Large Scale Combinatorial Optimization: A Methodological Viewpoint

Carmen GERVET

In this article, the author describes the results of a collaborative European project work with partners from Bouygues, Euro-Decision, ICL, IC-Parc, NTUA, and Renault.

Abstract. The industrial and commercial worlds are increasingly competitive, requiring companies to be more productive and more responsive to market changes (e.g. globalisation and privatisation). As a consequence, there is a strong need for solutions to large scale optimization problems, in domains such as production scheduling, transport, finance and network management. This means that more experts in constraint programming and optimization technology are required to develop adequate software. Given the computational complexity of Large Scale Combinatorial Optimization problems, a key question is *how to help/guide in the tackling of LSCO problems in industry.* Optimization technology is certainly reaching a level of maturity. Having emerged in the 50s within the Operational Research community, it has evolved and comprises new paradigms such as constraint programming and stochastic search techniques. There is a practical need, i.e. efficiency, scalability and tractability, to integrate techniques from the different paradigms. This adds complexity to the design of LSCO models and solutions.

Various forms of guidance are available in the literature in terms of 1) case studies that map powerful algorithms to problem instances, and 2) visualization and programming tools that ease the modelling and solving of LSCOs. However, there is little guidance to address the process of building applications for new LSCO problems (independently of any language). This article gives an overview of the *CHIC-2* methodology which aims at filling a gap in this direction. In particular, we describe some management issues specific to LSCOs such as risk management and team structures, and focus on the technical development guidance for scoping, designing and implementing LSCO applications. The design part in particular views the modelling of LSCOs from a multi-paradigm perspective.

1. Background and motivation

1.1. Introduction. The original work on methodology has been motivated by the growing use of the constraint programming technology to develop applications software for real world combinatorial problems like car sequencing, timetabling,

1991 *Mathematics Subject Classification.* Primary 90C27 65K10; Secondary 68T20.

This work has been supported by the CEC in the ESPRIT project 22165, Creating Hybrid solution in Industry and Commerce (*CHIC-2*). The author wishes to thank the anonymous referees for their helpful reviews, and Richard Wallace for his detailed and most valuable comments.

VLSI design. We focus on complex problems so-called Large Scale Combinatorial Optimization problems (LSCOs). Even though the maturity in optimization techniques has prompted the development of powerful modelling and programming languages, a high level of experience and expertise is still required to tackle LSCOs. The reason lies in the nature of LSCOs:

- Real world problems which do not fit into well-defined problem categories. There is no efficient solution models publicized in the research literature which would apply to them and take into account their unique features (e.g. multiple decision criteria, user-defined constraints)
- Large scale problems, characterized by large sets of data, variables and constraints. Existing algorithms do not always scale up, some decomposition aspects must be considered (e.g. problem structure, hybrid models, co-operation between solvers)
- Computationally difficult problems, i.e. *NP*-hard problems whose solving requires anyhow a lot of knowledge and experience.

From a technological point of view, three main paradigms have shown to contribute to the construction of powerful optimization algorithms: Operational Research (OR), Constraint Programming (CP), and stochastic search methods. Each paradigm provides its own forms of support to ease the formulation and tackling of combinatorial optimization problems. In many cases, the application can be solved efficiently thanks to the availability of increasingly powerful modelling and optimization tools and to the progress in research, particularly in results that exploit the hybridization of solvers. Hybridization is the process of integrating multiple solvers from different paradigms to co-operate and build a single algorithm. Intuitively, hybridization of solvers tends to solve large scale and complex problems efficiently by better exploiting LSCO structural features (i.e. using each solver when it is most appropriate). Some modelling/programming CP languages provide some support for hybrid modelling and solving (e.g. global constraints[1], integration of hybrid algorithms). However, upstream from the programming activity, a lot of design work is required in order to identify the structural features of an LSCO, investigate technical or efficiency needs for a problem decomposition, etc. There is little guidance in the literature on how to design hybrid models and algorithms efficiently. The proposed methodology addresses the design of LSCO models from a multi-paradigm perspective.

From a software development point of view, LSCO projects resemble any IT project and their lifecycle is quite similar. As F. P. Brooks says in [**Bro95**]:

> *The challenge and the mission are to find real solutions to real problems on actual schedules with available resources.*

However, due to the computational complexity of LSCO problems and the expertise required to tackle them, the success of an IT project with an LSCO component comprises specific issues. A main component of the work presented in this paper consisted of identifying these issues and providing methodological guidance (methods, processes, do's and don'ts) to tackle them in the best possible way. Little research has been pursued on this topic. One main reason is that the development of LSCO applications software is fairly recent (80s). So the research discipline on

[1]built-in relations which allow for concise statements and global solving of a collection of constraints. One way to achieve such a global reasoning is to use OR techniques in a CP setting.

software development methodology for LSCO problems is an area still in its formative stages. In [**CFGG95**], Chamard et al. summarize the lessons learnt during the *CHIC* project on building a Constraint Logic Programming (CLP) methodology.[2] This pioneering work focused on issues rather specific to CLP and provides practical guidelines for modelling and solving constraint optimization problems in CLP. In the *CHIC-2* project, our objectives were to pursue the work further in particular by extending its scope and agenda to include 1) latest research progress in optimization technology (i.e. the multi-paradigm perspective) and 2) to consider the software engineering aspects as well. We are not aware of any other previous work in this new discipline.

1.2. A practical approach. The methodology evolved over three years and was designed by a consortium of industrial and academic partners with expertise in: Operational Research (Euro-Decision), constraint programming (Bouygues, IC-Parc), stochastic search (NTUA, Renault), and project management (ICL, Renault). The industrial experience and technical expertise of the partners in the different optimization paradigms brought a rich and diverse knowledge to the consortium, which allowed us to approach the development and management of LSCO projects from complementary angles. In addition, we incrementally refined and evaluated the methodology document by tackling four industrial applications (construction scheduling, flow shop, generalized car sequencing, energy trading). The application work was organized so that each problem was undertaken by three different partners along the project life. The objective was to experiment with different methods whereby each partner would apply his expertise (in a given paradigm) and seek for the best results (e.g. optimality, efficiency) often based on co-operation of solvers. This distributed approach had a tremendous input to the methodology work, and its evaluation. It allowed us in particular to:

- Step back from the academic exercise of solving benchmark problems
- Identify and address management issues specific to LSCO applications
- Assess the complexity of scoping and specifying LSCO problems, and provide means to capture and validate user requirements
- Test the soundness of a model and improve the quality of solutions by developing different approaches (models and algorithms) in parallel.

All the difficulties we encountered and the experience we gained in this collaborative application work, have clearly driven the building of the *CHIC-2* methodology and later on its testing. In this document we give an overview of the methodology. More specifically, we address the issues which we have identified as specific to LSCOs, and present guidance to tackle them adequately. The methodology is publicly available on the following website:

`http://www.icparc.ic.ac.uk/chic2/chic2_methodology/`

1.3. Content. The paper is structured as follows. Section 2 will summarize the different optimization paradigms and their current forms of guidance to tackle LSCOs. Section 3 introduces the concept of software methodologies and the specific LSCO issues we will consider. Sections 4 to 6 present the core elements of the *CHIC-2* methodology that include specific management issues, and a detailed description of our approach to the development of LSCO solutions from a multi-paradigm perspective. Finally a conclusion is presented in section 7.

[2]Constraint Handling in Industry and Commerce, Esprit Project.

2. Optimization technologies

2.1. The Constraint Programming paradigm. The Constraint Programming (CP) paradigm emerged from the Artificial Intelligence field to extend logic-based programming languages to deal with *combinatorial search problems* modelled as Constraint Satisfaction Problems (CSPs) [**Mac77**]. Typical examples are scheduling, warehouse location, disjunctive scheduling and cutting stock problems. This was initially achieved by embedding and integrating the CSP model and consistency techniques [**Mon74, MF85**] into the logic programming paradigm [**Kow74, CKC83**] (cf. the CHIP system [**vH87, DSea88**]).[3] Today, the logic component of such languages is understood in the wide sense of allowing declarative statements of nondeterministic programs, whereby the modelling of the problem is meant to be independent of its solution method. To deal with combinatorial *optimization* problems, search procedures have been introduced in CP languages and hybridized with propagation techniques (e.g. the branch and bound algorithm and its variants [**GM84**]).

The success of CHIP in the late 80s prompted the development of new CP languages, but also raised the question of its limitations. When addressing LSCOs, *scaling* and *efficiency* became crucial issues that limited the potential 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. Components of LSCOs could be solved very efficiently and to optimality by special-purpose algorithms. Also, local search methods were often found to reach sub-optimal but good quality solutions more quickly. Thus, there was a practical necessity to hybridize constraint propagation methods with other constraint handling and search algorithms. Usually seen as a competitor, the OR field has become a source of inspiration for specialized algorithms. Today, the requirements to enhance the CP framework are being fulfilled at the level of CP languages which provide new facilities like:

- High level languages (e.g. ECL^{i}PS^e[**IC-00**], CLAIRE, CHIP)
- Debugging features (e.g. PROLOG IV[**BT95**])
- Global constraints (e.g. CHIP[**BC94**], ILOG SCHEDULE [**ILO97**])
- Search heuristics (e.g. CLAIRE)
- Support for solver hybridization (e.g. ECL^{i}PS^e).

Research progress on solver hybridization[4] has proved to improve the tackling of complex applications, in terms of efficiency, scalability and even tractability. Intuitively, it tends to solve large and complex problems efficiently by better exploiting LSCO structural features (i.e. using each solver when it is most appropriate).

The number of hybrid solutions that can be found in the literature is growing (see [**CL95, CL97, RW98, RWH99, ESW00**] to name but a few). Hybridization has allowed us to tackle new applications with improved quality of results. However, the extended set of modelling facilities and the availability of new solvers increase the expertise level required to tackle LSCO applications. We need to go through large user manuals, learn about related paradigms, and most importantly open our minds to new forms of models for LSCO problems. Some research work is being carried out to better understand the role of problem formulation with respect to

[3] Constraint Handling In Prolog.

[4] Hybridization is the process of integrating multiple solvers from different paradigms to build a single algorithm.

solution methods, however it still focuses on CSP models (e.g. binary CSP, discrete CSP, SAT) solved by constraint satisfaction techniques, and does not yet consider hybrid models and algorithms from other paradigms (e.g. [**BvB98**]).

2.2. The Operational Research paradigm. The Operational Research (OR) paradigm is more mature and already provides a lot of guidance for modelling combinatorial optimization problems and map them to powerful algorithms. Of particular interest to us is the framework of integer or mixed integer combinatorial optimization (for core knowledge [**GN72, NW88**]). OR techniques used to solve Mixed Integer Programming (MIP) problems make use of special purpose algorithms and search methods. Special purpose algorithms like network flow algorithms, can be of great importance in handling LSCOs since network flow problems frequently arise as subproblems. They can also be more efficient than generic algorithms.

Search algorithms such as branch and bound consist of iterating two steps: 1) solving a relaxation of an MIP problem using linear programming methods (e.g. the Simplex method [**Dan63**]), 2) splitting the problem into subproblems where a constraint is added. Very efficient implementations of the Simplex algorithm are available in the market (e.g CPLEX [**CPL94**]), and can handle thousands of variables and constraints. Branch and bound algorithms have been very successful in solving large instances of a variety of MIP problem classes such as set partitioning and set covering. However, the efficiency of such algorithms can vary according to problem instances since they do not exploit the problem structure. Research focuses on improving search algorithms and studying the structure of the solution set to derive new constraints that would approximate this set as close as possible. Such constraints are called cutting planes (e.g. [**Gom63**]) and polyhedral cuts. The latter ones are usually more powerful since they exploit the problem structure, but they are more difficult to compute (for core knowledge [**NW88**]).

Thus the modelling of a discrete combinatorial optimization problem is an essential component to its efficient solving. Some guidance is provided to design the right mathematical programming model by means of case studies (see surveys in [**Grö93, Wil94**]). Also there is a growing number of algebraic modelling languages to ease the formulation of LSCO models fed to LP/MIP solvers (e.g. GAMS [**BM82**], AMPL [**FGK93**], XPRESS-MP [**Das97**]). The weak point is the scarcity of work that exists to integrate mathematical programming models with other paradigms like constraint programming, even though they should be seen as complementary.

2.3. The stochastic search paradigm. A third paradigm found to be useful in solving LSCOs is based on stochastic search methods, including Simulated Annealing (SA), Tabu Search (TS) [**KGV83, Glo89, GL97**], and Genetic Algorithms (GA) [**Gol89**]. This paradigm differs from the previous two in the sense that the methods require few restrictions on the nature of the problem. Stochastic search methods are iterative improvement techniques which explore a space of complete solutions. The first two techniques work on a unique solution. Neighbours of the current solution are obtained by modifying one or a few of the corresponding assignments. At each step of the optimization process, the current solution is replaced by one of its neighbours. GAs work on a population of solutions. Tuning plays an important role, and the selection of appropriate parameters is very much case-study oriented (e.g. temperature and cooling function for SA, list of tabu moves and neighbourhood function for TS, selection, crossover and mutation operators for GA).

The main drawback of iterative improvement techniques is that they cannot guarantee that an optimal solution be found if it exists, although in some cases it can be shown that the optimization algorithm will asymptotically converge to a global optimum. On the other hand, they can provide good trade-off results (i.e. good solutions in little CPU time) for very large LSCOs where other methods would fail. A lot of research and experimental work is focusing on stochastic methods where the main technical issues are the choice of neighborhood functions, acceptable configurations and data structures to store each state. To ease the modelling and lighten the work of handling complex data structures, some recent work has focused on designing a modelling language for local search techniques (see [**MvH97**]).

2.4. Summary. The existing forms of guidance to tackle LSCOs remain mostly technical, and driven by the paradigm at hand. They consist of:

1. Case studies: Designing efficient algorithms for specific problem categories; e.g. heuristics techniques for CSPs, Polyhedral cuts for MIP, tuning parameters for stochastic techniques
2. Modelling guidelines: Mapping models to algorithms; e.g. adequate problem formulation in CSPs, MIP models and matrices property in MIP
3. Implementation support: Easing the tackling of LSCOs by embedding powerful algorithms into programming languages, and providing modelling facilities in CP languages (high level modeling, global constraints, support for hybridization), algebraic modeling languages, modeling languages for local search.

Given the level of maturity of the three paradigms we have presented, and considering the important role they play in LSCO applications, a logical step forward is to offer similar forms of *technical* guidance for a multi-paradigm technology. Support for hybrid modelling and solving is provided by some CP programming languages. However, upstream from the programming activity more guidance is needed as new issues arise at the design level: problem decomposition, algorithm characterization, integration of hybrid models, co-operation between solvers. Addressing the design of LSCOs from a multi-paradigm perspective is a core element of the *CHIC-2* methodology. Another key element is to provide forms of guidance for the development and management aspects of optimization projects.

3. An engineering perspective

As mentioned in the introduction, our goal is not only to consider the development of LSCO solutions from a multi-paradigm perspective, but also to consider the software engineering aspect of LSCO projects. In this respect, we have investigated several software engineering methodologies and have identified some features of LSCO problems which cannot be dealt with efficiently by such approaches. Our objective is to provide guidance for a dedicated approach that treats these issues.

First, let us recall general aspects of software engineering methodologies, their purpose and the different trends that are evident today.

3.1. Software engineering methodologies. A software engineering methodology provides a framework of engineering methods and processes to develop software. Most existing methodologies are scoped either for specific roles (project manager, application developer), or specific application domains and development technologies such as Knowledge-base Systems, Object Oriented Modelling. In our case, the scope is on how to use the optimization technology to tackle Large Scale Combinatorial Optimization applications. We are concerned both with the management and development of projects based on LSCO problems. Such a scope has not been considered yet but general software engineering methodologies already provide guidance.

We studied various software engineering methodologies focusing on different themes such as project management (LBMS [**LBM**], DSDM [**DSD**]) object oriented modelling (UML [**BRJ96**]), and knowledge-base systems (CommonKADS [**dHMW$^+$94**]). We undertook a survey to evaluate the applicability of such methodologies to our needs. The evaluation was essentially based on the diverse industrial experience of the consortium. Our conclusion was that many engineering features can be used directly to deal with projects with LSCO components. These include to some extent the project lifecycle (we will see which revisions are needed), most features of project management such as planning and resourcing, progress and change control, and the integration of LSCO components into full software systems (database connections, GUIs).

3.2. LSCO specific features. However, some aspects are strongly related to the technology at hand and the intrinsic computational complexity of LSCO problems, because both require expert knowledge and experience, and make such projects very risky. Our analytical survey revealed that the following aspects require a dedicated approach to execute LSCO projects successfully:

1. Project management
   - LSCO team structure (size, skills, working structure)
   - Development model (lifecycle)
   - Risk management
2. Development activities
   - Problem definition
   - Solution design
   - Programming

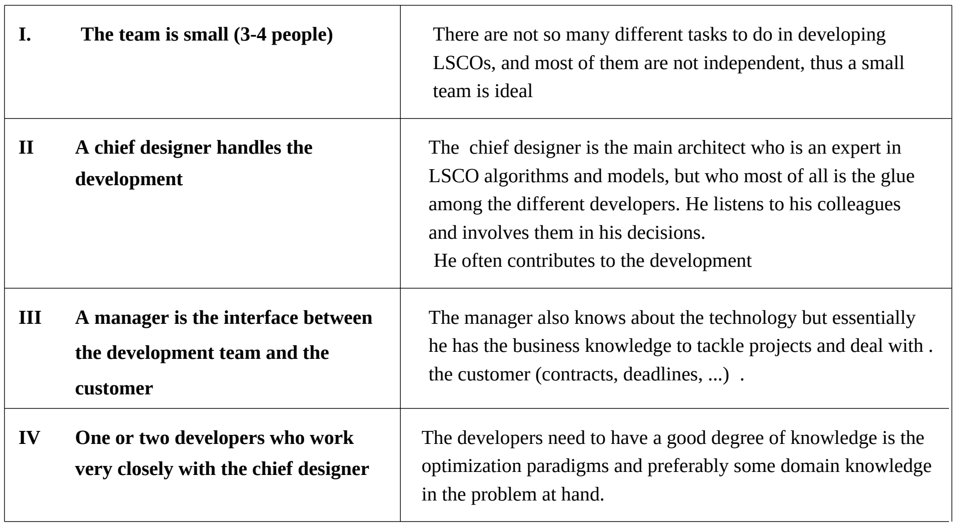
The *CHIC-2* methodology document and browser contain both the general software development features, and the specific management and development guidance. However, in this overview article we focus on the above issues. These are of interest both to researchers and application developers. The objective of the methodology is to help those who are not experts either in LSCO technology or in the development of LSCO applications. We hope to allow them to:

- *Reduce* the time required to tackle LSCOs, by providing generic processes for project management and development
- *Lower* the expertise level needed, by providing guidance to define, design and program LSCOs
- *Enhance* the quality of solutions, by measuring different criteria (correctness/time/cost) and considering real-world issues often left aside in an academic environment such as development time, generic and reusable code.

4. Project management

Project management is essentially about people and processes. Due to the computational complexity of LSCO projects, there are often two levels of project management: 1) one for the sole LSCO component, 2) and one for the global software development (including the engineering of the database and GUI work). The second one which comprises standard software engineering issues is not specific to our concerns and will not be addressed here. The management issues we address in this article are: LSCO team structure; development model and risk management.

4.1. LSCO team structure. The LSCO team is responsible for the development of the LSCO component. Its size depends obviously on the dimensions of the project. However, since the building of the LSCO component does not require many parallel tasks or heterogeneous skills, the LSCO team is preferably small. Experience has shown that there are some key structural factors to the construction of a good LSCO team. Figure 1 recalls the main ones [**Ger98**]. Others can be found in [**Bro95**], where Brooks discusses the building of teams for IT projects.



I.	**The team is small (3-4 people)**	There are not so many different tasks to do in developing LSCOs, and most of them are not independent, thus a small team is ideal
II	**A chief designer handles the development**	The chief designer is the main architect who is an expert in LSCO algorithms and models, but who most of all is the glue among the different developers. He listens to his colleagues and involves them in his decisions. He often contributes to the development
III	**A manager is the interface between the development team and the customer**	The manager also knows about the technology but essentially he has the business knowledge to tackle projects and deal with . the customer (contracts, deadlines, ...) .
IV	**One or two developers who work very closely with the chief designer**	The developers need to have a good degree of knowledge is the optimization paradigms and preferably some domain knowledge in the problem at hand.

FIGURE 1. LSCO Team Structure

Note that when there is not much need for parallel actions, the smaller the team, the less likely are the risks of misunderstandings and bad communication. For the same reasons, it is important to minimize the number of interfaces between the LSCO team and the customer team (including the end-users), since they would already tend to use different "working languages" making comprehension more difficult (application domain language versus technical language).

4.2. Development model. A development model or lifecycle structures the life of a project in terms of high level stages. Each stage is built towards the completion of well-defined objectives that measure the degree of advancement in terms of milestones (contractual agreements with the problem owner), and deliverables (e.g. written documents, prototypes, software). General IT project lifecycles can broadly follow two models: the waterfall or the spiral model [**Boe88**]. The waterfall model tends to show the evolution of a project from a contractual point of view by a sequence of stages, while the spiral model is more driven by iterative development processes usually used for complex and ill-defined decision problems. Each has its strengths and weaknesses [**PL90**]. The waterfall approach is a rather formal method broadly used among business developers while the the spiral approach is more popular among ADE specialists (Aerospace, Defense and Engineering). Whatever a company development model is, our experience shows that it is crucial to introduce iterative development processes to control risks related to ill-defined LSCO problems, technical complexity, etc. A spiral model does not require much revisions to tackle LSCO applications. However, if one uses the common waterfall approach it should be adapted as follows. The idea is to use a two-phase approach which can be structured around 7 stages as shown in Figure 2: an exploration phase (stages 1/2/3/4), and an integration phase (stages 5/6/7).

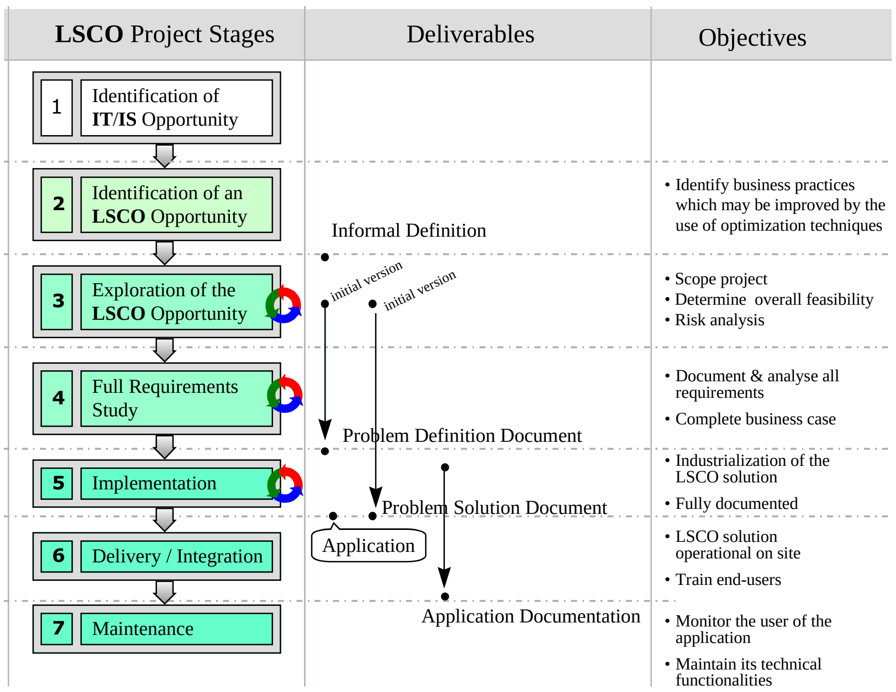

LSCO Project Stages
Deliverables
Objectives
1 Identification of IT/IS Opportunity
2 Identification of an LSCO Opportunity
3 Exploration of the LSCO Opportunity
4 Full Requirements Study
5 Implementation
6 Delivery / Integration
7 Maintenance
Informal Definition
initial version
initial version
Problem Definition Document
Problem Solution Document
Application
Application Documentation
• Identify business practices which may be improved by the use of optimization techniques
• Scope project
• Determine overall feasibility
• Risk analysis
• Document & analyse all requirements
• Complete business case
• Industrialization of the LSCO solution
• Fully documented
• LSCO solution operational on site
• Train end-users
• Monitor the user of the application
• Maintain its technical functionalities

FIGURE 2. LSCO Project Lifecycle

The exploration phase aims at ensuring technical feasibility by focusing on prospective and prototyping work (getting the objectives, the problem definition and the algorithms right). Once the technical feasibility of the LSCO problem is guaranteed the integration phase can be carried out. The circles illustrate iterative development processes, denoted risk driven processes, we have introduced to tackle complex LSCO applications using a waterfall lifecycle.

Note that depending on the type of LSCO project the time spent in each stage can differ tremendously. There are broadly two types of LSCO projects: i) computerization projects, and ii) prospective projects. A computerization project considers an existing well-defined problem. The role of the LSCO application is to automate an existing decision making process previously done by hand or using spreadsheets. Such projects can usually follow a standard waterfall approach. On the other hand, a prospective project is about ill-defined problems whereby the objectives are not clear and the technical feasibility unknown. Basically, the problem owner wants to improve his company working policies and requires the LSCO application to provide new solutions to a new problem. The role of the application is to help the problem owner identify the "right" problem, and work on solutions that would improve his existing best practices. Prospective projects can require a very large amount of time spent in the first four stages of the life cycle (mainly identification and exploration of LSCO opportunity, and full requirement study). In such cases, the risk driven processes are essential. In particular the exploration stage can consist of a large amount of incremental iterations over the problem definition, the design of the solution and the programming activities.

4.3. The DSDM/RAD development model. The development model we have presented is structured around a 7-stage project lifecycle that includes risk driven processes, and three generic technical activities carried out during such processes (to be presented in the next sections). Such a vertical structure might not be the one in place in some companies where a lot of application work is prospective. Their software development methodologies can rather be oriented towards Rapid Application Development (RAD) methods (also called Dynamic Systems Development Methods- DSDM [**DSD**]). An RAD model considers broadly a two-phase lifecycle where the first phase is the incremental and iterative development of a solution, and the second one is the integration of the LSCO component into a full software system.

The cornerstone of an RAD development process consists of:

- Timeboxes that set deadlines for the completion of a subset of the business objectives. The objectives selected are quantifiable such that progress towards goals can be measured. They can also be prioritized according to the customer's needs and the assessment of the technical difficulties
- Iterative prototyping and incremental development process. Solutions are built and evaluated incrementally as the objectives evolve. The problem definition evolves with the computerized solutions.

The question is: can the *CHIC-2* methodology be applied when the customer company has a different culture towards the project development lifecycle ? We believe it can for the following reasons.

First, the technical activities used in RAD methods are similar to the ones we present in section 6. The incremental process monitored by timebox objectives aims at iterating over the problem definition, design and programming activities. At the beginning of each timebox a new set of objectives is defined (new iteration through the problem definition activity), followed by a development process that iterates between the design and programming activities. The computerized solution is the output of the current timebox.

Second, the incremental definition and resolution of an LSCO problem can be viewed as taking place essentially during the "exploration of an LSCO opportunity" stage in the *CHIC-2* lifecycle (cf. Figure 1). This would mean that most of the incremental development phase takes place in stage 3. The integration phase of RAD methods follows that of the *CHIC-2* lifecycle.

Thus the *CHIC-2* development model we have presented defines a backbone for an LSCO project, while remaining flexible in the sense that the time spent in each stage depends on the LSCO application and the customer's culture. An RAD method would concentrate on stage 3 for the incremental development, as an alternative to the risk driven processes we consider.

5. Risk management

LSCO projects are risky in nature because they are computationally difficult to solve and they are not always well-defined (e.g. complex objective functions, hard and soft constraints, ill-defined data). Thus risk management plays a key role to minimize dangers of project delays and failure.[5] The objective is to reduce uncertainties by pro-actively analyzing the various risks and identifying countermeasures to control them. There are three main categories of risks handled in project management methodologies:

- Organizational (project planning, relations with the customer, political factors)
- Financial (project cost and resource plan)
- Technical (ill-defined problems, computational complexity, maturity of the technology).

In this article we focus on the assessment and control of LSCO related risks which are essentially technical. Technical risks depend on two main factors: 1) the type of problem we are dealing with (e.g. well-defined versus ill-defined), and 2) the complexity of the solution methods one intends to use. Depending on the degree of innovation of the project and the technology used, the level of risk will vary considerably. As an LSCO becomes increasingly complex (more innovative, less well-scoped) and the solution methods needed are less mature (e.g. new algorithms, hybrid methods), the level of risk grows.

5.1. Risk driven processes. Each of the seven stages of the waterfall model described is commonly characterized by its objectives, inputs, outputs, internal processes and development activities. For complex stages like 3/4/5 where the risk exposure can be high (e.g. ill-defined problems, technical decomposition required, efficiency requirements), we propose risk driven processes. We define a risk driven process as a means of *controlling* the technical risks associated with the complex stages of ill-defined LSCO problems. It comprises a set of technical activities (problem definition, design and programming), followed by a risk analysis and a decision making process. The analysis work evaluates the remaining risks associated with the project. To help in the risk analysis, we propose a set of risk scenarios against which risks can be assessed and decisions be made. A set of such scenarios is presented in the following section.

The decision process can be either to go forward (risks resolved or minimized), to refine the work done (reduce further the technical risks) or even to drop out of the project (too many unresolvable risks). One can drop out of an LSCO project

[5]A risk can be defined by 1) a degree of uncertainty regarding the occurrence of an event, 2) a negative effect if the event takes place [**HK96**].

for various reasons like: the client has changed his mind and is reorganizing his company, or the problem cannot be solved in the given time schedule. Risk driven processes are usually iterative because if the assessment of risks does not lead to conclusive results some more design, prototyping, etc, is required.

The *CHIC-2* risk driven process with respect to the exploration stage is illustrated in Figure 3.[6]

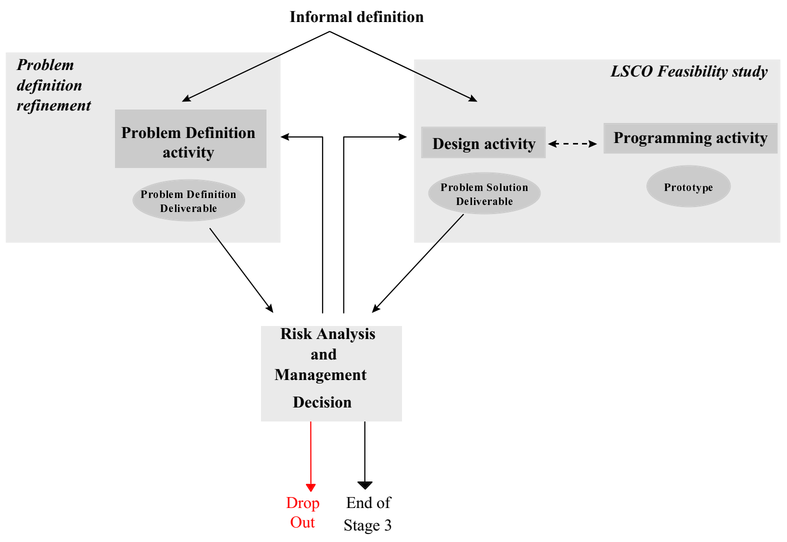

Informal definition
Problem definition refinement
Problem Definition activity
Problem Definition Deliverable
LSCO Feasibility study
Design activity
Programming activity
Problem Solution Deliverable
Prototype
Risk Analysis and Management
Decision
Drop Out
End of Stage 3

FIGURE 3. Risk Driven Process for the Exploration Stage

During the exploration stage the problem definition is refined and a technical feasibility study is performed. The feasibility study will attempt to reduce the technical risks. For example, this may include the development of one or several prototypes but can be limited to the design of a solution. Also, if the project corresponds to a prospective LSCO application, the problem definition activity can be very complex and iterated many times to clarify the objectives. The risk driven process will ensure that the risks are controlled and monitored.

5.2. Risk scenarios. While risk driven processes aim at monitoring the reduction of risks, the *CHIC-2* methodology also proposes some guidance to *assess/ foresee* some technical risks which are most likely to occur during the complex stages. For example, situations like the ones sketched below can increase risks of project delays or failure:

- Reluctance to spend enough time and effort on the feasibility study during the exploration and full requirement study stages (cf. Figure 2), will increase the risks of failure.

[6]Risk driven processes for stages 4 and 5 can be found on the website.

- Bad communication between the LSCO team and the problem owner: the problem owner does not understand the technical complexity of his application and is not willing to cooperate with the LSCO team (e.g. importance of providing real data sets). Or, the LSCO team does not understand the problem from the problem owner perspective, and tends to approximate its formulation such that it fits some well-defined models. Such problems can make difficult the success of exploration and full requirement study stages.
- Inappropriate level of expertise of the LSCO team: restricted ability to produce a "good" solution within the required timescales. This category of risks is most relevant to exploration and implementation stages of the project lifecycle.

A very effective approach to manage technical risks consists of identifying potential risk scenarios, assessing their probability of occurrences at a given stage, their impact on the project, and deriving countermeasures. We propose a set of risk scenarios specific to the development of LSCO projects. For each scenario we have identified their potential impact (in case of occurrence at a given stage) and derived guidelines in terms of countermeasures to prevent and possibly avoid them. An illustrative table of risk scenarios and guidance is given in Appendix A.[7]

In the next section we proceed with the description of the different development activities mentioned earlier. While the project management guidelines are concerned with the people and processes (lifecycle, risk management, teams), the development guidelines presented in the next section deal with the actual definition and construction of the LSCO solution. They are primarily meant for the LSCO designer and developers.

6. Art and craft

This section presents each development activity with its LSCO specificity. There are three generic activities in software development:

- Problem Definition
- Solution Design
- Programming

We are not concerned here about "when" to undertake each activity but rather about "how". The "when" is monitored by the development model and risk driven processes (e.g. see Figure 3 illustrating stage four).

Even though the activities are common to most IT projects, each of them has an LSCO specificity. The core questions are:

1. What are the specific LSCO issues to be addressed ?
2. How shall we tackle them with a multi-paradigm perspective ?

The *CHIC-2* user guide wishes to answer these questions by presenting LSCO issues relative to each activity and proposing guidance to tackle them successfully. The guidance acts at two levels: a local process for each activity composed of different tasks ("what" to do), and a set of best practice guidelines to fulfill the tasks successfully ("how" to do it). Each process is visually represented by a triangle connecting three tasks. A key point is that we want to ensure that each activity outcome is validated locally. Thus in each tripartite process one task plays the local

[7]A complete list of the scenarios we have identified is available from the website.

role of validation, allowing the LSCO team to evaluate the technical work within each activity.

We present the iterative process of each activity and illustrate some of the *CHIC-2* guidelines on one specific task per activity. The remaining ones can be found in the complete guide.

6.1. Problem definition activity. Problem definition is about scoping the problem, capturing all the user requirements, and validating the requirements with the problem owner. It can take place in all the stages of the exploration phase (1/2/3/4). Natural language can be very ambiguous, and problem owners often use specialized terms to describe their problem, so it is crucial to remove potential ambiguities during this activity. In order to minimize risks of misunderstanding, and guarantee (as much as possible) a complete and correct problem definition, we propose an iterative process in three steps, illustrated in Figure 4, and a set of guidelines that consist of:

1. Capturing the user requirements. Guidance: follow a checklist of key questions for capturing "all" the relevant requirements
2. Building a conceptual model. Guidance: write a structured form of the user requirements from a technical perspective
3. Validation. Guidance: Evaluate the conceptual model against the user requirements ensuring the mathematical understanding of the user requirements.

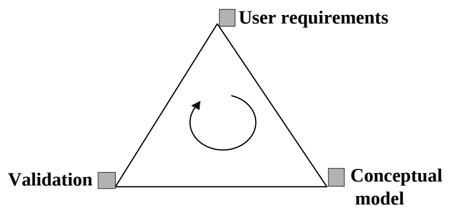


FIGURE 4. LSCO Problem Definition Activity

The goal of building a conceptual model in the problem definition activity is to extract the LSCO specificity from the user requirements before attempting to design a model. It is important to identify from a technical point of view the different components of the user requirements, their relationships and the dimensions of the problem. The core idea is to check that all relevant requirements are captured, and will not be simplified or approximated when viewed from a technical perspective.

CHIC2 guidance for the conceptual model. A conceptual model is a formulation of a problem by means of concepts. For an LSCO problem there are four basic concepts that constitute the kernel of any problem formulation independently of the model to be used: Inputs, Outputs, Constraints and Decision Criteria. They are illustrated in Figure 5, together with their relationships and dependencies.

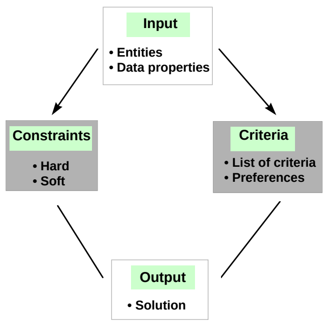


Figure 5. LSCO Conceptual Model Structure

Each of the four concepts is common to any optimization paradigm: Inputs consist of the different entities and data required for the problem (resources, contract profiles, warehouses,..) with their specificities and arities. Outputs are the solution elements the problem owner is looking for. At this stage they are independent of the decision variables that will be used to model the problem. Constraints are all the company policies that state the problem whether they must be fully satisfied (hard) or must preferably be satisfied (soft). Finally, the decision criteria are all the functions that contribute to defining the ideal solution. At this stage they need not be combined into a single objective function.

By extracting these basic technical components during the problem definition activity, we wish the LSCO team to make sure that:

- They fully understand the customer's problem from an LSCO perspective. This will avoid misunderstandings from occurring during the design and programming activities.
- They have identified most potential ambiguities in the user requirements (e.g. fuzzy statements like "in most cases", "maybe"). In such situations, the LSCO team can come back to the user and ask for refinements of terms before going ahead with the design and programming activities.
- They are not influenced or biased by any form of design model. The conceptual model should remain independent of any modelling framework and solution methods. One of its roles is to prevent the LSCO team from approximating the user requirements and simplifying them such that they "fit" the modelling language at hand.

Another advantage of the conceptual model, is that it can help the customer to better understand the way the LSCO team views and addresses their LSCO problem.

Documentation. The information captured during the problem definition activity is written in a document to be validated by both the LSCO and customer team. A layout of the problem definition document we propose, which will be maintained and updated all throughout the project life is depicted in Appendix B.

6.2. Design activity. The design activity concerns the extraction of problem structural features and the building of one or multiple model(s) of the problem (subproblems). The features will help characterize the algorithm to build. As opposed to the conceptual model, the design model is dependent on the technology. Just as the conceptual model is an intermediate step between the user requirements and the modelling, the design work acts as a preliminary step to the programming activity. It clearly depends on the LSCO team's knowledge about the different optimization paradigms, and their experience in problem modelling and matching algorithm characteristics.

Furthermore, in the context of a multi-paradigm framework, the design activity is particularly necessary since we do not assume any specific modelling framework nor optimization technology. New issues arise and need to be considered like technical decomposition of the problem, choice of a formulation, characterization of the (hybrid) algorithm to build. Indeed, if we would consider mainly the constraint programming paradigm, the focus would be on the design of a CSP model. The issues would deal with the selection of the decision variables, the constraint formulation, the choice of powerful heuristics and search methods. Similarly, if we would consider mainly the MIP framework, the focus would be on the design of the matrix model, i.e. finding the right formulation of the constraints that would lead to a matrix that best exploits the integrality constraints, etc. But, we are opening our technological environment to a multi-paradigm framework. Thus, there are no restrictions relative to the choice of model and algorithms apart from the ones set by the problem itself. This is very important for tackling LSCOs applications, since it allows us to focus on formulating the real world problem, instead of simplifying it so that it suits our modelling language/framework/experience.

Thus one of the main roles of the methodology for this activity, is to provide guidance for the LSCO team to study the problem structure and properties from different perspectives. This will allow the LSCO team to select the paradigm or hybridization of paradigms which will make the best use of problem features (in terms of pruning and search power). For this purpose, we propose the following iterative process for the design activity that focuses on three main tasks:

- Finding the algorithmic flavors
- Designing model(s) of the LSCO problem
- Ensuring that the the model and algorithm form a good couple

The last item illustrates the strong dependency which exists between a design model and the algorithm. A full evaluation of the mapping might only be possible after doing some programming. This would then require some tuning and adjusting of both the model and the algorithm (this is the local validation task). The process is presented Figure 6.

Once the tuning/adjusting is completed (for the current state), a Problem Solution Document is written that describes the design model of the problem and a characterization of the algorithm. One could argue that we do not need a design activity, and that we could proceed directly from the problem definition to the programming activity. This might hold for paradigms which are supported by very expressive modelling languages (e.g. CHIP), and this will certainly be true the day our languages are purely declarative.[8] However, from a software engineering

[8]A declarative language, ideally allows one to declare or state "what" the problem is, without specifying "how" it should be solved.

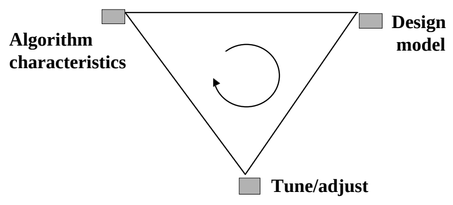


FIGURE 6. LSCO Design Activity

point of view, it is essential to have a permanent access to the latest description of the model(s) and algorithms implemented. Thus besides being a pre-programming activity, LSCO design acts as a communication tool for software maintenance.

CHIC-2 guidelines for algorithm characterization. The objective of this task is to extract from the problem definition the LSCO features with an open view to hybrid algorithms. The main concern is not to focus on one problem feature but rather to consider different perspectives. Whether one is in research or industry, we usually seek answers to some technical questions at this stage, like the following ones posted to both the CP and the OR newsgroup with respect to a specific LSCO:

```
Does my problem fall into a well-defined category ?
Are there any tools suitable for this class of problems ?
What types of heuristics apply to this class of problems ?
Are there any pointers to research on this class of problems ?
```

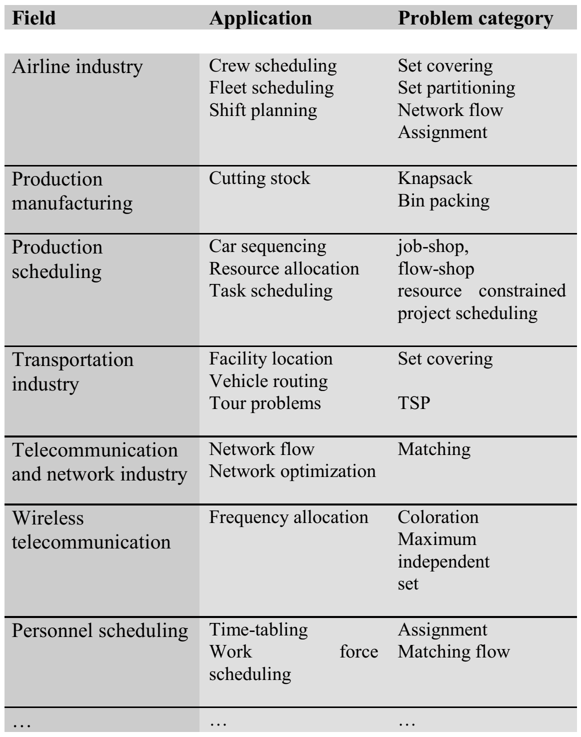
Clearly, an algorithm cannot be defined if the problem is not well understood. We propose a list of more specific questions for the LSCO team to analyze the user requirements from a mathematical and logical perspective. The questions are structured into three main groups as follows:

1. Problem structure; Can you classify the application domain of the problem: scheduling, resource allocation, transportation, time-tabling, etc ? What are the semantics and mathematical properties of your constraints ?
2. Quantitative aspects; Problem size. How big is a data set ? How many decision variables do you foresee, what would be their domain size ? How many hard constraints ?
3. Qualitative aspects; Solution type. Are you looking for an optimal solution or not ? How many criteria are you considering ? Is the objective function well-defined ? What is the level of user interaction required in the decision making process ? Are there requirements for solution quality (robustness, hard constraints on the execution time, ...) ?

The guidelines we propose to characterize the algorithm consist of three complementary views to extract the LSCO features from the problem definition:

• **Constraints and variables properties.** We propose checklists to identify the nature of the LSCO problem (unconstrained, constrained), the type of constraints and objective function (linear, quadratic, symbolic, and others), the type(s) of variables (boolean, continuous, discrete, mixed or non-numerical), the nature of the available data (deterministic, probabilistic). This approach is intended to help the LSCO team find out which methods to try out first before attempting more complex ones (e.g. based on hybridization and problem decomposition).

• **Problem decompositions (problem categories, constraint semantics).** If the LSCO problem appears to be intractable when solved as a whole, problem decomposition is necessary. The problem decomposition guidelines aim at helping the LSCO team identify *the most adequate way to decompose their LSCO problem.* They focus on two complementary aspects: 1) identify well-defined sub-problems (among existing problem categories) of the LSCO that can be efficiently handled by a single method; 2) structure the constraint set in different ways according to the constraints' semantics or their mathematical structure. The first aspect corresponds rather to an OR perspective. The guideline table, depicted Figure 7, presents different sub-problems of a set of application classes.



Field	**Application**	**Problem category**
Airline industry	Crew scheduling	Set covering
	Fleet scheduling	Set partitioning
	Shift planning	Network flow
		Assignment
Production manufacturing	Cutting stock	Knapsack
		Bin packing
Production scheduling	Car sequencing	job-shop,
	Resource allocation	flow-shop
	Task scheduling	resource constrained project scheduling
Transportation industry	Facility location	Set covering
	Vehicle routing	
	Tour problems	TSP
Telecommunication and network industry	Network flow	Matching
	Network optimization	
Wireless telecommunication	Frequency allocation	Coloration
		Maximum independent set
Personnel scheduling	Time-tabling	Assignment
	Work force scheduling	Matching flow
…	…	…

FIGURE 7. LSCO Sub-problem Categories

For such categories there is an efficient solver available in the literature or in optimization software packages (it might be dependent on the problem size). The main downfall of focusing on problem categories is that the actual LSCO problem may not be adequately tackled; some so-called side constraints may not fit in the problem category selected. The second approach corresponds rather to a Constraint Programming viewpoint and deals with such constraints.

Given the whole spectrum of LSCO problems, there exist classes of constraints which can constitute the basis of a problem decomposition: resource constraints, temporal constraints, spatial constraints. Methods to handle each class of constraints are suggested in the user guide and browser, together with some examples.

• **"Do's and Don'ts" for algorithm selection.** This guideline emerged from the academic and industrial experience of the *CHIC-2* consortium in tackling LSCO problems. A table of do's and don'ts is depicted Figure 8.

Application	**Do's**	**Don'ts**
Set covering Set partitioning including possibly additional knapsack constraints	Mixed Integer Programming Column generation for very combinatorial problems, allows an efficient technical decomposition	
Set selection	LP, CP with partial search (LDS) local optimisation	Pure CP
Scheduling Disjunctive scheduling (single tasks)	B&B, constraint propagation using literature shaving tabu heuristics	LP, genetic algorithms, simulated annealing
Production scheduling	small size : CP + heuristics. big size (multi-product, multi-machines), linear processes : MIP can bring satisfactory solutions, combined sometimes with heuristics for post-processing. May need to decompose the problem	Use of "pure" approaches
Time-tabling Fixed set of activities	Global constraints, flow algorithms. LP model, local optimisation	Pure CP (local propagation for hard problems)
Routing problems Travelling Salesman Problem	CP for small complex problems only Local optimisation by default, Branch and Cut is an option	LP without expertise
Vehicle routing problems	Local optimisation	CP only
Tour problems	Column generation MIP approaches (proved to solve such problems optimally)	If max duration constraints, don't use network flow approach with LP
Flow problems	LP models, flows, global analysis	Pure CP, pure LP genetic algorithms
…	…	…

FIGURE 8. Do's and Don'ts Table

The idea consists of presenting some choices of algorithms for a set of well known LSCO applications. The choices describe what the consortium considers to this date to be most efficient and successful. The guideline can be used as *a form of advice* when choosing initial methods to experiment with.

6.3. Programming activity. The programming activity corresponds to the actual encoding and evaluation of the design work. It can take place during any risk driven process of the exploration phase, and the final implementation stage.

Independently of any programming or modelling language, we propose some generic guidance relative to the following tasks:

1. Coding LSCO models and solution methods
2. Testing computerized solutions
3. Debugging programs

Those three tasks are core to any IT programming activity but do have an LSCO specificity in terms of "how" to complete the tasks. The iterative process built upon those activities is shown in Figure 9. The testing task plays the role of local validation and might require the design model to be refined and the algorithm to be tuned (if the results produced are not satisfactory). This goes with updating the design work.

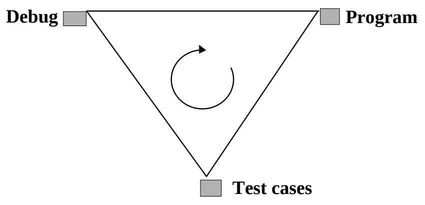


FIGURE 9. LSCO Programming Activity

CHIC-2 guidelines for program encoding: Overview. The guidelines we propose for coding the model and algorithm concern prototyping issues (e.g algorithm tuning, model adjusting, obtaining initial solutions, iterative process, reuse of code), handling of data flows between models, quality of code (generality, simplicity), and most specifically encoding hybrid algorithms. Currently, the guidance for hybrid encoding is illustrated using ECL^{i}PS^e programs, since the language is specially designed to allow for complex hybridization of solvers [**IC-00**]. We address various aspects such as which solver controls the search, and which degree of integration shall be considered:

- Loose; each solver works on its own and sends data to the other one
- Tight; a solver is called from another one when some constraint state or conditions are reached.

An example of tight integration of hybrid solvers is thoroughly described in [**ESW00**].

CHIC-2 guidelines for program testing. The main concerns when testing LSCO programs are correctness, robustness and efficiency of the program. However, from an application point of view, we also need to evaluate the program significance with respect to the client requirements. We propose a set of testing guidelines, which essentially consist of:

- Two forms of test (well-defined versus prospective projects). For well-defined projects, use real input data and compare the produced results with the customer's outputs. For prospective projects, provide solutions to the customer to be evaluated on the user site.
- Two approaches for one LSCO. In practice it has been shown extremely useful to consider two different implementations for an LSCO. The two approaches can differ in their models and in the techniques used to build the algorithms. We strongly encourage the use of techniques from different paradigms. This allows the LSCO team to test the correctness of the design model with respect to the problem definition, and possibly identify errors that lie in the formulation of the problem (which have not been identified in the design activity). Even though this can be time consuming (in the short term), in the end it ensures that the problem definition and the design models are coherent with each other; and at the same time that good quality solutions are produced. Two members of the LSCO team can carry parallel implementations.
- An evaluation of solution quality. The degree of optimality may not be the main concern in some LSCO applications. The customer is also interested in the CPU time, and the memory usage. So it is important to measure the average performance considering all criteria and check the customer's preferences.

Examples of multiple approaches for one LSCO such as the inventory management and the energy trading applications can be found respectively in [**RKCP97, CK98**] and [**Ger97, GCM99**], as well as in the user guide.

CHIC-2 guidelines for program debugging: Overview. If the testing procedure fails, the program needs to be debugged requiring some revisions of the model and algorithm. Of course, many iterations between test and debug can occur and one should be ready to change the code significantly. We provide a set of "tools and tricks" to debug LSCO programs independently of the programming/modelling language at hand. Examples of tricks are:

- The importance of using a graphical visualization of the searching process and the outputs. This can facilitate the reading of the results, help in understanding what went wrong, and also help discussing with the customer and identify core errors. Many CP languages such as OZ Explorer, provide means to visualize the search space.
- To use scaled down examples, in order to analyze by hand the results in the middle of the search
- To instrument the program to trace the key points.

More detailed guidance on each aspect can be found on the browser.

7. Conclusion and future work

In this article we have presented an overview of the *CHIC-2* methodology. The work focused on identifying and tackling the specific aspects of LSCO projects from a multi-paradigm and engineering approach. We have proposed dedicated means to deal with the computational difficulty, complex technology and risky nature of LSCO projects. These include in particular:

1. Risk driven processes as part of the development model to develop LSCO solutions in a real world environment.
2. A tripartite structure for each development activity which includes a local validation of the outcomes (documents, programs)
3. A conceptual model as part of the problem definition activity
4. A multi-paradigm approach in the design and programming activities

Any work on methodology is an ongoing work. However, we believe this constitutes a first step towards a generic approach for tackling LSCO applications that reduces the risks encountered in such projects, and the level of expertise required. The methodology is used by all the partners of the *CHIC-2* consortium. The user guide is publicly available at http://www.icparc.ic.ac.uk/chic2/.

References

[BC94] N. Beldiceanu and E. Contejean, *Introducing Global Constraints in CHIP*, Mathematical Computation Modelling (Elsevier Science, ed.), vol. 20(12), Pergamon, 1994, pp. 97–123.

[BM82] J.J. Bisschop and A. Meeraus, *On the Development of a General Algebraic Modeling System in a Strategic Planning Environment*, Mathematical Programming Study, vol. 20, 1982, pp. 1–29.

[Boe88] B. W. Boehm, *A Spiral Model of Software Development and Enhancement*, Computer, 1988.

[BRJ96] G. Booch, J. Rumbauch, and I. Jacobson, *The Unified Modeling Language for Object Oriented Development (v0.8 with v0.9 addendum)*, Rational Software Corporation, 1996.

[Bro95] F. P. Brooks, *The Mythical Man-Month*, Addison Wesley, 1995, New edition.

[BT95] F. Benhamou and Touraivane, *Prolog IV: langage et algorithmes*, Journées Francophones de la Programmation Logique, JFPL'95, 1995, in French, pp. 50–64.

[BvB98] F. Bacchus and P. van Beek, *On the conversion between non-binary and binary constraint satisfaction problems*, 15th National Conference on Artificial Intelligence, 1998.

[CFGG95] A. Chamard, A. Fischler, B. Guinaudeau, and A. Guillaud, *CHIC Lessons on CLP Methodology*, Tech. report, Dassault Aviation and Bull, 1995.

[CK98] Y. Caseau and T. Kokeny, *An Inventory Management Problem*, Constraints journal, vol. 3 (4), 1998.

[CKC83] A. Colmerauer, H. Kanoui, and M. Van Caneghem, *Prolog, bases théoriques et développements actuels*, T.S.I. (Techniques et Sciences Informatiques) **2** (1983), no. 4, 271–311.

[CL95] Y. Caseau and F. Laburthe, *Improving Branch and Bound for Jobshop Scheduling with Constraint Propagation*, CCS'95 (Springer Verlag, ed.), 1995.

[CL97] Y. Caseau and F. Laburthe, *Solving Various Weighted Matching Problems with Constraints*, CP'97 (1997), 17–31.

[CPL94] CPLEX Optimization Inc., *Using the CPLEX Callable Library, version 3.0*, 1994.

[Dan63] G. B. Dantzig, *Linear Programming and Extensions*, Princeton university Press, 1963.

[Das97] Dash Associates, *Xpress, user guide*, 1984-1997.

[dHMW+94] R. de Hoog, R. Martil, B. Wielinga, R. Taylor, C. Bright, and W. Van de Velde, *The common KADS model set*, Tech. report, ESPRIT project P5248, 1994.

[DSD] *DSDM homepage: http://www.dbs-group.co.uk.*

[DSea88] M. Dincbas, H. Simonis, and P. Van Hentenryck et al, *The Constraint Logic Programming Language CHIP*, FGCS (Japan), Aug. 1988.

[ESW00] H. El-Sakkout and M. Wallace, *A Probing Strategy for Minimal Temporal Perturbation in Dynamic Scheduling*, Constraints Journal, vol. 5 (4), 2000, To appear.

[FGK93] R. Fourer, D. Gay, and B.W. Kernighan, *A modeling language for mathematical programming*, The Scientific Press, San Francisco, 1993.

[GCM99] C. Gervet, Y. Caseau, and D. Montaut, *On Refining Ill-defined Constraint Problems: A Case Study in Iterative Prototyping*, PACLP'99 proceedings (1999), 255–275.

[Ger97] C. Gervet, *Ongoing research in CHIC-2: Decision making under uncertainty*, CP'97, ERCIM and Compulog workshop on constraints, 1997.

[Ger98] C. Gervet, *IC-PARC methodology*, 1998.

[GL97] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic, 1997.

[Glo89] F. Glover, *Tabu Search*, Orsa Journal of Computing, no. 1, 1989, pp. 190–206.

[GM84] M. Gondran and M. Minoux, *Graphs and algorithms*, Series in Discrete Mathematics, Wiley-interscience, Great Britain, 1984.

[GN72] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*, Wiley-interscience, 1972.

[Gol89] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[Gom63] R.E. Gomory, *An Algorithm for Integer Solutions to Linear Programs*, Recent Advances in Mathematical Programming, 1963, pp. 269–302.

[Grö93] M. Grötschel, *Combinatorial Optimization: A Survey*, Technical Report 93-29, DIMACS, Princeton University, New Jersey 08544, May 1993.

[HK96] F. J. Heemstra and R. J. Kusters, *Dealing with risk: a practical approach*, Journal of Information Technology, vol. 11-4, Chapman and Hall, 1996, pp. 333–346.

[IC-00] IC-PARC, Imperial College, *ECLIPSE version 5.0, user manual*, 2000.

[ILO97] ILOG Inc., *Ilog solver, user manual*, 1997.

[KGV83] S. Kirkpatrick, C. Gelatt, and M. Vecchi, *Optimization by Simulated Annealing*, Science, no. 220, 1983, pp. 671–680.

[Kow74] R.A. Kowalski, *Predicate Logic as a Programming Language*, IFIP (1974), 569–574.

[LBM] *LBMS homepage: http://www.platinum.com/products/appdev/ppcpr.htm.*

[Mac77] A. K. Mackworth, *Consistency in networks of relations*, Artificial Intelligence (1977).

[MF85] A. K. Mackworth and E. C. Freuder, *The complexity of some polynomial network consistency algorithms for constraint satisfaction problems*, Artificial Intelligence **25** (1985).

[Mon74] U. Montanari, *Networks of Constraints: Fundamental Properties and Applications to Picture Processing*, Information Science, 1974, pp. 95–132.

[MvH97] L. Michel and P. van Hentenryck, *LOCALIZER a modeling language for local search*, CP'97 (1997).

[NW88] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, Wiley, 1988.

[PL90] DeGrace P. and Stahl L.H., *Wicked Problems, Righteous Solutions: A Catalogue of Modern Software Engineering Paradigms*, Prentice-Hall, 1990.

[RKCP97] R. Rodosek, T. Kokeny, Y. Caseau, and C. Le Pape, *Creating Hybrid Solutions for Inventory Management Problems*, ERCIM/Compulog Workshop on Constraints in conjonction with CP'97, 1997.

[RW98] R. Rodosek and M. G. Wallace, *A Generic Model and Hybrid Algorithm for Hoist Scheduling Problems*, CP'98, 1998.

[RWH99] R. Rodosek, M. Wallace, and M. Hajian, *A New Approach to Integrating Mixed Integer Programming with Constraint Logic Programming*, Annals of Operations Research, 1999.

[vH87] P. van Hentenryck, *Consistency techniques in logic programming*, Ph.D. thesis, University of Namur, Belgium, July 1987.

[Wil94] H.P. Williams, *Model Building in Mathematical Programming*, Wiley, 1994.

Appendix A. Risk scenarios in LSCO project development

The following table presents a sample of the risk scenarios, consequences and countermeasures we have identified. It is mainly addressed to the LSCO team but can bring useful information to the customer as well.

Risk Scenario	Consequences	Countermeasure
1. Customer organisation have no previous experience of LSCO projects or other IT projects of similar size and complexity.	The customer abandons the project during or after Stage 3 of the project lifecycle.	At Stage 2, the LSCO expert must ensure that the sponsor fully understands the future stages of the LSCO project lifecycle, the working methods to be employed and the commitment required. Illustrate this by examples of similar projects which have been successfully completed. Identify an "expert" within the customer organisation for the project.
2. The first prototype built highlights major problems with the problem solution and the customer believes that the project is not progressing satisfactorily.	The customer abandons the project during Stage 3 or 4 of the lifecycle.	In risk analyses carried out in Stages 3 & 4 of the lifecycle, ensure that failure of the initial prototypes is considered and contingency plans are made.
3. Users continuously change their needs - functionality is changed or added.	A satisfactory prototype or solution cannot be quickly achieved.	User requirements must be specifically documented and agreed to before commencing any design or prototyping activities. Functionalities which do not change the problem nature can be ignored at Stage 3.
4. The data is not available or not usable for the optimisation tool.	No test can be performed with real data and the project is cancelled.	Validate as early as during Stage 2 that real data will be available and that the customer is able to provide some initial data sets during Stage 3.
5. The customer expects / was promised a high optimisation percentage (say 20%) but the actual result is much worse (say 1 - 2%)	User is disappointed - no further business. The application is not used.	Do not raise the customer / customer's expectations too high during Stage 3 of the lifecycle - be realistic about anticipated gains. Gains should be measured against what is done in practice, taking into account the dynamics of the problem, not against a theoretical, static estimation.
6. The LSCO team is specialised in one optimisation technique, tool or type of problem.	The solution design may not be the most efficient one.	Ensure that an LSCO team with a wide experience of different types of problem and optimisation techniques is used. Sub-contract part of the project to an external supplier.
7. The proposed design is a fully automated optimisation system (black-box).	The solution may not be robust when faced with changes in the user requirements, thereby leading to lack of commitment from the customer.	Optimisation systems based on Decision Support Systems (DSS) should be used at least during the exploration phase in preference to black-box systems.
8. Requirements are vague, or customer is hesitating between several alternatives.	No efficient design of a solution can be prepared.	Use prototypes and graphical visualisation to help the customer in its choices.
9. The proposed results are very different from conventional or manual ones.	The customer does not trust the application or does not want to apply the results due to the important changes it requires.	Introduce in the model optional constraints that allow to control the distance between the proposed solution and the conventional ones. The customer will then progressively appropriate the system.

Appendix B. Problem definition document layout

I Business Case

X ltd is a Y million £ company,...
The coming market deregulation requires X to optimize their...
X is looking for a DSS that ...

II Problem organisational decomposition

X currently operates by solving the problem with the following organisational modules,...

$$\begin{tikzcd}
{} \arrow[r, "\text{Input}"] & \boxed{\text{M1}} \arrow[r, "\text{Output}"] \arrow[d] & {} \\
{} \arrow[r, "\text{Input}"] & \boxed{\text{M2}} \arrow[r, "\text{Output}"] \arrow[d] & {} \\
{} \arrow[r, "\text{Input}"] & \boxed{\text{M3}} \arrow[r, "\text{Output}"] & {}
\end{tikzcd}$$

Reconsidering this structure is possible if good operational or quality reasons

III LSCO requirements

Given a set of tasks and jobs to schedule
Find the optimal plan such that the following constraints are satisfied

Refine and structure the LSCO requirements by building the conceptual model

Input data: ...
Ouput data:...
Constraints:...
Decision criteria:...

IV Other requirements

Fuctional requirements (level of user interaction with the system)

Execution time between 30sec and 1 min, optimal solutions required,...
GUI: I want to be able to interrupt the optimisation and make decisions
All outcomes should be displayed as graphs and matrices
The menus should look as follows,...

Technical requirements (includes all hardware and software specifications if any)

Information System (IS) environment
Software components (programming language,...)
Hardware components
Operating Systems, Data base connexions

Business requirements

Project cost, and budget
Constraints on delivery dates,...
Documents to be produced (user guide, training material, maintenance documentation,...)

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