solutions [Kumar, 1992]. Without any time limit, these methods are complete. In a limited time context, the notion of completeness is given up, and the notion of non systematic search is introduced [Harvey & Ginsberg, 1995; Harvey, 1995]. The main idea is to follow the heuristics in a more cleaver way, by trying to diversify the search progressively.

Local search methods transform a (complete) assignment of the decision variables iteratively, by following a cost gradient and making a few random choices to escape from local minima [Aarts & Lenstra, 1997]. These methods are well adapted to a limited time context, but they do not benefit from the constraint technology.

Recent works are around the hybridization of tree search and local search methods. Hybrid search methods are divided into three categories:

[2]Environnement d'Optimisation en-LignE dédié Télécom / On-Line Optimization Framework dedicated for Telecom domain

[3]Réseau National de Recherche en Télécom

[4]An anytime algorithm must be able to supply at any time a useful solution, even if it is not optimal or if its optimality was not established

- embedded hybridization combines the basic mechanisms of tree and local search methods during the search (e.g. [Mazure *et al.*, 1996]),
- hybridization by composition clearly separates tree search and local search methods during the search (e.g. [Wallace, 1996]),
- and methodological hybridizations where a tree search or a local search method is used to solve a distinct but complementary problem to help the search on the initial problem (e.g. [Kask & Dechter, 1996]).

An attempt to insert local search methods in Constraint Programming is a very important research direction, see works from [Pesant & Gendreau, 1999].

2.2 Taking into account time and space limits

In a few pathological cases, tree search methods and constraint propagation can use a very large amount of memory. The computing depth of these recursive algorithms has to be bounded in order to have a guarantee on space. The time guarantee is obtained by an operating system alarm. And for very short computation time, partial solutions are returned, built in a deterministic way.

An important issue is to exploit the knowledge of a time contract for the search. This knowledge allows to control the search in order to produce better quality results.

In the case of a hybrid search algorithm by composition of several tree/local searches, the time contract will be divided among the different searches [Horvitz & Zilberstein, 2001]. Each search will have its own deadline. The way the time contract is divided depends on a given temporal policy. For a local search method, its complexity is easily controlled by its number of iterations. For a tree search method, its complexity can be controlled by parameterizing the method. For that, we introduce special parameters dealing with incompleteness of the search [Givry *et al.*, 1999]. The parameters are automatically tuned, by using time estimation tools [Lobjois & Lemaître, 1998].

2.3 High level primitives for designing search algorithms

Besides the modeling phase there is still a complex programming phase for designing an accurate and efficient search algorithm. The next challenge to be addressed by the constraint community will be to extend the modeling approach to the algorithmic phase. Recent works, like SaLSA [Laburthe, 1998] and search building blocks in OPL [Hentenryck, 1999; Perron, 1999], go in that direction.

Our interest is to go further in the concept of concurrent constraint model-based programming [Jourdan, 1995a] defining an infrastructure which will make the modeling task within the reach of a non constraint expert user. The reuse of models is a major step toward that ambitious goal. To enhance flexible reuse, the models can be broken down into reusable components, containing also specific search methods and specific domain heuristics.

3 Overview of the constraint technology at THALES

Several constraint languages have been designed at THALES. The first one was Meta(F) [Codognet *et al.*, 1992]. The idea behind this meta constraint propagator, built on top of Sicstus prolog, was to say that the performance of the propagator itself was not so important comparing to the gain obtained by pruning the search space. At the end, Meta(F) was great to prototype applications but not to develop operational applications. Later we have designed CMeta, it was the first constraint language implemented as a full C library. The library includes C routines for the constraints and C routines for the implementation of non determinism search algorithms. The performances of CMeta occurred to be very interesting but it appeared very difficult to extent the language with new constraints, almost impossible to enable user defined constraints, and not easy to write search algorithms. From these two interesting experiences, the main learned lesson is, that it is important to use an efficient host programming language and to not sacrifice the high level of abstraction required by a good constraint language.

The new technology we are currently devising at THALES Research and Technology is built on this last remark. Our current constraint technology, called Eclair(c) [Laburthe *et al.*, 1998; OpenEclair, 2000; Eclair1.4, 2001], is based on the host language Claire(c) [Caseau & Laburthe, 1996]. Claire(c) provides most of the basic mechanisms necessary to design a constraint language. Moreover, Claire(c) is a very efficient source to source compiler which enables to not be stick with a specific target programming language. This last point answers another problem we have to face, it is to deploy constraint realizations in various technical contexts from embedded systems to large scale distributed systems where the zoology of programming languages is heterogeneous (Fortran, Ada, C, Java, ...). Consequently, we would like to be able to derive our models, written in the high level constraint language, on several usual target programming language.

The Eclair(c) acronym stands for Expressing Constraints with a Library of Algorithms for Inference and Resolution. Eclair(c) is a finite domain constraint solver over integers written in the Claire(c) programming language. The Eclair(c) library has an open architecture, which allows the user to add its own constraints in a straightforward manner. Actually, there is no difference between a user defined constraint and Eclair(c) constraints. Both are designed at the same level of abstraction. The current release of Eclair(c) [Eclair1.4, 2001] includes arithmetic constraints, global constraints and Boolean combinations. Eclair(c) provides a standard labeling procedure for solving problems and a branch-and-bound algorithm for combinatorial optimization problems. Programmers can also design easily their own non-deterministic procedures thanks to the dramatically efficient trailing mechanism available in Claire(c).

Eclair(c) has been designed for on-board real-time applications. Primitives are offered for the control of memory allocation and time bounding. Applications written in Eclair(c) have guarantees on time and space.

On top of Eclair(c) we are currently adding high level primitives for expressing hybrid search algorithms taking into account a time contract. A template mechanism is introduced which allows the reuse of anytime search methods.

Conclusion

Constraint Programming languages are among the very few new language proposals that are seeing industrial success while being based on a novel programming paradigm. Constraint Programming incorporates techniques from Mathematics, Artificial Intelligence and Operations Research, and it offers significant advantages in these areas since it supports fast program development, economic program maintenance, and efficient runtime performance, thanks to:

- a safe incremental programming approach,

- a high level abstraction which allows reasoning on the fundamental properties of the system, reducing the gap between specifications and code, and gives the opportunity to capitalize at the model level,
- a formal approach enabling the refinement of requirements,
- the combination of search and incremental constraint solving capabilities, and the relative efficiency of the resulting applications.

These characteristics are added to the strength in general-purpose symbolic processing of the underlying logic programming kernel on which they are often built. Constraints extend this kernel to numerical domains and beyond, offering a natural platform in which applications combining symbolic and numeric computing can be easily developed.

The following research directions will allow the current limitations of Constraint Programming to be overcome:

- For algorithm efficiency: hybridizations between tree search, local search and constraint propagation.
- For system adaptability: computational complexity, performance prediction and real-time monitoring of solution quality. Learning of search control strategies.
- For capitalization: high level primitives for designing search algorithms and domain frameworks.

Note that there is a workshop at the annual Constraint Programming conference (CP-2001, 1st December 2001) on on-line combinatorial problem solving and Constraint Programming (see http://www.lcr.thomson-csf.com/projects/www_eole/workshop/olcp.html).

References

[Aarts & Lenstra, 1997] E. Aarts, J. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, 1997.

[Barták, 1998] Roman Barták. On-line guide to constraint programming. http://kti.ms.mff.cuni.cz/~bartak/constraints/, 1998.

[Beldiceanu & Contejean, 1994] N. Beldiceanu, E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20(12):97–123, 1994.

[Caseau & Laburthe, 1996] Y. Caseau, F. Laburthe. Introduction to the claire programming language. Technical Report LIENS technical report 96-15, Ecole Normale Supérieure, 1996.

[Cheng *et al.*, 1999] B.M.W. Cheng, K.M.F. Choi, J.H.M. Lee, J.C.K. Wu. Increasing constraint propagation by redundant modeling: an experience report. *Constraints*, 4(2):167–1992, 1999.

[Codognet *et al.*, 1992] P. Codognet, F. Fages, J. Jourdan, R. Lissajoux, T. Sola. On the design of meta(f) and its application in air traffic control. In *Proc. of JICSLP workshop on constraint logic programming*, Washington, DC, 1992.

[Eclair1.4, 2001] PLATON, THALES Research & Technology, Orsay, France. *Eclair reference manual*, v1.4 edition, 2001.

[EOLE web, 2000] Eole project: On-line optimization framework for telecom. http://www.lcr.thomson-csf.com/projects/www_eole (in french), 2000.

[Givry *et al.*, 1999] Simon de Givry, Pierre Savéant, Jean Jourdan. Optimisation combinatoire en temps limité : Depth first branch and bound adaptatif. In *Proc. of JFPLC-99*, Lyon, France, 1999.

[Harvey & Ginsberg, 1995] William D. Harvey, Matthew L. Ginsberg. Limited discrepancy search. In *Proc. of IJCAI-95*, pages 607–613, Montréal, Canada, 1995.

[Harvey, 1995] William D. Harvey. *NONSYSTEMATIC BACKTRACKING SEARCH*. PhD thesis, Stanford University, March 1995.

[Hentenryck *et al.*, 1997] P. Van Hentenryck, L. Michel, Y. Deville. *Numerica: A Modeling Language for Global Optimization*. The MIT Press, Cambridge, Mass., 1997.

[Hentenryck, 1999] P. Van Hentenryck. *OPL: The Optimization Programming Language*. The MIT Press, Cambridge, Mass., 1999.

[Horvitz & Zilberstein, 2001] E. Horvitz, S. Zilberstein. Computational tradeoffs under bounded resources. *Artificial Intelligence*, 126(1-2), 2001.

[Jourdan, 1995a] Jean Jourdan. *Concurrence et Coopération de Modèles Multiples dans les Langages de Contraintes CLP et CC : Vers une Méthodologie de Programmation par Modélisation*. PhD thesis, Université Denis Diderot U.F.R. Informatique, 1995.

[Jourdan, 1995b] Jean Jourdan. Concurrent constraint multiple models in CLP and CC languages: toward a programming methodology by modelling. In *Proc. of INFORMS*, New Orleans, USA, October 1995.

[Kask & Dechter, 1996] K. Kask, R. Dechter. A graph-based method for improving gsat. In *Proc. of AAAI-96*, pages 350–355, Portland, OR, 1996.

[Kumar, 1992] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[Laburthe *et al.*, 1998] François Laburthe, Pierre Savéant, Simon de Givry, Jean Jourdan. Eclair: A library of constraints over finite domains. Technical Report technical report ATS 98-2, Thomson-CSF LCR, Orsay, France, 1998.

[Laburthe, 1998] François Laburthe. SaLSA: a language for search algorithms. In *Proc. of CP-98*, pages 310–324, Pisa, Italy, October 26-30 1998.

[Lobjois & Lemaître, 1998] L. Lobjois, M. Lemaître. Branch and Bound Algorithm Selection by Performance Prediction. In *Proc. of AAAI-98*, pages 353–358, Madison, WI, 1998.

[Marriott & Stuckey, 1998] Kim Marriott, Peter J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.

[Mazure *et al.*, 1996] B. Mazure, L. Saïs, E. Grégoire. Boosting complete techniques thanks to local search methods. *Symposium on Math&AI-96*, 1996.

[OpenEclair, 2000] Eclair solver, 2000. Open source version at http://www.lcr.thomson-csf.com/project/openeclair.

[Perron, 1999] Laurent Perron. Search procedures and parallelism in constraint programming. In *Proc. of CP-99*, pages 346–360, Alexandria, Virginia, October 11-14 1999.

[Pesant & Gendreau, 1999] G. Pesant, M. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 1999.

[Tsang, 1993] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[Van Hentenryck, 1989] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series, The MIT Press, 1989.

[Wallace, 1995] Mark Wallace. Constraint programming. Technical report, IC-Parc,, Imperial College, London, UK, September 1995.

[Wallace, 1996] R. Wallace. Enhancements of branch and bound methods for maximal constraint satisfaction problem. In *Proc. of AAAI-96*, pages 188–195, Portland, OR, 1996.