Constraint Logic Programming

Francesca Rossi*

Università di Padova
Dipartimento di Matematica Pura ed Applicata
Via Belzoni 7, 35131 Padova, Italy.
E-mail: `frossi@math.unipd.it`

Abstract. Constraint logic programming (CLP) is a multidisciplinary research area which can be located between Artificial Intelligence, Operation Research, and Programming Languages, and has to do with modeling, solving, and programming real-life problems which can be described as a set of statements (the *constraints*) which pose some relationship between the problem's variables. This survey paper gives a brief introduction to CLP, presents the state of the art in CLP research and applications, points out some promising directions for future applications, and discusses about how to cope with current research challenges.

1 Introduction and Main Concepts

In the last 10-15 years, Constraint Logic Programming (CLP) has evolved from a basic research idea to a powerful programming paradigm that is increasingly used to model and solve many hard real-life problems. Several companies have started up to exploit this technology, and the number of industrial applications is now growing very quickly. Also the ACM has recently recognized the power of CLP, by listing it in 1996 among one of the few areas of Computing Science which are classified as "strategic directions" [88].

A recent review [109] of an introductory book on CLP [124] succeeds in our view to describe the essence of constraint programming:

> One of the most important tasks of a programmer is to transform the functional specifications of a program into a language code that closely reflects the underlying physical architecture of the computer. In this process, he or she has to define objects and procedures that correspond to entities and operations in the application domain, then explicitly maintain relationships among programmer-defined entities, and finally find objects that satisfy them.
> Suppose you are tired of such tedious work. You want simply to state relationships between objects and ask your assistant (a program) to ensure that these relationships are maintained. You have then upgraded yourself from a traditional programmer to a constraint programmer.

* Other contributing authors: Frederic Benhamou, Manuel Hermenegildo, Francois Pachet, Helmut Simonis, Thomas Sjoland, Mark Wallace. These authors have provided a lot of useful material for the generation of this survey.

In fact, constraint programs are highly declarative, and allow to describe many real problems by using constraints, that is, statements which pose some relation among the problem's variables. Such a way of specifying problems can be introduced in many traditional programming environments, but the best match is found when using a declarative programming paradigm like logic programming (LP) [117], which is also based on relations.

More precisely, each constraint specifies a relation among a subset of the objects of the problem, where each object has a set of possible values. Thus a constraint restricts the space of possibilities for such objects. For example, in the description of an electric circuit, a typical constraint is that "the sum of currents flowing into a node must be zero". Or, in the problem of assigning offices to employers, one can have the constraint that "Mary and John must share an office, but John and Susan must have different offices". Or also, in a graphical interface system, one may want that "two graphical objects never overlap".

The description of a problem as a set of constraints is very declarative: it just says "what" has to hold, not "how" to make it hold. Many (partial or complete) solution techniques have been developed with the aim to find one or more *solutions* for a constraint problem, that is, assignments of variables to values such that all constraints are satisfied. The most widely known and used complete solution technique is based on assigning a value to each variable at a time, while satisfying all constraints involved in the assigned variables, until either all variables are assigned to a value (thus we have a solution), or there is some variable which cannot be assigned, in which case the *search* returns to a previous decision and chooses an alternative value for one of the variables. This method to search for a solution, usually called *backtracking search*, may in general explore the whole tree of possibilities before finding a solution, or discovering that there is none.

Many incomplete solution techniques have been developed to help backtracking search to have a better performance. Most of them are based on the concept of *constraint propagation*, which has to do with the possibility of using the information given by some constraints to infer necessary conditions over the constraints' variables. Moreover, such conditions can in turn help to deduce further consequences on other constraints, and so on. In this way, information is propagated among constraints passing through the shared variables. Assume for example to have a graphical layout system, where, for any two graphical objects, we have the constraint that they cannot overlap. Suppose now that we use the mouse to move one of the graphical objects in an area where there are already others objects. Then, by constraint propagation, such objects will be moved to free areas, or, if there is no free area which is big enough for them, possibly many objects will be moved around in the screen to reach a situation where no two objects overlap. Constraint propagation was initially used in the 70's for scene-labeling applications [129, 182], but since then it has been extended and adapted to many other contexts, producing a high number of propagation algorithms, the most influential being [129, 120, 119, 50, 114, 121, 52]. Many constraint systems have been developed: each one has one or more constraint classes in mind, and for those classes it provides both complete and incomplete solution algorithms, usually combined together. Examples of classes of constraints that have been considered and for which efficient propagation and solving techniques have been built are finite domain constraints, arithmetic constraints, linear constraints, boolean constraints, and interval constraints.

Being able to model and solve a real-life problem with the available constraint classes and solving techniques may seem enough, but in fact in most cases it may be crucial the possibility to directly control both the search and the propagation. In fact, one may want to choose the propagation technique on the fly, depending on the particular application, and also to control how the search tree is explored. For this reason, usually constraint systems (providing some built-in propagation and solution algorithms) are embedded into a high-level programming environment, which allows to have the desired control. The fact that both constraint systems and logic programming [117] are declarative frameworks based on relations and backtracking search made their marriage easier and more natural. More precisely, when it became clear that term unification was just a special form of constraint satisfaction, LP was generalized by replacing terms with constraints, and unification with a suitable constraint handling system. The resulting programming paradigm, called Constraint Logic Programming (CLP), has been the basis (not necessarily syntactically) for many commercially successful systems which model and solve real-life problems by using the constraint (logic programming) technology. In particular, CHIP [56] was the first CLP language to employ constraint propagation [84]. Other examples of CLP-related systems are the constraint handling libraries of ILOG [96] and COSYTEC [46], and the CLP languages Prolog III [43,44], CLP(R) [99], ECLiPSe [95,180], CIAO [91], and clp(fd) [37]. With these systems, many hard application domains have been tackled successfully using constraint-related technologies. Examples are circuit verification, scheduling, resource allocation, timetabling, control systems, graphical interfaces, and many combinatorial problems [177,84].

The presence of constraints within a language is useful in many respects. In fact, the direct representation of a problem in terms of constraints results in short and simple programs. Moreover, the embedding of constraint-based techniques into a high-level language gives the right level of abstraction for the programmer, who can concentrate on choosing the best technique combination for the problem at hand. Also, since programs can be quickly developed and modified, it is possible to experiment with various ways of solving a problem until the best and fastest program is found.

While the birth of CLP languages was based on the observation that constraints could be used to model real-life statements better than term equalities, and also to control the search for a solution, another generalization came to life when it became evident that constraint could also be used to naturally model agents' communication and synchronization in concurrent systems. In fact, agents can communicate by posting constraints on shared variables, and can synchronize by checking whether a certain constraint has been posted by some other agent. This observation was the main idea underlying the concurrent constraint (CC) programming framework [154], which posed the theoretical foundations for the birth of other concrete languages, like AKL [81, 131, 102] and Oz [89, 170, 82, 93], which can model reactive and distributed systems via CLP-related technology.

The rest of this survey paper is organized as follows. First we will describe the state of the art in constraint-related research in Section 2, and then we will attempt to give an overview of the current scenario for constraint-based applications in Section 3. In Section 4 we will discuss about the relationship between theoretical research and application work, by looking at the proceedings of the CP conferences, and in Section 5 we will report on the results of a questionnaire concerning CP-related applications. Then, in Section 6 we will point out some directions for future opportunities in applications using constraints, both in currently considered domains and also in new ones, and in Section 7 we will discuss some current challenges for constraint-related research.

2 Research in CLP: state of the art and analysis

The state of the art in CLP research and application is mainly reported in the proceedings of the annual international conference on Principles and Practice of Constraint Programming (CP) [130, 64, 169, 122, 98] and of the conference on Practical Applications of Constraint Technology (PACT/PACLP) [176, 178, 179, 132, 116], and also in the recently established CONSTRAINTS journal by Kluwer Academic Publishers. Other related conferences and journals also report work on CLP, mainly those in areas like Artificial Intelligence (Joint International Conference on AI – IJCAI, Conference of the American Association for AI – AAAI, European Conference on AI – ECAI, The AI Journal), Logic programming (International Logic Programming Symposium – ILPS, Journal of Logic Programming), Databases, and Operation Research.

In this section we will try to give an overview of the current research efforts in CLP, by listing the main areas of research related to CLP. Notice that some research work concerns more than one of the areas listed below. As an example, constraint propagation has to do with finite domain constraint *solving*, but it is used also in constraint *programming*; in these cases, we have chosen to describe it within the area which is more central to the work (in the above example, constraint solving).

Another important remark is that the following list is far from being complete. One of the reasons is that a complete list covering all aspects of CLP research could easily fill several books. However, both the incompleteness and any other drawback of this paper should be attributed to the main author, who did her best but could not avoid, among other faults, following a bias towards more familiar research issues. Nevertheless, fortunately many other survey papers on CLP are present in the current literature [177, 100], as well as introductory papers [40] and books [124, 139, 84, 153, 14, 175]. Thus the reader may refer to such sources to fill the gaps of this paper.

Finite domain constraint solving. The origins of finite domain constraints are to be found in Artificial Intelligence (AI), where finite domain constraints were first introduced and constraint propagation was used to solve picture processing problems since the 70's [129, 182]. Many research efforts still lie in the AI area.

Since solving a finite domain constraint system is an NP-hard problem, much work has focussed on the identification of *tractable classes* of constraint problems, that can be solved in polynomial time (see [114, 121, 175] for an overview of many algorithms). Such classes are usually identified by a precise graph structure (like a tree [65, 149, 51]) or by a precise combination of algebraic operators [107]. Recently, graph decomposition strategies and also constraint propagation techniques have been used also for continuous constraint problems [22, 152, 41], and for problems where constraints represent differential equations [53]. For some classes of constraint problems, it has also been recognized that enforcing a low level of constraint propagation guarantees a backtrack-free search [52]. Recently, most of the graph decomposition method for constraint problems have been compared, and the results suggested that the best method is based on the concept of hypertree decomposition, developed in database theory [79].

Recently, the concept of a constraint language has been carefully formalized, so that its expressive power can be studied and it can be told if constraint problems expressed in a certain language are tractable [104–107, 97, 39, 108, 103].

For both tractable and non-tractable constraint problems, it is crucial that the backtrack search process performs as good as possible. For this reason, much work has been devoted to study several heuristics which may influence the behavior of the search engine, like the variable and value orders [141] and also the amount and kind of look-ahead to perform at each step [115].

Another issue that has received great attention recently is the possibility of locating the "really hard" problems. It turns out that when constraint problems are generated randomly, for example to evaluate the behavior of a new algorithm, most of the generated problems are easy. Thus, special care is needed to make the random generators produce non-trivial problems. It has been recently demonstrated that for many random generators there is a phase transition from easy to hard problems, and that hard problems are located where only few solutions exists [32, 168, 140]. Current on-going work is mainly concerned with finding new transition phases (for new algorithms, or new classes of constraint problems) [72], testing new algorithms on hard problems, and evaluating the power of the random model [2, 118].

Programming with Finite Domain Constraints. Many CLP languages have been built to deal with finite domain constraints. The main examples are CHIP [56], Eclipse [95, 180] and clp(fd) [37]. These languages have a wide applicability, and raise a number of important research issues.

Much work on finite domain CLP is centered on the introduction of *global constraints*. The notion of a global constraint is basically as follows: there are cases in which having one bigger constraint (bigger in the sense of the number of its variables) may be better with respect to a set of smaller constraints. A typical example concerns a set of disequality constraints, which seldom give rise to useful constraint propagation. In such cases, it may be necessary to perform a complete search among the domains of all the variables to find one solution. This useless expensive search can be avoided by using a constraint called *alldifferent* which connects all the variables. In fact, by using this one constraint, the problem of finding a solution can be reformulated as a bi-partite matching problem between variables and values and classical graph algorithms can be applied, which can help to derive enough information to reduce the search. In general, the introduction of global constraints works as follows: instead of expressing a problem by many primitive binary constraints, we describe it with a few global constraints on sets of variables. In this way, we can model problems in an easier way, we have a smaller number of constraints, and more sophisticated and efficient algorithms can be used. However, global constraints should not be created *ad hoc* for each application, but should be general enough to be used in many applications. Typical examples are the *alldifferent* constraint cited above, the *atmost* constraint, which sets a limit on the number of variables with a certain value, and the *cumulative* constraint.

Languages like CHIP [56] have a complex global constraint morphology including these and many other global constraints [11]. Other languages, like Sicstus Prolog [29], Eclipse [95, 180], IF Prolog, and Oz [89, 170, 82, 93] include variants of some of the global constraints used in CHIP.

As noted above, when introducing new global constraints, one has also to provide efficient handling mechanisms for them. An example is the filtering algorithm proposed in [144] for global sequencing constraints, and used for difficult car sequencing problems. Other studies on the use of constraint propagation over global constraints (in particular, the alldifferent constraint) show the practical value of using this kind of constraints [159, 143]. Global constraints have been recently used also for partial constraint satisfaction problems, where some constraints must be violated to find a solution, showing a big potential for an efficient handling of such problems [7].

As said in the introduction, one of the main advantages in embedding a constraint system within a CLP language is that we can control the search process. In the last few years, the use of *partial search techniques* has significantly improved search methods. Partial search techniques limit the number of alternatives explored in different nodes of the search tree. In this way, the usual depth-first backtracking search used in CLP, which sometimes explores only a small portion of the search tree, can be replaced by other search schemes (like LDS [83] or credit-based search [10]) which can consider many different paths in the search tree, without much implementation overhead.

Dynamic constraint problems are constraint problems which change over time, by either the addition or the deletion of some constraints, or a combination of these two operations. While constraint addition is what usually is modeled at each computation step of a CLP program, constraint retraction is seldom considered, although it would very useful in many scenarios, like for interactive or dynamic systems. For this reason, some effort has been devoted to embed the notion of constraint retraction, and also some efficient algorithms to accomplish it, within classical CLP languages. For example, the clp(fd) programming framework has recently been extended with both a syntax, a semantics, and some efficient methods to handle the possibility of retracting a constraint [74]. Notice that constraint retraction allows also the modeling of hypothetical and interactive reasoning. Another aspect of dynamic constraint problems is that, once a solution is found, there may be changes which invalidate such a solution. This problem can be addressed either by developing efficient ways to find a new solution from the old one, or by generating stable solutions, that is, solutions which are more likely to remain as such even after a certain class of changes [181].

Non-linear constraints. The possibility to handle and solve non-linear constraints over real numbers is important for many areas (like economics, mechanics, engineering, ...). Problems involving such constraints are usually very expensive computationally, thus it is often hard to ensure termination and to guarantee finding all solutions. Beside classical techniques, now many tools use interval methods for solving this kind of problems. Also for this reason, interval constraints have been introduced in many CLP languages, like ECLiPSe [95, 180, 183] and CLP(BNR) [16]. In particular, both the Helios language [85] and Numerica [86] have been designed appositely to solve non-linear constraint systems using interval analysis algorithms, which combine techniques from numerical analysis and artificial intelligence. These language allow to state many nonlinear applications in a very high-level formulation, thus letting the user focus on modeling aspects instead of low-level details, and has been shown to outperform traditional constraint solving methods and interval algorithms on many constraint-solving and global optimization benchmarks.

A special issue of the Constraints journal has been devoted to the subject of interval constraints [15].

Modeling. It is very important to have a good modeling language to express constraint problems. At the moment, most constraint systems are either extensions of a declarative programming language (like Prolog) or libraries used together with conventional programming languages (like C or C++). Currently there is a lot of work in trying to introduce constraint modeling languages that make the modeling phase easier and more natural (already cited examples are Helios [85] and Numerica [86]). Some of these languages make use of visual tools to describe and generate constraint programs.

For example, in the FORWARD refinery scheduling system [76], which uses a graphical specification tool for CHIP [163], the model of a refinery is entered via a graphical drawing tool and the constraint model is generated from the layout. CML is a constraint modeling language which has been thought for facilitating the tasks of stating and modifying combinatorial problems [3]: CML problem specifications allow for constraint formulations that reflect natural language expressions, and that are then translated into efficient CHIP programs. Another recently developed modeling language is VISUAL SOLVER [87], which allows to specify both the problem statement and the heuristics to be used to find solutions.

Other recent approaches to modeling a real problem as a constraint problem use techniques related to negotiation: the constraint solver interacts with a human providing partial solutions (suggestions) based on partial knowledge, and gaining further information about the problem from the human through his/her evaluation of the suggestions [67]. Besides negotiation, also other techniques, which have not been familiar to the constraint community so far, have been used to help the user in the modeling phase. An example is the use of machine learning techniques: in the context of soft constraints (see later), the user provides the system with some examples of solutions ratings, and the system learns the whole constraint problem [150]. In this way, and also by using an incremental acquisition/learning technique, the task of modeling a real life problem as a (soft) constraint problem is greatly simplified.

Debugging. One common problem for users of the CLP systems in use is the sometimes unpredictable behavior of the constraint model: even small changes in a program or in the data can lead to a dramatic change in the performance. This is because the process of performance debugging, designing and improving constraint programs for a stable execution over a variety of input data, is currently not well understood. Related to this problem is the long learning curve that novices have experienced. While simple applications can be built almost immediately, it can take a long time to become familiar with the full power of a constraint system. One attempt to solve these problems considers the possibility of visualizing the search tree [30]. Tools like the Oz Explorer [157], the CHIP search tree tool [164], or the system recently developed within the CIAO system [91], which provides a 3D visualization via VRML [167], allow a much deeper understanding of the search process, since they provide enough information to analyze propagation and the state of the variable domains at each node of the tree. A general and comprehensive approach to CLP debugging has been taken within the DiSCiPl European research project (22532), where several tools have been developed to help the programmer during all phases of development of a constraint program. Such tools are used for static analysis, for error diagnosis, for search space analysis and for constraint propagation analysis: no single tool is powerful enough, but their combination makes them effective enough to provide a good debugging environment.

Soft constraints. By developing real applications, it has increasingly become clear that finite domain constraints were not enough to model faithfully a real problem. The main problem is that real problems usually cannot be simply described by a set of true/false statements, like classical constraints are, because they have features like preferences, probabilities, costs, and uncertainties. Also, many real problems, even when they are correctly modeled, usually are overconstrained [101]: the user puts so many constraints that its is impossible to satisfy all of them. In these cases, we first have to find out that there are no solutions (and it may take a long time to do this), and then we have to decide which constraints to relax so as to make the problem soluble. For these reasons, the concept of soft constraint was introduced: a soft constraint is just a constraint plus an additional element, which can be interpreted as a cost, or a level of importance, or a degree of uncertainty, or a similar criterion. Then, finding a solution does not mean any longer to find a way to satisfy all constraints, but to obtain an assignment of the variables to values which has the "best" value for the chosen criterion. Additional complexity is present when we have different criteria (like to maximize the levels of importance, and also to minimize the sum of all costs). Research in this area first started with the *hierarchical CLP* (HCLP) system [25], a CLP language where each constraint has a level of importance and a solution of a constraint problem is found by respecting the hierarchy of constraints. Then much work on *fuzzy constraints*, where each constraint has associated a value between 0 and 1 and we have to maximize the minimum of such values, has been done to formalize the new framework and study efficient solution algorithms for it [58, 151, 60, 155]. A combination of constraint hierarchies and fuzziness has also been used [110]. Overconstrained problems have also been tackled using the notion of *partial constraint satisfaction*, where a metric was added to the constraint problem to help to produce a solution that satisfied as many constraints as possible [66].

Lately some theoretical work has focussed on the development of more general frameworks for soft constraints, like the *semiring-based formalism* [18, 20, 19], where each tuple in each constraint has an associated element taken from a semiring, and the *valued constraint formalism*, where each constraint is associated to an element from a totally ordered set [156]. For such general formalisms, the main efforts are in trying to extend and adapt the propagation and solution techniques typically used for classical finite domain constraints, and to study their properties. Recently, the finite domain CLP language clp(fd) [37] has been extended to handle semiring-based constraints, obtaining a language paradigm called clp(fd,S) [73] where S is any semiring, chosen by the user. By choosing one particular semiring, the user decides to use a specific class of soft constraints: fuzzy, or optimized, or probabilistic, or even classical hard constraints. For valued constraints, we do not have yet full CLP languages working with them, but we have many techniques to achieve good lower bounds for their optimal solutions, which in most real cases seem to be good enough [49, 28].

Constraint query languages. As noted above, constraints are just relations, thus it is obvious that many researchers have investigated the relationship between CLP technology and Databases (DB). In this area of research, there are mainly two ways in which constraints and databases have interacted so far. On one side, results on DB (for example, on decomposition and information derivation) have been generalized and adapted to constraint problems, generating useful tools to test the tractability of some classes of constraint problems and efficiently solve such classes [80, 38]. On the other hand, classical query languages in DB have been extended to deal with constraints, which are used not only to specify integrity information, but also as a mean to represent in a compact way a possibly infinite set of DB tuples. In fact, a relational tuple can be seen as a particular type of constraint: more precisely, a conjunction of equality constraints between attributes of the tuple and values from a given domain. The introduction of new logical theories to express relationships (that is, constraints) between the attributes of a DB item lead to the definition of *Constraint Databases* as a new and interesting research area [112, 26].

Constraints have been added to relational DBs at different levels. At the data level, constraints may represent generalized tuples that are able to specify possibly infinite sets of tuples. In this sense, for example, constraints are a powerful mechanism to model spatial and temporal concepts that arise in DBs, where often infinite information has to be represented [13]. To deal with constraints, both the relational calculus and the relational algebra of classical DBs have been extended [111, 112, 77], leading to new and more expressive query languages for which many properties are being studied. An example of such languages is obtained by adding linear constraints to Datalog, which has been studied in [147]. Other examples are safe stratified Datalog, considered in [146], and the integration of linear constraints with DBs, for which query optimization has been studied in [27]. Query optimization has also been studied in [173] in the context of deductive queries over constraint DBs.

Concurrent and distributed systems. The use of constraints in modeling the behavior of concurrent systems has started with the development of the concurrent constraint (CC) programming framework, where concurrent agents share a store of constraints where they may either post some more constraints (tell operation), to communicate with the other agents, or ask whether some constraint is already present (ask operation), to synchronize with the other agents. We already said in the introduction that some real languages, like Oz [89, 170, 82, 93], AKL [81, 102, 131], CHR [70], and CIAO [91], have taken up this model and implemented it (although each one differently), thus providing a programming framework for concurrent systems.

Recently much research effort has tried to extend this model to consider also distributed systems, where the store is not shared by all agents but is rather a collection of local stores, and also software mobility, like that present in web-based agent systems. In this respect, both languages and new solution algorithms have been proposed, by adapting existing techniques to a distributed context (for example in [184]). Another line of extension of the CC framework is concerned with the introduction of time, in order to be able to model and program real-time applications [48].

One of the advantages of the languages based on the CC framework is that, via the ask operation, it is possible to describe within the language many algorithms or techniques (like constraint propagation methods) which in sequential CLP languages have to be provided by the system. This added flexibility is used for example in the language Oz, where the programmer has all the necessary tools to program his/her own inference engines [158].

Comparison/combination of various techniques. Some of the real problems that are currently solved by using CLP technologies can also be solved by Operation

Research (OR) techniques, with complementary advantages: CLP offers generality, flexibility, a high-level modeling environment, compactness of the problem representation, constraint propagation, and fast methods to achieve a good solution, while OR offers search control and sometimes it is the only one to find the optimal solution. Therefore, it seems a good idea to try to get the best from these two kinds of technologies, by building systems which can use both of them to solve a problem. This approach has been used already in some systems, like the 2LP system [127, 126], as well as in [47], which uses this combined approach for specific telecommunication problems. However, we are still at the beginning of this research area.

3 CLP applications: state of the art and analysis

In this section we will give an overview of the main application domains in which CLP technology has been applied and has shown to be competitive both in terms of modeling flexibility and solution efficiency. As in the previous section, we have structured this overview by listing the main application domains; however, in most cases a CLP application has aspects related to several domains. In these cases, we have chosen to describe it within the domain which is closer to the main aim of the application. As for the research overview, also this application survey is far from being complete, although we believe that most of the main application domains are represented here. A good survey paper on CLP applications, from which we took many useful points and insights, is [177].

Assignment problems. Assignment problems were one of the first type of industrial applications that were solved with the CLP technology. These problems usually have to handle two types of resources, and constraints among them, and try to assign one resource of the first kind to one of the second kind such that all constraints are satisfied.

An example is the stand allocation for airports, where aircrafts (the first kind of resources) must be parked on the available stands (the second kind of resources) during their stay at the airport. The first industrial CLP application was developed for the HIT container harbor in Hong Kong [137], using the language CHIP: the objective was to allocate berths to container ships in the harbor, in such a way that resources and stacking space for the containers is available. Other Hong Kong applications are at the airport, where a CHIP-based system is used to solve the counter allocation problem [34], and another constraint-based system, which uses the ILOG libraries, is used for the stand allocation problem since mid-1998 [35]. Another system, called APACHE [55], was a demonstrator for stand allocation at Roissy airport in Paris: the objective was to replan the allocation when a change of arrival/departure times occurred.

Network management. Another application domain for finite domain CLP is network management, where many different problems can be addressed and solved using CLP.

The LOCARIM system was developed by COSYTEC for France Telecom: starting from an architectural plan of a building, it proposes a cabling of the telecommunication network of the building. The PLANETS system, developed by the University of Catalonia in Barcelona for the Spanish electricity company, is a tool for electrical power network reconfiguration which allows to schedule maintenance operations by isolating network segments without disrupting customer services. The company Icon in Italy produces a load-balancing application which is controlling network flow for the inter-banking system in Italy. The Esprit project CLOCWiSe (IN 10316I) is using CHIP for the management and operational control of water systems. The planning of wireless digital networks for mobile communication has been tackled by the system POPULAR [69], written in ECLiPSe and then in CHR: the main advantages with respect to other approaches to this same problem (using traditional imperative programming) are a good balance among flexibility, efficiency, and rapid prototyping.

Scheduling problems. Perhaps the most successful application domain for finite domain CLP are scheduling problems. Given a set of resources with given capacities, a set of activities with given durations and resource requirements, and a set of temporal constraints between activities, a "pure" scheduling problem consists of deciding when to execute each activity, so that both temporal constraints and resource constraints are satisfied.

A typical example of a constraint-based scheduling application is ATLAS [166], which schedules the production of herbicides at the Monsanto plant in Antwerp. The PLANE system [12] is used by Dassault Aviation to plan the production of the military Mirage 2000 jet and the Falcon business jet. The objective is to minimize changes in the production rate, which has a high set-up cost, while finishing the aircraft just in time for delivery. The MOSES application was developed by COSYTEC for an animal feed producer in the UK: it schedules the production of compound food for different animal species, eliminating contamination risk and satisfying costumer demand with minimal cost. The FORWARDC system is a decision support system, based on CHIP, which is used in three oil refineries in Europe to tackle all the scheduling problems occurring in the process of crude oil arrival, processing, finished product blending and final delivery [76]. Recently, Xerox has adopted a constraint-based system for scheduling various tasks in reprographic machines (like pothocopiers, printers, fax machines, etc.); the role of the constraint-based scheduler is to determine the sequence of print making and to coordinate the time-sensitive activities of the various hardware modules that make up the machine configuration [68]. Recent results on the tractability of classes of constraint problems have shown that such scheduling problems are indeed tractable, and thus amenable for an efficient solution [142].

Preemptive scheduling problems are those scheduling problems where activities may be interrupted over time. Also for these problems CLP technology has shown its potential: the CLAIRE SCHEDULE is a constraint programming library which can successfully handle these problems, also in the presence of preferences [134, 135].

By comparing the way scheduling problems are tackled by using imperative languages (C), generic CLP languages (CHIP), and specific constraint programming tools (CLP over sets), a recent study (on a specific scheduling problem: the cyclic hoist scheduling problem) has shown that the CLP approach is superior in many respects: development time, nodes visited in the search tree, number of generated feasible solutions, even efficiency [5]. According to this study, the main reason for the success of CLP on these problems is the use of constraint propagation, which helps to reduce both the development and the execution time when the solving algorithms are not precisely known.

It has also been realized that, for some classes of scheduling problems, there is no one technique which is better than all the others, but rather a combination of several techniques is the best approach. Thus hybrid algorithms, employing both CLP and other techniques, have recently been developed. An example is the combination of CLP and Integer Programming, that has been used to tackle multiple-hoist scheduling problems [148].

Transport problems. A variety of transport problems have been tackled using constraints. These problems are often very complex due to their size, the number and variety of constraints, and the presence of complex constraints on possible routes. Moreover, often these problems have a personnel allocation problem as a sub-aspect, usually with complex safety and union regulations.

The COBRA system [165] generates diagrams representing workplans for train drivers of North Western Trains in the UK. For each week, around 25.000 activities must be scheduled in nearly 3.000 diagrams, taking a complex route network into account. The DAYSY Esprit project (8402) and the SAS-Pilot program [8] considers the operational re-assignment of airline crews to flights. This same problem is tackled also by another system [63], which uses a combination of CLP and OR techniques. A recently developed system uses the ILOG constraint libraries to produce and optimize train operating plans for a freight railway company, by creating transport movements for rolling stock between sources, hubs and sinks while satisfying transport requirements [125].

Also for transport problems (as for scheduling problems), in some cases the best method employs a combination of several techniques. For example, CLP and local search were both used to solve vehicle routing problems in [161], and in [138] the same problems are tackled via a combination of CLP and OR techniques.

Personnel assignment. Personnel assignment problems are another typical application area for finite domain CLP. Changing work rules and regulations impose difficult constraints which evolve over time. Also, user preferences often lead to over-constrained problems, which have no solution satisfying all constraints. Another important aspect is the requirement to balance the work among different persons, which leads to hard optimization problems.

The Gymnaste system [31] produces rosters for nurses in hospitals, and is used in many hospitals in France. Around 250 technicians and journalists of the French TV and radio station RFO are scheduled with the OPTI-SERVICE system [42], which generates weekly workplans from the individual activities which must be covered by different persons with different qualifications. The personnel for the TGV high-speed train bar and restaurant services is scheduled with the EPPER application [59]: all services for a month are covered by a team of 250 people. Recently a distributed CLP system has been used to tackle a workforce management problem within British Telecom [113]. Also Telecom Italia is using a constraint-based system to schedule all its technical tasks (about 100.000 tasks involving 20.000 technicians) over its territory [136]; the system which controls the whole workforce management has a scheduling module, called ARCO, which dispatches activities to technicians, and which makes extensive use of constraint propagation techniques.

Controlling electro-mechanical systems. CLP technology is also being exploited to build control software for electro-mechanical systems with a finite number of inputs, outputs, and internal states. Each component of a complex system is connected to a small part of the overall system, and its behavior can be captured quite simply, but when the system is considered as a whole the number of global states can become very large. Thus a smart technology that is able to handle such combinatorial explosion is needed.

Control systems are needed for many applications, ranging from lifts, photocopiers and car engines, to assembly lines and power stations. The applications currently being tackled with CLP technology are those at the smaller end, where it is still possible to prove that with a given control certain global properties of the system can be guaranteed. Typical properties that must be proved are safety properties (for example, that a lift should never start to move while the doors are still open), fairness properties (for example, that the lift must eventually answer all requests), and liveness properties (for example, that the lift eventually will stop). A system tackling one of such applications and proving some of these properties is the system verification environment SVE [62]. Another one, mainly concerned with the verification of liveness properties in parallel systems, uses a combination of Petri nets and constraint programming within the 2LP programming language [128].

Constraint-based spreadsheets. The spreadsheet is one of the most prevalent decision support software tools in use today. However, usual spreadsheets have two limitations: they only compute in a fixed direction (from input cells to output cells), and the outputs can be defined only when the inputs are precisely defined. On the other hard, constraints allow variables to influence each other in all directions (like in $X \leq Y$), and work naturally with partial information (such as a variable with a non-singleton domain). The idea to apply CLP technology to overcome these two limitations of the current spreadsheets has been taken up by several systems.

The Short Term Planning (STP) application at Renault [33] assigns product orders to factories so as to minimize transportation costs for delivering cars to costumers. This is a transportation problem with a lot of side-constraints, and usually it is unsolvable, so some of the constraints have to be relaxed. However, the absence of a fixed priority order or cost function, and the continuous changes in the context of the problem, requires that many decisions have to be taken by the end user. A constraint-based spreadsheet was developed to help the user in this task. During the planning process, the figures contained in the spreadsheet are not precise numbers, but intervals (min-max). Constraints are attached to different cells and are used to propagate the consequences of modifications imposed by the user, who can either enter a specific value for a cell (as usual), or just restrict the range of possible values by narrowing the interval. The system propagates the consequences of such user choices by making them explicit: either an inconsistency is found, or tighter bounds are deduced for other variables. When the problem formulation is satisfactory, the user invokes a linear programming algorithm to produce an optimal solution. Since February 1995, the production of all Renault cars has been planned with the STP system.

Another promising application of constraint-based spreadsheets is to financial planning [94]: the spreadsheet allows the financial planner to explore potential investments. Financial applications often involve non-linear constraints which are hard to handle with mathematical programming techniques. Reasoning on intervals makes this easier, since it is possible to compute the min and max of a non-linear function given the min and the max for each of its inputs. The problem of planning the budgets of the districts of Moscow, and to control that such plans are followed, has been recently tackled by using a constraint-based spreadsheet where the sub-definite constraint solving method is embedded within the ECLiPSe language [162].

Another example of a constraint-based spreadsheet is the Configuration Spreadsheet system [145], which helps users to obtain a consistent and optimal configuration for problems like configuring a radio base station, or setting the parameters of mobile phone network transceivers.

Interactive problem solving. For many applications, the end user must be able to cooperate with the software in developing solutions. CLP technology, especially constraint propagation, supports precisely the kind of feedback necessary for these applications. In fact, at each step the consequences of the particular choice are made explicit by the propagation engine. The user can then either reject the choice, or use the propagated information to make the next choice, or allow the system to continue the search automatically.

For example, constraint programming systems with this kind of behavior have been developed to address the timetabling problems of Banque Bruxelles Lambert [57] and of the Medical Faculty of the Humbolt University [78]. Another system [61] assigns cashiers for El Corte Ingles in Spain, and another one [17] allocates gates at Changi airport. This last system supports several different Gantt chart displays which allow to manipulate graphically the resource assignments. Yet another system allows for the interactive construction of a solution for a nurse scheduling problem, and it has been tested at a clinic in Munich [1].

The need for interactivity may also arise from the fact that in some applications it may not be feasible to acquire all problem features at the beginning. This is what may happen for example in 3D object recognition problems, where visual feature acquisition is a very time-consuming task. For this reason, a recently developed system, based on ECLIPSE, allows to have only some features present when starting the solution process, and to interactively acquire the others on demand only when needed [71].

Another class of problems that may require interactivity are those which are ill-defined, and thus require further refinement. Examples of such problems are the speculative constraint problems, which arise when one wants to anticipate future changes and decide what to do accordingly. A recent study of these problems, applied to the specific problem of risk management in energy trading, has shown the potential of constraint technology to handle also this kind of interactivity [75].

Graphical interfaces. Another large (and one of the oldest) application area for CLP technology is graphical interfaces. The role of CLP here is to keep all the graphical objects in the correct relation to each other, after some update to the graphical window – typically via a mouse. Thus the objective is not to find an optimal solution, but a natural one, and relatively quickly. Also, constraint propagation in graphical interface applications has to be fast and has to propagate changes, not just refinements; moreover, inconsistencies are not possible, and must therefore be handled by making further changes. For example, in an electronic room planning application, it might be possible to drag a graphical chair across a graphical room, but the constraints should prevent it from occupying a position which overlaps with the position of any other piece of furniture in the room [92].

In reality, constraints and graphical interfaces may interact in two ways: on one side, constraints may be used to provide added features to graphical interfaces. On the other side, graphical interfaces (either based on constraints or on traditional techniques) may be useful in many CLP applications to help the user in the modeling phase. In the following we give examples of both kinds of systems.

A commercial constraint library for constructing interactive graphical user interfaces is presented in [24]. The company Object Technology International has used this library to build a graphical editor for a large database. A recently developed object-oriented constraint solving toolkit, called QOCA, provides a simple and flexible interface for interactive graphical applications [123]. Such a system has been used in graphic editors, for the error correction in visual language parsing, for graph layout, and for interactive document layout on the Internet. Another constraint solving algorithms, especially designed for interactive graphical applications, is Ultraviolet [23]. Constraints have also been used to add useful features and efficiency to graph drawing algorithms [172], and to generic visualization and animation tools [171].

A graphical interface using constraint-based techniques has been developed to solve the assembly problem (that is, the spatial joining of separate rigid bodies) within professional CAD/CAM systems [160]. The language EaCL [174] has been designed to help users to specify constraint satisfaction problems; it relies on the ZDC user interface, which allows to specify variables, domains, and constraints, and provides also on-line tutorials and examples.

Overconstrained problems. Many real problems are overconstrained: the user puts so many constraints that it is impossible to satisfy all of them. In these cases, we first have to find out that there are no solutions (and it may take a long time to do this), and then we have to decide which constraints to relax so as to make the problem soluble. To solve this second problem, one approach is to ask the end user to make some decisions, as in the interactive problem solving applications described above. Otherwise, we can associate a cost to each constraint, or form a hierarchy of constraints (where higher constraints are more important) [25], or, more generally, resort to soft constraints [66, 58, 151, 156, 20].

Typical over-constrained problems are timetabling problems, where constraints come from many sources (like the availability of rooms and all the professors' preferences). A recent system uses both hard and soft constraints to model and solve time-tabling problems at the University of Siegen, Germany [9], and another system employs two phases of constraint satisfaction (the first one to assign faculty to courses, and the second one to assign time slots to courses) to solve the timetabling problems of the Computer Science program of the Florida Institute of Technology [21]. Another systems, which tackles over-constrained problems by using a hierarchical constraint system, is currently tested at the DRK hospital Neuwied to solve the nurse scheduling problem [4].

Other application domains. A hard problem that has been recently tackled using the CLP technology is the *protein structure prediction problem*, which is one of the most important problems in computational biology [6]. This problem consists of finding the conformation of a protein with minimal energy. It seems that the use of constraints helps to reduce the amount of search involved in this NP-complete problem.

Constraints have also been used by Sony for several applications related to *music* [133]. One of the problems tackled by Sony is the automatic harmonization problem: given a melody, the task is to compute a four-voice harmonization of the melody, which satisfies the rules of harmony and counterpoint. The rules of harmony are very natural to describe as constraints, since they typically state incompatibilities. The Sony application solves this problem by using structured variables which represents chords, and by using arc-consistency-based algorithms. Another Sony application related to music is the Active Listening project, which allows a music listener to set some constraints, which may state for example that some instruments are linked together. A typical constraint of this kind may state that the volume of the drum and the bass must be equally strong, or that there must be a balance between two sound sources. During the play of the music, the listener may change the volume of some instruments, but the system makes sure that all the constraints must be satisfied. A last Sony application of constraints to music is the RecitalComposer project, which produces sequences of music titles, taken from a large-scale catalogue, according to certain user preferences, some global properties on the coherence of sequences, and some constraints on the access to the catalogue.

4 Research vs. applications in the CLP community

By looking at the proceedings of the CP conferences [130, 64, 169, 122, 98] since their birth in 1995, we tried to understand whether the current CLP community is working more on using the current technology to build useful and efficient applications or on extending/adapting/modifying the technology from the theoretical point of view to be able to, later, tackle even more complex applications in a better way. The graphic in Figure 1 shows the percentage of application, constraint programming, and constraint solving papers in the proceedings of all the CP conferences held so far. We did not use the proceedings of the PA-CLP conferences for this graphic because they are too biased toward application papers, which make almost 100% of the accepted papers every year.

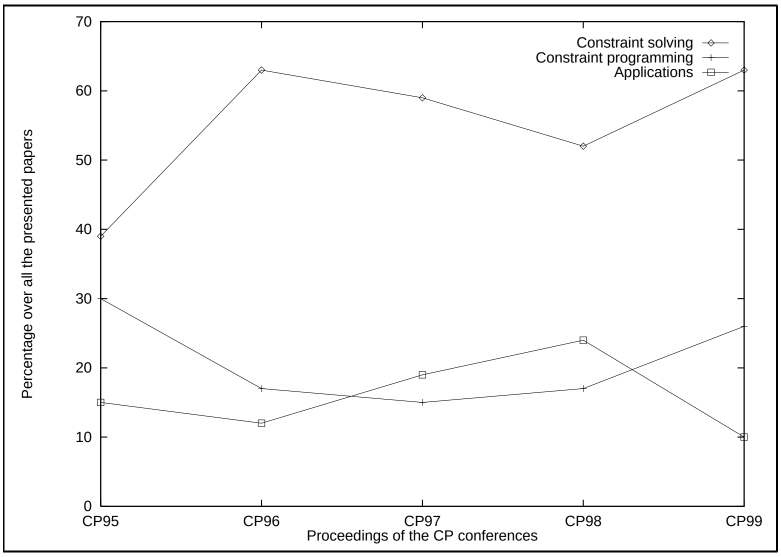

Fig. 1. Theory vs. applications in the CLP community.

Constraint solving is more than half of the work. The main thing we noticed in these results is the large percentage of papers on constraint solving (always more than 50% in the last three years) over all the other kinds of papers. This is in some sense reasonable: constraint solving is needed both for solving constraint problems and also for programming with constraints. Thus the results of such papers are useful for the vast majority of the CLP community. However, another reason for the presence of so many constraint solving papers in the proceedings of the CP conferences may also be the fact that usually this kinds of papers are self-contained and gracefully balanced (they usually contain a new algorithm plus the experiments), and thus they can be more easily appreciated by the program committees.

Application papers. Another point is that the percentage of application papers is not so high: at most 24%. Although this may seem strange, it is understandable by considering that there is an annual conference (PACLP) entirely devoted to this subject. However, at least 10% of the paper are always on applications. This means that the CLP technology has so many application domains that original and interesting application reports can easily exceed the space of an annual conference devoted to them.

Constraint programming papers. The third line represents the percentage of constraint programming papers, where by constraint programming we mean both CLP and also other kinds of programming paradigms using constraints (like imperative, object-oriented, ...). In the three years before the last one such a line of research has contributed for at most 17% of the papers. This may suggest that in those years people were more interested in supplying existing constraint programming languages with new constraint solving techniques (thus the high number of constraint solving papers) rather than developing new languages for constraint programming. The recent trend in producing more constraint programming papers may be due to the need of augmenting existing CP languages with useful tools that make CP programming easier and more user-friendly.

5 A questionnaire about CLP applications

The information provided by the PACL conference proceedings and by the application papers at the CP conferences could provide, in our view, a reasonable snapshot of the current state-of-the-art in CLP-related applications. In fact, these are the main sources of information that we used to write our application survey. However, they could give us only a partial view of some important issues related to CLP applications domains and technologies. In particular, they could only partially give us the motivations underlying a certain piece of work or the choice of a certain technology, nor they could tell us about all the application needs that could not be addressed by the current technology. For this reason, and in order to build a more complete picture, we have prepared a questionnaire on CLP applications, where the questions were mainly related to how much, and which kind of, constraint technology was used, what were the major advantages and drawbacks, and what were the needs not yet addressed by the available tools.

The questionnaire was distributed among the participants of the PACL 1999 conference, and also among the members of the Compulog Net [54] mailing list. A total of 35 fully-filled questionnaires were returned, and here is a list of the main things we noticed in the answers.

Which application tools? First, we asked which tool was used to tackle applications. From this question, it turned out that there is no one winning tool, but instead several different languages are currently used to develop CLP applications: Ilog Solver, Chip, Sicstus Prolog, CLP(R), ECLiPSe, Ciao, Oz, IF-Prolog, clp(fd), and also proprietary systems. Among all these tools/languages, Sisctus Prolog and ECLiPSe are by far the most popular (13 and 11 preferences respectively), followed by ILOG Solver (6 preferences), CHIP (5) and IF-Prolog (4). The fact that some of these languages are almost free for the academia (like Sicstus Prolog and ECLiPSe) may of course influence the preferences of the users.

Which application problems? Then we asked which applications were tackled. Here, as predictable, the list is very long, but scheduling is by far the most studied: 22 preferences. Then we have resource allocation (12), graphical layout (6), and vehicle routing (3). All other applications had just one or two preferences; among them, we may notice the presence of some non-standard domains, like printed-circuit board design, 3D object recognition, consistency checks of rule sets, properties of logic programs, music, DB integration, protein topology, speculative projects, and web applications.

Advantages of CLP? The next question was about the advantages of current CLP technology. Expressivity was the most chosen (27 preferences), followed by flexibility (25 preferences) and fast prototyping (24 preferences). The other answers (easy to use and easy to learn) were not neglected (8 and 5 preferences respectively) but evidently have less impact on the CLP community.

Drawbacks of CLP? We also asked to list the drawbacks of current CLP technology, and we provided three answers: not easy to learn (18 preferences), not efficient enough (15 preferences), and difficult to adapt to specific problems (7 preferences). From such answers, we can see that even for people within the CLP/LP/Computational Logic community, current CLP tools are not easy to learn, which is in some sense surprising. However, it is true that to learn the basics of a CLP tool is very easy, but it is usually very hard to learn how to exploit its full power.

Needs not yet addressed by CLP? The question about the drawbacks leads to the next, and in our view, most interesting, question about the needs that arise from the application domains and that current CLP technology cannot fulfill. Here we had a lot of different answers, but the most chosen one was the need for tools dealing with over-constrained problems (14 preferences). It is in fact notorious that CLP experts dealing with user-specified constraint problems often report of over-constrained situations. The next most chosen answer was the need for tools dealing with user preferences (12 preferences). This need is of course related to over-constrained systems, but it also arises from real problems where preferences are an intrinsic feature (like timetabling). Then we have the need for modeling tools (10 preferences). In fact, it is only recently that this aspect has been considered, and modeling tools/languages have been developed (see Section 3). Many other needs were listed, like for debugging tools, for tools dealing with dynamic constraint problems, for the integration of different solvers, for explanation tools, and for tools that help to study the complexity of a problem and possibly discover its tractability.

Summarizing, what we learned from these questionnaires is that current CLP technology provides a lot of useful tools/languages, which offer high expressivity and flexibility, and allow a fast prototyping phase. Many applications are currently successfully tackled by such tools. However, if we want to make CLP a successful technology for most real-life problems we still need to improve its modeling, solving and explanation capabilities, especially in the context of over-constrained problems and problems with preferences.

6 Future opportunities for CLP applications

As shown in Section 3 and also by the answers to the questionnaire, many and diverse application problems are currently being tackled with great success by the CLP technology. However, there are still some problem features that have not been exploited by using constraints, and also some new application domains which only very recently CLP is starting to explore.

6.1 In current domains

There are so many applications domains that are currently successfully tackled by using CLP, that it is difficult to say what has to be done better or more in such domains. However, we would like to point out two aspects of current CLP technology that are in our view still underestimated in such domains, and not as used as they should be. However, we understand that such aspects also involve many research issues, still to be studied, and thus we will discuss them also later in the section devoted to the research challenges for CLP (Section 7).

More use of the modeling languages/tools. As noted above, one of the main needs for many applications is to have some help in the modeling phase, when the user description of the problem has to be transformed into a specification that the CLP language at hand can handle and solve. Some modeling tools and languages have recently been developed (see Section 3), and they should definitely be used to achieve a good model of the real life problem. This is especially true for problems with preferences or soft constraints, since otherwise these intrinsic features could be lost.

More use of soft constraint technology. Soft constraints have recently been developed to generalize classical hard constraints and make them more general. Some applications have already been developed to solve problems usually tackled by classical constraints, like time-tabling and workforce management, by using some tool to model and handle soft constraints. In this way, user preferences and other soft features could be considered in the problem formulation. However, it seems that most of the problems currently tackled by the CLP technology are still expressed and solved by using classical hard constraints. This may make the modeling process easier at first, but it may also generate apparent over-constrained problems or also an unnecessary level of user's involvement in the solving process. We believe that by using the current technology on soft constraints (although it is far from being completely satisfactory, see later), most of these problems could be represented more faithfully and then solved more automatically.

6.2 In new domains

Although constraints have been around for more than 30 years, it is only recently that they have reached the state of a full-fledged research area, as illustrated by the existence of international conferences and journals devoted to this domain. Moreover, the relatively recent developments of CLP languages and systems has allowed for a wide applicability of this technology, as shown by all the applications listed in Section 3.

However, it is also a widespread feeling that CLP techniques are vastly under-exploited outside of the CLP community. The reason for this is that constraints are often misunderstood outside this community (or also geographically, outside Europe). For historical reasons, CLP is seen as an extension of logic programming, and the relative decline of this branch of computer science, combined with the failure of the Japanese 5th generation project, do not contribute to promoting the field. Culturally, constraints are often considered a mathematical approach for classical OR problems, thereby inspiring suspicion. Technically, constraints are critiqued as being a purely engineering branch of computer science, which therefore should only be considered as a mere reformulation of standard optimization techniques. Of course there is much more to constraints, but it is true that CLP lies in an undefined zone between theoretical research and engineering, which makes it difficult to clearly define what constraints are. For this reason, it is important that constraints are started to be used for new application domains, so that people outside the CLP community can see their real advantages.

Multimedia. We believe that multimedia is one of such domains, for several reasons [133]. First, the availability of large catalogues of multimedia information to users via networks creates a huge demand for high-level user services. Second, the possibility, with digital representation of audio-visual signals, to incorporate the so-called "content information" together with the data (see the Mpeg-4 standard for audio, or the upcoming Mpeg-7 standard for audio-video), makes it possible to use symbolic techniques such as constraints to specify and solve problems related to audio-video manipulation and access. These issues may create new instances of classical problem domains, like scheduling for digital audio broadcasting, or also completely new classes of problems, like those involved in providing a music listener with a control system which has both interactivity and flexibility.

Virtual reality. Another new domain where constraints could be useful is the area of virtual reality [36]. In this domain, many interactions (or forbidden interactions) between objects can be seen as constraints, and implemented efficiently as such. This generalizes in an obvious way 2D geometrical constraints. The VRML language [45] is currently quite successful for describing 3D virtual worlds. Interaction between objects is performed in VRML by propagating events through routes, linking the output event of an object to the input event of another object, or by using the predefined "interpolators". However, nothing prevents to use some Java subprogram instead of these interpolators. Such Java programs can indeed represent constraints, in the form of generic propagation rules, and therefore constraints can be used to describe the relations between virtual objects, bringing much richer relations and interactions in the virtual world.

Network-wide programming. A third new domain where CLP could have a significant role is network-wide programming. This type of programming is bound to be of growing importance due to the popularity of Internet and the WWW protocols. CLP systems appear to have a number of features which appear to place them well to make an impact in this area, like dynamic memory management and compilation to architecture independent byte-code. Moreover, the notion of constraint entailment which comes from the CC framework offers a rich notion of inter-process communication and synchronization. Also, it seems natural that the more sophisticated network applications, such as intelligent information retrieval or distributed decision making, will require complex symbolic and numeric capabilities, at which CLP has already been shown particularly successful.

7 Research Challenges

According to the results of our questionnaire, there are many real needs that currently the CLP technology cannot address fully. For some of them there is already some ongoing effort, and thus we may predict the birth of practical tools in the next few years. Nevertheless, for most of the research challenges we list below, a complete answer should involve so many aspects that we may predict a longer term research effort, and a satisfactory answer, both for theoreticians and application people, only in several years.

Again, we need to remark that the list below cannot be complete, mainly because of our lack of an adequate level of understanding of the numerous research areas within CLP. However, some of the gaps of this list can be filled by looking at a special issue of the CONSTRAINTS journal devoted to "Strategic Directions in Constraint Programming" [88].

Modeling. It seems that the most urgent of all needs, but also the longest to be achieved in our view, is the possibility to have user-friendly tools that allow to model constraint problems in a faithful way. Such tools should be very high-level, because the user should concentrate on the main issues and not on the details, and should also provide a reasonable level of interactivity, because a black box cannot be convincing enough for the user. This raises issues of knowledge acquisition, effective representation, and possibly also problem reformulation and abstraction. Current CLP research should move into this direction, maybe also by using foreign technologies (negotiation and learning are already being used) to provide enough help during the modeling phase. This studies are most important as we think that different models may lead to very different performances. This research challenge is obviously a long-term goal, since it may take a long time to develop new modeling techniques and to combine them together to achieve an integrated modeling environment.

Soft constraints. Preferences (or similar concepts) arise in many real problems. Although there are already a number of constraint-based approaches to formalize such features, we still need to work to develop efficient and general methods to solve soft constraints and to program with them. Also the modeling issue described above assumes in this context an additional complexity. In fact, modeling a soft constraint problem is in general more difficult for the user, since he/she must be able to clearly state all the soft features of the real problem at hand (preferences, uncertainties, fuzziness, etc.). Also, solving a soft constraint problem means not just finding one solution, as in the classical hard constraint case, but finding a good solution, since the soft features introduce optimization criteria (often more than one).

Interactive/explanation tools. This need for interactivity applies not only to the modeling phase, but also to the solving phase, especially when the initial problem is over-constrained. In fact, the user need to know why no solution exists, and has to be involved in the process of modifying the constraint set in order to find a solution. Therefore, good interactive solving tools, as well as explanation tools, should be provided. Some interactive systems already exist (like most of the constraint-based spreadsheets listed above), but we are still far from having a general environment which cooperates with the user to solve a constraint problem.

Integration of CLP with other techniques. Another important topic for future research is the integration of constraint programming with other programming or solving techniques, like integer programming. The integration can take various forms. For example, the introduction of constraints within a branch-and-bound scheme, or the redundant modeling of a problem by using both methods and making the two models communicate.

The integration need is felt also at a higher level: the satisfaction approach (to find a solution of a constraint problem) and the perturbation approach (given a solution, find another one) need to be re-united. Most of the work in the CLP area has so far focussed on the satisfaction approach, which assumes a static view of constraint problems, but in a lot of real problems we need to find a new solution after a perturbation of an old one. Some work has been done to integrate these two viewpoints, but we are still far from having a system where both approaches coexist.

Debugging/visualization tools. One of the most common problems stated by users of CLP systems is their sometimes unpredictable behavior. Even small changes in a program or in the data can lead to a dramatic change in the performance. To help in this respect, new and more advanced debugging and visualization tools should be provided, in order to foster the development of programs which are more stable over a variety of data. This is maybe the most short-term of the research challenges listed here, since there is currently an Esprit project devoted to this research area. Thus we have already seen some interesting results in this direction, and we may expect to see other results soon.

Search control tools. Related to the above point, is the need of search control tools, that would allow a better understanding of what happens during the search for a solution. For many problems, the choice of a search routine can be crucial to the performance, but currently it is generally developed on an *ad hoc* basis. The possibility of studying the effect of different search methods, either in alternative to the usual depth-first chronological backtracking, or in combination with it, could provide dramatic improvements in solution quality and solving efficiency for many hard problems.

Global constraints. It is predictable that requirements for particular applications will lead to the creation of new global constraints. This will require new research efforts to develop efficient constraint propagation techniques for such new constraints.

Implementation of CLP systems. The performance of current CLP systems often compares favorably to other constraint solving and programming tools. However, their performance is still lower in many cases than that of traditional imperative languages. To fill this gap, advanced compilation and global analysis tools should be developed, as well as automatic parallelization techniques [90].

Network-wide programming. As noted in the previous section, CLP tools could be of some help in future systems for network-wide programming. In fact, CLP systems seems to have some features which are appropriate for using them in this area, and also some notions which are new to the area (like constraint entailment), which could make current network applications more sophisticated. However, it is easy to imagine that this is not just an application issue, since also research issues are touched when trying to use current CLP technology in this area. A few examples of such issues are the introduction of the notions of mobility and distribution in CLP, as well as the communication among solvers on different sites, both for distributed consistency checking and also for knowledge extraction from different sources.

8 Conclusions

We hope to have shown in this paper that the CP technology and research is by now mature enough to tackle a number of hard problems. Most application areas by now use CP-related results and algorithms to either solve their problems more efficiently, or in a more declarative and flexible way.

However, we have also pointed out several current needs that the users feels, and discussed the work which still needs to be done to address them. This work includes several areas and lines of research and development, and in our view it must concern mainly the following issues: the possibility of being helped during the modeling phase, a certain level of interactivity during the solving process, and the use of preferences in both over-constrained problems and also in intrinsically soft problems. We are highly confident that the CP community will be able to carry out this work and achieve several useful results that will allow both to satisfy the current needs, thus making the CP technology more appetible for a larger community, and also to investigate new ways of using the CP machinery within other research communities, in a way that that will improve them as well.

References

1. S. Abdennadher and H. Schlenker. Interdip - an interactive constraint based nurse scheduler. In *Proc. PACLP99*, 1999.
2. D. Achlioptas, L. M. Kirousis, E. Kranakis, D. Krizanc, M. S. O. Molloy, and Y. C. Stamatiou. Random constraint satisfaction: A more accurate picture. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97)*. Springer-Verlag, LNCS 1330, 1997.
3. K. Andersson and T. Hjerpe. Modeling constraint problems in CML. In *Proc. PACT98*, 1998.
4. H. Meyer auf'm Hofe. ConPlan/SIEDAplan: Personnel assignment as a problem of hierarchical constraint satisfaction. In *Proc. PACT97*, 1997.
5. A. Bachelu, P. Baptiste, C. Varnier, E. Boucher, and B. Legeard. Multi-criteria comparison between algorithic, constraint logic and specific constraint programming on a real scheduling problem. In *Proc. PACT97*, 1997.
6. R. Backofen. Constraint techniques for solving the protein structure prediction problem. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
7. P. Baptiste, C. Le Pape, and L. Peridy. Global constraints for partial CSPs: a case-study of resource and due date constraints. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
8. G. Baues, P. Kay, and P. Charlier. Constraint based resource allocation for airline crew management. In *Proc. ATTIS'94*, 1994.
9. M. Baumgart, H. P. Kunz, S. Meyer, and K. Quibeldey-Cirkel. Priority-driven constraints used for scheduling at universities. In *Proc. PACT97*, 1997.
10. N. Beldicenau, E. Bourreau, P. Chan, and D. Rivreau. Partial search strategy in CHIP. In *Proc. 2nd Int. Conf. on Meta-Heuristics, Sophia-Antipolis, France*, 1997.
11. N. Beldicenau and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modeling*, 20(12), 1994.
12. J. Bellone, A. Chamard, and C. Pradelles. Plane - an evolutive planning system for aircraft production. In *Proc. 1st Int. Conf. on Practical Applications of Prolog (PAP92)*, 1992.

13. A. Belussi, E. Bertino, and B. Catania. Manipulating spatial data in constraint databases. In *Proc. 5th Symp. on Spatial Databases.* Springer-Verlag, LNCS 1262, 1997.
14. F. Benhamou and A. Colmerauer. *Constraint Logic Programming: Selected Research.* MIT Press, 1993.
15. F. Benhamou and P. Van Hentenryck, editors. *Special issue on Interval Constraints.* CONSTRAINTS: An International Journal, Vol.2, N.2, Kluwer Academic Publishers, 1997.
16. F. Benhamou and W. Older. Applying interval arithmetic to real, integer and boolean constraints. *Journal of Logic Programming*, 1995.
17. R. Berger. Constraint-based gate allocation for airports. In *Proc. ILOG User-group meeting*, 1995.
18. S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95.* Morgan Kaufman, 1995.
19. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Logic Programming. In *Proc. IJCAI97.* Morgan Kaufman, 1997.
20. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, March 1997.
21. J. J. Blanco and L. Khatib. Course scheduling as a constraint satisfaction problem. In *Proc. PACT98*, 1998.
22. C. Bliek, B. Neveu, and G. Trombettoni. Using graph decomposition techniques for solving continuous CSPs. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98).* Springer-Verlag, LNCS 1520, 1998.
23. A. Borning and B. Freeman-Benson. Ultraviolet: a constraint satisfaction algorithm for interactive graphics. *CONSTRAINTS: An International Journal*, 3(1), 1998.
24. A. Borning and B. N. Freeman-Benson. The OTI constraint solver: a constraint library for constructing graphical user interfaces. In *Proceedings 1st Int. Conference on Principles and Practice of Constraint Programming (CP95).* Springer-Verlag, LNCS 976, 1995.
25. A. Borning, M. Maher, A. Martindale, and M. Wilson. Constraint hierarchies and logic programming. In Martelli M. Levi G., editor, *Proc. 6th International Conference on Logic Programming*, pages 149–164. MIT Press, 1989.
26. A. Brodsky. Constraint databases: Promising technology or just intellectual excercise? *CONSTRAINTS: An International Journal*, 2(1):35–44, 1997.
27. A. Brodsky, J. Jaffar, and M. Maher. Toward practical query evaluation for constraint databases. *CONSTRAINTS: An International Journal*, 2(3 and 4):279–304, 1997.
28. B. Cabon, S. de Givry, and G. Verfaille. Anytime lower bounds for constraint violation minimization problems. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98).* Springer-Verlag, LNCS 1520, 1998.
29. Mats Carlsson and Johan Widen. SICStus Prolog User's Manual. on-line version at `http://www.sics.se/sicstus/`. Technical report, Swedish Institute of Computer Science (SICS), 1999.
30. M. Carro and M. Hermenegildo. Some design issues in the visualization of constraint program execution. In *Proc. Joint Conference on Declarative Programming (AGP'98)*, 1998.
31. P. Chan, K. Heus, and G. Veil. Nurse scheduling with global constraints in CHIP: Gymnaste. In *Proc. PACT98*, 1998.
32. P. Cheesman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proc. IJCAI91*, 1991.

33. T. Chew and J.-M. David. A constraint-based spreadsheet for cooperative production planning. In *Proc. AAAI SIGMAN workshop on knowledge-based production planning, cheduling and control*, 1992.
34. K. P. Chow and M. Perrett. Airport counter allocation using constraint logic programming. In *Proc. PACT97*, 1997.
35. A. H. W. Chun, S. H. C. Chan, F. M. F. Tsang, and D. W. M. Yeung. Stand allocation with constraint technologies at Chek Lap Kok international airport. In *Proc. PACLP99*, 1999.
36. P. Codognet. The virtuality of constraints and the constraints of virtuality. *CONSTRAINTS: An International Journal*, 2(1):45–50, 1997.
37. P. Codognet and D. Diaz. Compiling constraints in `clp(fd)`. *Journal of Logic Programming*, 27(3), 1996.
38. D. Cohen, M. Gyssens, and P. Jeavons. Derivation of constraints and database relations. In *Proceedings 2nd Int. Conference on Principles and Practice of Constraint Programming (CP96)*. Springer-Verlag, LNCS 1118, 1996.
39. D. A. Cohen, P. Jeavons, and M. Koubarakis. Tractable disjunctive constraints. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97)*. Springer-Verlag, LNCS 1330, 1997.
40. J. Cohen. Constraint logic programming. *Communications of the ACM*, 33(7), 1990.
41. H. Collavizza, F. delobel, and M. Rueher. A note on partial consistency over continuous domains. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
42. C. Collignon. Gestion optimisee de ressources humaines pour l'audiovisuel. In *Proc. CHIP users' club*, 1996.
43. A. Colmerauer. Opening the Prolog-III universe. *BYTE magazine*, 12(9), 1987.
44. A. Colmerauer. An introduction to Prolog-III. *Communication of the ACM*, 1990.
45. The VRML consortium. *The virtual reality modeling language*. ISO/IEC DIS 14772-1, 1997, 1997. URL: `www.vrml.org/Specifications/VRML97/index.html`.
46. COSYTEC, editor. *CHIP C library*. COSYTEC SA, Parc Club Orsay Universite, 4, rue Jean Rostand, 91893 Orsay cedex, France, 1995.
47. K. Darby-Dowman, J. Little, G. Mitra, and M. Zaffalon. Constraint logic programming and integer programming approaches and their collaboration in solving an assignment scheduling problem. *CONSTRAINTS: An International Journal*, 1(3), 1997.
48. F. de Boer, M. Gabbrielli, and M. C. Meo. Semantics and expressive power of a timed concurrent constraint language. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97)*. Springer-Verlag, LNCS 1330, 1997.
49. S. de Givry, G. Verfaille, , and T. Schiex. Bounding The Optimum of Constraint Optimization Problems. In G. Smolka, editor, *Proc. CP97*, pages 405–419. Springer-Verlag, LNCS 1330, 1997.
50. R. Dechter and J. Pearl. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, 34(1), 1988.
51. R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38, 1989.
52. Rina Dechter. From local to global consistency. *Artificial Intelligence*, 55:87–107, 1992.
53. Y. Deville, M. Jansenn, and P. Van Hentenryck. Consistency techniques in ordinary differential equations. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.

54. DFKI. *The Compulog Net network of excellence.* `http://www.compulog.org`, 2000.
55. M. Dincbas and H. Simonis. Apache - a constraint based, automated stand allocation system. In *Proc. of Advanced Software technology in Air Transport (ASTAIR'91)*, 1991.
56. M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proc. International Conference on Fifth Generation Computer Systems.* Tokyo, Japan, 1988.
57. A. Dresse. A constraint programming library dedicated to timetabling. In *Proc. ILOG User-group meeting*, 1995.
58. D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. IEEE International Conference on Fuzzy Systems*, pages 1131–1136. IEEE, 1993.
59. A. Dubos and A. Du Jeu. Application EPPER planification des ageents roulaants. In *Proc. CHIP users' club*, 1996.
60. H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *Proc. 1st European Ccongress on Fuzzy and Intelligent Technologies (EUFIT)*, 1993.
61. J. Fernandez and E. Sanchez. A cashier assignment system. In *Proc. ILOG User-group meeting*, 1995.
62. T. Filkhorn, H.-A. Schneider, A. Scholz, A. Strasser, and P. Warkentin. SVE system verification environment. Technical report, Siemens AG, Technical report SVE, ZFE BT SE, 1995.
63. F. Focacci, E. Lamma, P. Mello, and M. Milano. Constraint logic programming for the crew rostering problem. In *Proc. PACT97*, 1997.
64. E. Freuder, editor. *Principles and Practice of Constraint Programming–CP96.* Springer-Verlag, LNCS 1118, 1996.
65. E. C. Freuder. Backtrack-free and backtrack-bounded search. In Kanal and Kumar, editors, *Search in Artificial Intelligence*, pages 343–369. Springer-Verlag, 1988.
66. E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *AI Journal*, 58, 1992.
67. E. C. Freuder and R. J. Wallace. Suggestion strategies for constraint-based matchmaker agents. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98).* Springer-Verlag, LNCS 1520, 1998.
68. M. Fromherz, V. Gupta, and V. Saraswat. Model-based computing: constructing constraint-based software for electro-mechanical systems. In *Proc. Conf. on Practical Applications of Constraint Technology (PACT95)*, 1995.
69. T. Fruhwirth and P. Brisset. Optimal planning of digital cordless telecommunication systems. In *Proc. PACT97*, 1997.
70. Thom Fruhwirth. Constraint simplification rules. In A. Podelski, editor, *Constraint Programming: Basics and Trends.* Springer-Verlag, LNCS 910, 1995.
71. M. Gavanelli, E. Lamma, P. Mello, and M. Milano. Extending CLP(FD) with interactive data acquisition for 3D visual object recognition. In *Proc. PACLP99*, 1999.
72. I. Gent, E. MacIntyre, P. Prosser, P. Shaw, and T. Walsh. The constrainedness of arc consistency. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97).* Springer-Verlag, LNCS 1330, 1997.
73. Y. Georget and P. Codognet. Compiling semiring-based constraints with clp(fd,s). In M.Maher and J-F. Puget, editors, *Proc. CP98.* Springer-Verlag, LNCS 1520, 1998.

74. Y. Georget, P. Codognet, and F. Rossi. Constraint retraction in clp(fd): Formal framework and performance results. *CONSTRAINTS: An International Journal*, 4(1), 1999.
75. C. Gervet, Y. Caseau, and D. Montaut. On refining ill-defined constraint problems: A case study in iterative prototyping. In *Proc. PACLP99*, 1999.
76. F. Glaisner and L.-M. Richard. FORWARD-C: A refinery scheduling system. In *Proc. PACT97*, 1997.
77. D. Q. Goldin and P. Kanellakis. Constraint query algebras. *CONSTRAINTS: An International Journal*, 1(1 and 2):45–84, 1996.
78. H.-J. Goltz and D. Matke. Combined interactive and automatic timetabling. In *Proc. PACLP99*, 1999.
79. G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. In *Proc. IJCAI99*, 1999.
80. M. Gyssens, P. G. Jeavons, and D. A. Cohen. Decomposing Constraint Satisfaction Problems Using Database Techniques. *Artificial Intelligence*, 66(1):57–89, 1994.
81. S. Haridi and S. Janson. Kernel Andorra Prolog and its computational model. In *Proc. ICLP90*. MIT Press, 1990.
82. S. Haridi, P. Van Roy, P. Brand, and C. Schulte. Programming languages for distributed applications. *New Generation Computing*, vol.16, n. 3:223–261, 1998.
83. W. Harvey and M. Ginsberg. Limited discrepancy search. In *Proc. IJCAI95*, 1995.
84. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.
85. P. Van Hentenryck. Helios: a modeling language for global optimization. In *Proceedings PACT'96*, pages 317–335, 1996.
86. P. Van Hentenryck. *Numerica: A Modeling Language for Global Optimization*. MIT Press, 1997.
87. P. Van Hentenryck. Visual solver: A modeling language for constraint programming. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97)*. Springer-Verlag, LNCS 1330, 1997.
88. P. Van Hentenryck and V. Saraswat, editors. *Special issue on Strategic Directions on Constraint Programming*. CONSTRAINTS: An International Journal, Vol.2, N.1, Kluwer Academic Publishers, 1997.
89. M. Henz, G. Smolka, and J. Wurtz. Oz – a programming language for multiagent systems. In *Proc. IJCAI93*. Morgan Kaufmann, 1993.
90. M. Hermenegildo. Some challenges for constraint programming. *CONSTRAINTS: An International Journal*, 2(1):63–70, 1997.
91. M. Hermenegildo and the CLIP group. Some methodological issues in the design of CIAO – a generic, parallel concurrent constraint system. In *Principles and Practice of Constraint Programming*. Springer-Verlag, LNCS 874, 1994.
92. L. Hermosilla and G. Kuper. Towards the definition of a spatial object-oriented data model with constraints. In *Proc. 1st Int. CONTESSA workshop on constraint databases and their applications*. Springer-Verlag, LNCS 1034, 1996.
93. `http://www.mozart-oz.org/`. *The Mozart project*, 1999.
94. E. Hyvonen. Interval constraints spreadsheets for financial planning. In *Proc. 1st Int. Conf. on Artificial Intelligence Applications on Wall Street*. IEEE Computer Society Press, 1991.
95. IC-PARC. *The ECLiPSe Library Manual Release 4.2*. `http://www.icparc.ic.ac.uk/eclipse/`, 1999.

96. ILOG, editor. *ILOG SOLVER: Object oriented constraint programming*. ILOG SA, 12, Av. Raspail, BP 7, 94251 Gentilly cedex, France, 1995.
97. G. Istrate. Counting, structure identification and maximum consistency for binary constraint satisfaction problems. In *Proceedings 3rd Int. Conference on Principles and Practice of Constraint Programming*. Springer-Verlag, LNCS 1330, 1997.
98. J. Jaffar, editor. *Principles and Practice of Constraint Programming–CP99*. Springer-Verlag, LNCS 1713, 1999.
99. J. Jaffar and al. The CLP(R) Language and System. *ACM Transactions on Programming Languages and Systems*, 1992.
100. J. Jaffar and M.J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19 and 20, 1994.
101. Michael Jampel, editor. *Over-Constrained Systems*. Springer-Verlag, LNCS 1106, 1996.
102. S. Janson. *AKL - A Multiparadigm Programming Language*. PhD thesis, Uppsala Theses in Computer Science 19, ISSN 0283-359X, ISBN 91-506-1046-5, Uppsala University, and SICS Dissertation Series 14, ISSN 1101-1335, ISRN SICS/D-14–SE, 1994.
103. P. Jeavons. Constructing constraints. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
104. P. G. Jeavons and D. A. Cohen. An algebraic characterization of tractable constraints. In *Proc. of Computing and Combinatorics (COCOON'95)*. Springer-Verlag, LNCS 959, 1995.
105. P. G. Jeavons, D. A. Cohen, and M. Gyssens. A unifying framework for tractable constraints. In *Proc. 1st Int. Conf. on Principles and Practice of Constraint Programming (CP95)*. Springer-Verlag, LNCS 976, 1995.
106. P. G. Jeavons, D. A. Cohen, and M. Gyssens. A test for tractability. In *Proc. 2nd Int. Conf. on Principles and Practice of Constraint Programming (CP96)*. Springer-Verlag, LNCS 1118, 1996.
107. P. G. Jeavons, D. A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of ACM*, 44, 1997.
108. P. G. Jeavons, D. A. Cohen, and M. Gyssens. How to determine the expressive power of constraints. *CONSTRAINTS: An international Journal*, 4(2), 1999.
109. Yao Jianchao. Book review of [124]. *Intelligence: New visions of AI in practice*, 10(2), 1999.
110. R. W. L. Kam and J. H. M. Lee. Fuzzifying the constraint hierarchies framework. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
111. P. Kanellakis and D. Q. Goldin. Constraint programming and database query languages. In *Proc. Int. Symp. on Theoretical Aspects of Computer Software*. Springer-Verlag, LNCS 789, 1994.
112. P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):25–52, 1995.
113. F. Kokkoras and S. Gregory. D-WMS: Distributed workforce management using CLP. In *Proc. PACT98*, 1998.
114. V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, 13(1):32–44, 1992.
115. Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97)*. Springer-Verlag, LNCS 1330, 1997.

116. J. Little, editor. *Proc. 1st Int. Conf. on the Practical Applications of Constraint Technologies and Logic Programming–PACLP99*. The Practical Application Company, 1999.
117. J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1993.
118. E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. Random constraint satisfaction: Theory meets practice. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
119. A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1984.
120. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
121. A.K. Mackworth. Constraint satisfaction. In Stuart C. Shapiro, editor, *Encyclopedia of AI (second edition)*, volume 1, pages 285–293. John Wiley & Sons, 1992.
122. M. Maher and J.-F. Puget, editors. *Principles and Practice of Constraint Programming–CP98*. Springer-Verlag, LNCS 1520, 1998.
123. K. Marriott, S. S. Chok, and A. Finlay. A tableau based constraint solving toolkit for interactive graphical applications. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
124. K. Marriott and P. J. Stuckey. *Programming with constraints: an introduction*. MIT Press, 1998.
125. W. Maximuck and M. Reviakin. A prototype of train schedule module. In *Proc. PACT98*, 1998.
126. K. McAloon. Cooperating solvers: A progress report. In *Proc. PACLP99*, 1999.
127. K. McAloon and C. Tretkoff. 2LP: Linear programming and logic programming. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming*. MIT Press, 1995.
128. S. Melzer. Verification of parallel systems using constraint programming. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97)*. Springer-Verlag, LNCS 1330, 1997.
129. U. Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, 7:95–132, 1974. Also Technical Report, Carnegie Mellon University, 1971.
130. U. Montanari and F. Rossi, editors. *Principles and Practice of Constraint Programming–CP95*. Springer-Verlag, LNCS 976, 1995.
131. J. Montelius and K.A.M. Ali. An and/or-parallel implementation of akl. *New Generation Computing*, 14:31–52, 1996.
132. G. A. Narboni, editor. *Proc. 4th Int. Conf. on the Practical Applications of Constraint Technology–PACT98*. The Practical Application Company, 1998.
133. F. Pachet. Constraints and multimedia. In *Proc. PACLP99*, 1999.
134. C. Le Pape and P. Baptiste. A constraint programming library for preemptive and non-preemptive scheduling. In *Proc. PACT97*, 1997.
135. C. Le Pape and P. Baptiste. Resource constraints for preemptive job-shop scheduling. *CONSTRAINTS: An International Journal*, 3(4), 1998.
136. R. Pareschi. Work force management in telecommunications and beyond telecommunications. In *Proc. PACLP99*, 1999.
137. M. Perrett. Using constraint logic programming techniques in container port planning. *ICL Technical Journal*, pages 537–545, 1991.

138. G. Pesant, M. Gendreau, and J.-M. Rousseau. GENIUS-CP: a generic single-vehicle routing algorithm. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97)*. Springer-Verlag, LNCS 1330, 1997.
139. A. Podelski, editor. *Constraint Programming: Basics and Trends*. Springer Verlag, LNCS 910, 1995.
140. P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81, 1996.
141. P. Prosser. The dynamics of dynamic variable ordering heuristics. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
142. L. Purvis and P. Jeavons. Constraint tractability theory and its application to the product development process for a constraint-based scheduler. In *Proc. PACLP99*, 1999.
143. J.-C. Regin. The symmetric alldiff constraint. In *Proc. IJCAI99*, 1999.
144. J.-C. Regin and J.-F. Puget. A filtering algorithm for globaal sequencing constraints. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97)*. Springer-Verlag, LNCS 1330, 1997.
145. M. Renschler. Configuration spreadsheet for interactive constraint problem ssolving. In *Proc. PACT98*, 1998.
146. P. Z. Revesz. Safe stratified Datalog with integer order programs. In *Proceedings 1st Int. Conference on Principles and Practice of Constraint Programming (CP95)*. Springer-Verlag, LNCS 976, 1995.
147. P. Z. Revesz. Safe Datalog queries with linear constraints. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
148. R. Rodosek and M. Wallace. A generic model and hybrid algorithm for hoist scheduling probleems. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
149. F. Rossi and U. Montanari. Exact solution of networks of constraints using perfect relaxation. In *Proc. International Conference on Knowledge Representation (KR89)*. Morgan Kaufmann, 1989.
150. F. Rossi and A. Sperduti. Learning solution preferences in constraint problems. *Journal of Experimental and Theoretical Computer Science*, 1998. Vol 10.
151. Zs. Ruttkay. Fuzzy constraint satisfaction. In *Proc. 3rd IEEE International Conference on Fuzzy Systems*, pages 1263–1268, 1994.
152. D. Sam-Haroud and B. Faltings. Consistency techniques for continuous constraints. *CONSTRAINTS: An International Journal*, 1, 1996.
153. V. Saraswat and P. Van Hentenryck, editors. *principles and Practice of Constraint Programming*. MIT Press, 1995.
154. V.A. Saraswat. *Concurrent Constraint Programming*. MIT Press, 1993.
155. T. Schiex. Possibilistic constraint satisfaction problems, or "how to handle soft constraints?". In *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.
156. T. Schiex, H. Fargier, and G. Verfaille. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. IJCAI95*, pages 631–637. Morgan Kaufmann, 1995.
157. C. Schulte. Oz explorer: a visual constraint programming tool. In *Proceedings of the 14th Int. Conf. on Logic programming*, pages 286–300. MIT Press, 1997.
158. C. Schulte. Programming constraint inference engines. In *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97)*. Springer-Verlag, LNCS 1330, 1997.

159. K. Sergiou and T. Walsh. The difference all-difference makes. In *Proc. IJCAI99*, 1999.
160. B. Seybold, F. Metzger, G. Odan, J. Bathelt, F. Collenberg, J. Taiber, K. Simon, and M. Engeli. Spatial modelling with geometric constraints. In *Proc. PACT97*, 1997.
161. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
162. I. Shvetsov, V. Kornienko, and S. Preis. Interval spreadsheet for problems of financial planning. In *Proc. PACT97*, 1997.
163. H. Simonis. Visual CHIP - a visual language for defining constraint programs. In *CCL II workshop*, 1997.
164. H. Simonis and A. Aggoun. Search tree debugging. Technical report, COSYTEC SA, 1997.
165. H. Simonis and P. Charlier. Cobra - a system for train crew scheduling. In *Proc. DIMACS workshop on constraint programming and large scale combinatorial optimization*, 1998.
166. H. Simonis and T. Cornelissens. Modelling producer/consumer constraints. In *Proceedings 1st Int. Conference on Principles and Practice of Constraint Programming (CP95)*. Springer-Verlag, LNCS 976, 1995.
167. G. Smedback, M. Carro, and M. Hermenegildo. Interfacing Prolog and VRML and its applications to constraint visualization. In *Proc. PACLP99*, 1999.
168. B. Smith and M. E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81, 1996.
169. G. Smolka, editor. *Principles and Practice of Constraint Programming–CP97*. Springer-Verlag, LNCS 1330, 1997.
170. Gert Smolka. A foundation of higher-order concurrent constraint programming. In *Proc. 1st CCL*. Springer-Verlag, LNCS 845, 1994.
171. S. Takahashi, S. Matsuoka, K. Miyashita, and H. Hosobe. A constraint-based approach for visualization and animation. *CONSTRAINTS: An International Journal*, 3(1), 1998.
172. R. Tamassia. Constraints in graph drawing algorithms. *CONSTRAINTS: An International Journal*, 3(1), 1998.
173. D. Toman. Memoing evaluation for constraint extensions of Datalog. *CONSTRAINTS: An International Journal*, 2(3 and 4):337–360, 1997.
174. E. Tsang, P. Mills, R. Williams, J. Ford, and J. Borrett. A computer aided constraint programming system. In *Proc. PACLP99*, 1999.
175. Edward P. K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
176. M. Wallace, editor. *Proc. 1st Int. Conf. on the Practical Applications of Constraint Technology–PACT95*. The Practical Application Company, 1995.
177. M. Wallace. Practical applications of constraint programming. *Constraints: An International Journal*, 1, 1996.
178. M. Wallace, editor. *Proc. 2nd Int. Conf. on the Practical Applications of Constraint Technology–PACT96*. The Practical Application Company, 1996.
179. M. Wallace, editor. *Proc. 3rd Int. Conf. on the Practical Applications of Constraint Technology–PACT97*. The Practical Application Company, 1997.
180. M. Wallace, S. Novello, and J. Schimpf. Eclipse: A platform for constraint logic programming. Technical report, IC-Parc, Imperial College, London, 1997.

181. R. J. Wallace and E. C. Freuder. Stable solutions for dynamic constraint satisfaction problems. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98)*. Springer-Verlag, LNCS 1520, 1998.
182. D. L. Waltz. Understanding line drawings of scenes with shadows. In Winston, editor, *The Psycology of Computer Vision*. McGraw-Hill, 1975.
183. T. M. Yakhno, V. Z. Zilberfaine, and E. S. Petrov. Applications of ECLiPSe: Interval domain library. In *Proc. PACT97*, 1997.
184. M. Yokoo. Asynchronous weak-committment search for solving distributed constraint satisfaction problems. In *Proc. CP95 (Principles and Practice of Constraint Programming)*. Springer-Verlag, LNCS 976, 1995.