Large Combinatorial Optimization Problems: 
a Methodology for Hybrid Models and Solutions

Carmen GERVET
IC-Parc, Imperial College
William Penney Laboratory
London SW7 2AZ, U.K.
E-mail: c.gervet@icparc.ic.ac.uk

Abstract: Large Scale Combinatorial Optimization problems (LSCO) appear in numerous types of industrial applications (e.g. production scheduling, routing problems, financial applications). They are NP-complete problems characterized by large sets of data, constraints and variables, and often have an impure structure. Tackling such problems successfully requires both experience and skill. The current trend in the Constraint Programming (CP) community is to enhance the features of CP languages to ease the modelling and solving of LSCO problems by providing 1) natural modelling, 2) built-in constraints facilities (global constraints), and 3) integration of different constraint solvers (from mathematical programming, constraint programming, stochastic search methods). The common aspect is the growing awareness that we need to hybridize different models and methods and go beyond the constraint programming framework, essentially for efficiency and scaling reasons. This sets strong requirements upstream the programming phase to reduce the increasing level of expertise that is required to model the problem and map the model to adequate methods. This talk aims at filling a gap in this direction. We present the ongoing work within the CHIC2 Esprit project and at IC-Parc, in terms of providing a thinking process and a methodology for modelling LSCO problems with a hybrid perspective. We address the aspects of problem modelling, algorithm characterization, and mapping of the model to hybrid algorithms.

Keywords: Optimization Problem solving, hybrid models, methodology

1 From constraint programming to hybrid methods...

The Constraint Programming (CP) framework has emerged in the wide field of Artificial Intelligence to extend the application domain of logic based programming languages to deal with combinatorial search problems modelled as CSPs[Mac77]. This has been achieved by embedding and integrating the CSP
model and consistency techniques within the logic programming paradigm (cf. the CHIP system [DSea88][Hen89]). Today, the logic component of such languages is understood in the wide sense of allowing for declarative statements of nondeterministic programs, whereby the modelling of the problem is independent of its solving. Consistency techniques [Mon74][Mac77] are constraint propagation techniques that prune the search space associated to a CSP by removing values from variable domains that can never be part of any feasible solution.

Today, the logic component of such languages is understood in the wide sense of allowing for declarative statements of nondeterministic programs, whereby the modelling of the problem is independent of its solving. Consistency techniques [Mon74][Mac77] are constraint propagation techniques that prune the search space associated to a CSP by removing values from variable domains that can never be part of any feasible solution.

The success of CHIP in the late 80s has prompted the development of numerous CP languages that allowed for a natural and declarative statement of combinatorial search problems as CSPs, and whose main solver is based on constraint propagation techniques. First signs of hybridization were the introduction of search procedures in CP languages to deal with combinatorial optimization problems. One of them is the branch and bound search algorithm and its variants (see [GM84] for a detailed description).

However, when addressing LSCO problems, scaling and efficiency became crucial issues that limited the potentials of CP technology. It appeared clearly that constraint propagation coupled with variants of the branch and bound search was not the “magic” answer to large scale problems. Sup-parts of LSCO problems could be solved very efficiently and to optimality by special-purpose algorithms or global transformation methods, and some local search methods could perform better than branch and bound and reach fairly “good” solutions.

Thus, there was a practical necessity to hybridize constraint propagation methods with other constraint handling and search algorithms. First seen as a competitor who needed to be beaten [Cra94], the OR field has become the main source of inspiration for specialized algorithms and Mathematical Programming algorithms/packages together with stochastic search methods. Today, the requirements to enhance the CP framework is being fulfilled at the level of CP languages which provide new facilities like:

- High level languages (e.g. in ECLiPSe[IC-98], CLAIRE, CHIP)
- Debugging features (PROLOG IV[BT95], cf. DISCIPL project).
- Global constraints (e.g. in CHIP[BC94], ILOG SCHEDULE [ILO97]).
- Integration of hybrid solvers (e.g. in ECLiPSe, CLAIRE, ILOG PLANNER)

Also, the number of hybrid solutions that can be found in the literature is booming (see [CL95] [ESRW98][CL97][RWH98] to name a few).
However, these new language features set requirements upstream the programming phase to revise the way we approach LSCO problems in terms of formulation of an algebraic model, identification of constraint handling and search methods, and mapping of the methods to the model.

2 Why revising our views on problem modelling?

Some general guidelines to model and solve combinatorial problems using the CLP technology were first derived in the CHIC Esprit project [CFG95]. In the ongoing CHIC2 project, we wish to complete this work with the idea of implementing hybrid solutions to LSCO problems.

Let us recall the process of solving an LSCO from a pure CP perspective. The generic approach consists of the following incremental steps:

- Defining the problem with the client

- Extracting the relevant technical information in terms of input data, outputs (decision variables), constraints (hard/soft) and decision criteria

- Identifying the characteristics of the search space (number of variables and constraints, client decision strategies, ...)

- Modelling the problem using a CP language

- Following an incremental prototyping approach to improve the model and experiment with different search strategies.

One can notice that prototyping tends to replace the need for an algebraic or mathematical model because CP languages provide a lot of flexibility to represent the constraints in a natural setting, and they very often allow the developer to define new constraints (commonly called user-defined constraints). User-defined constraints come together with the transformation rules that set how to check and infer their local consistency. Such a flexibility results directly from constraint propagation algorithms that are totally independent of the constraints at hand. Indeed, the resolution phase in CP distinguishes two cooperating processes: the constraint handling algorithm and the search engine.

The constraint handling algorithm is a generic constraint propagation algorithm, also called fixed point algorithm, that calls in a data-driven way the...
transformation rules to infer the local consistency of each constraint in the program. This generic algorithm is part of the language and independent of the constraints and computation domain we are working with. The transformation rules must have some properties (contractance, idempotence, monotonicity) which guarantee that the propagation algorithm always terminates, has a unique fixed point independent of the ordering of the transformation rules, and is correct (e.g. see in [Ben95] [Ger97]). The search engine is usually left to the developer, unless a programming language provides built-in optimization predicates. Its definition is a complex task that aims at deriving optimization procedures and/or heuristics to explore the search space in order to look for optimal or “good” solutions.

The CP technology showed its ability to represent flexible models and solve small problems with complex constraints. As we mentioned earlier, it needs to be coupled with Mathematical Programming (MP) approaches and specialized OR algorithms to be efficient when tackling large scale problems. Indeed, MP algorithms can solve optimization problems with thousands of variables and constraints.

However, MP algorithms or MP software packages can not be applied to “any” model. A MP model can sometimes be shaped in terms of a matrix model [JL98b] but it usually has a standard form like:

\[
\begin{align*}
\text{(Objective)} & \quad \text{Min} \sum_i c_i \times x_i \\
\text{(Constraints)} & \quad \forall j, \sum_i a_{ij} \times x_i \leq b_j, a_{ij}, b_j \in \text{integer} \\
\text{(Decision variables)} & \quad x_i \in \mathbb{R}^+ \quad (\text{or } \mathbb{Z}^+ \text{ for integer problems})
\end{align*}
\]

The MP community is aware that such models are not always adequate, and are generally independent of the problem data. In such cases other OR models can be used like simulation models. The MP process to tackle LSCO problems focuses on the building of a well-defined mathematical model [Wil94] that can be mapped and input in a specified shape to existing algorithms and/or packages [Tah95]. The formulation of the mathematical model is of crucial importance to solving the problem (specially for integer programming models). To design the model, the LSCO problem is first classified (e.g. set partitioning, set covering, matching flow) and depending on the problem constraints and decision variables it can be modelled from a set of different well-defined models (e.g. linear models, integer models, dynamic models). Because MP technology was not initially aiming at any computerized solution, the models were necessary to reveal relationships between entities which were not apparent in a natural language definition of the problem, but also to analyze the mathematics of
the problem (e.g. the number of constraints is a good indicator of computational complexity, sparsity of the inherent matrix) and help suggest methods, to possibly experiment different models [NW88].

Thus the MP modelling and solving process is quite different from the CP one. The first framework focuses on the mathematics of the problem, its classification and the structure of the solution space (properties of the associated matrix) whereas the latter focuses on the structure of the search space, its characteristics and the flexibility of the programming language to build user-defined constraints, transformation rules and search engines. The use of MP and specialized OR techniques within CP languages requires the future developer to put more emphasis on the algebraic modelling of the problem before attempting to use a programming language. Among many reasons is the need to identify potential decompositions of the LSCO problem [JL98a], in order to map sub-parts of the problem with existing OR algorithms. One simple example can be found in [ESRW98] where the algebraic model showed that a sub-part of the LSCO describes an MP problem whose associated matrix is unimodular. This property ensures that MP will find the optimal integer solution to this sub-problem (much faster than CP would).

3 Towards a new thinking process to address LSCO problems

In this talk we introduce a new process to tackle LSCO problems with a hybrid perspective. We take into account the main differences that exist between a language driven CP modelling and a well-specified mathematical MP approach. This process takes into account the necessity to have a form of mathematical model before any attempt to implement the solution. We propose a modelling called algebraic modelling that is not limited to well formed mathematical formulas but allows for natural settings like all-diff($X_1,\ldots,X_n$).

The global process follows a two stages approach: problem definition and solution construction. However, within the construction stage we make a clear distinction between a design phase and a programming phase, even though they remain tightly connected.

3.1 The different stages

The design phase takes as input a formula-free document that describes in plain text the user requirements and presents a conceptual model of the problem in terms of entity/relationships model (inputs, outputs, relations, and
decision criteria). The design phase transforms this document by defining an algebraic model of the problem, extracting the characteristics of an algorithm by studying the classification of the problem (and sub-parts of it) and its specificity in terms of dimensions (number of data, variables, constraints) and the structure of the constraints. The local view (CP perspective) focuses on the different components and dimensions of the problem, while the global view tries to classify the problem or sub-parts of it (MP perspective). During this phase different types of problem decomposition will be analyzed: organizational, technical and hybrid decomposition (see [JL98a]). In case of a problem decomposition, there should be one algebraic model per sub-problem. Finally, and this is a crucial point, the design phase ensures that a mapping of the algebraic model(s) to the algorithm(s) exists. The output of the design phase is a written document called “problem solving document” that contains the algebraic model(s) and the design of the algorithm(s) we intend to implement.

The programming phase focuses on the coding of the model and algorithm in a programming language and identifies test cases to check the correctness and efficiency of the solution. Ideally the design phase should derive algebraic models in a “ready to be implemented” form. This can not be the case today where most CP languages offer large sets of predefined predicates that do not follow a mathematical like setting. These high-level facilities and programming abstraction are certainly ideal for the CP community and its application developers. However, we believe that the CP framework is ideal to integrate OR algorithms and software packages. Thus, if (and only if) we wish the OR community to make a powerful use of our programming facilities we might have to move towards modelling languages like VISUAL SOLVER or LOCALIZER [MH97]. The OR community is already making a move in this direction by offering modelling languages like AMPL [FGK93], XPRESS-MP [Das97], LP-TOOLKIT

Finally, it is important to note that the design and programming phases are tightly connected. For example during the programming phase it might become clearer that we need to tune the algorithm and/or adjust the algebraic model. At this point we need to go back to the design phase and revise our problem solving document. These iterative steps from the design phase to the programming phase and reciprocally is called prototyping.

3.2 The global process

The global structure of the transformation process from the problem definition to its computerized solution is illustrated in Figure 1. The prototyping phase is illustrated by the dotted rectangle that goes through the items 4-5-7-9.

---

Figure 1: LSCO problem solving process
Each of the three stages (problem definition, solution design and programming) is represented by a triangle. The triangular shape suggests that each item in a triangle is different from the others but not independent and all three form one entity or structure that leads to a well-defined output (document or program). Also, in each triangle one component plays the local role of validation that gives the green light to go ahead or stay within the triangle for further adjustments or even go back to a previous state of the process (cf. the iterative prototyping cycle).

An essential point that is visually suggested by this solving tree is that both the design and programming phases are reflections of the problem definition phase. If we match problem definition with design and problem definition with programming, we obtain two lattices. As a conclusive image, these two lattices lead us to view the process of design and programming as refinements of the problem definition. Ideally, when the problem is fully solved and validated by the client the boundaries of these lattices are pruned and unified. The solution is in fact a hidden component of the problem definition. Also, we can see that the rectangle illustrating the prototyping cycle represents the incremental process of transformation of our views on the problem and its solution. Both the design and programming phases are tools for us to extract the solution. Today each of the two phases brings its own contribution to the construction stage. But depending on the future evolution of CP languages and hybrid solutions the programming phase could be raised to the design level.

4 Acknowledgements

This work has been supported by the CEC within the ESPRIT project CHIC2 22165 (see http://www.icparc.ic.ac.uk/chic2/). We wish to thank the partners and colleagues of the project who have largely contributed to the ideas presented in this paper. Special thanks to Pascal Brisset who showed a particular interest in this work.

References


