Fifth Congress of Logic applied to Technology (LAPTEC'05)
Isao Shirakawa
IOS Press, Ohmsha, 2003

Constraint programming for software engineering

Narendra Jussien
École des Mines de Nantes – LINA CNRS FRE 2729
4 rue Alfred Kastler – BP 20722
F-44307 Nantes Cedex 3

Abstract. Constraint programming is a research topic at the crossroads of artificial intelligence, operations research and numerical analysis. This rather new research field as now proved useful for solving complex combinatorial problems in decision making processes. New applications arise now and particularly in software engineering. In this paper, we show the interest and the perspectives in using constraint programming in this technological field through several examples (including identification and correction of degraded design patterns in object oriented software, implementation of an application in an anytime context, design of hard real time systems).

Keywords. artificial intelligence, constraint programming, software engineering, applications

1. Introduction

Constraint programming is a research topic at the crossroads of artificial intelligence, operations research and numerical analysis. This rather new research field as now proved useful for solving complex combinatorial problems in decision making processes. New applications arise now thanks to new extensions to constraint programming. In this paper, we show the interest and the perspectives in using constraint programming in software engineering through three examples.

2. Constraint programming

Constraint programming is a research topic at the crossroads of several areas: discrete mathematics, numerical analysis, artificial intelligence, operations research, formal calculus, etc. The *constraint programming* community can be proud of several success stories in various domains: combinatorial problems, scheduling, financial analysis, simulation and synthesis of electronic circuits, failure diagnosis, decision support systems, molecular biology, geometrical problem solving, etc.

Constraint programming is nowadays spreading into the industry thanks to two main societies in the field, Ilog (`www.ilog.com`) and Cosytec (`www.cosytec.com`) which have developed, for example, softwares for air traffic optimisation, timetabling and restoring problems, etc. Several public domain constraints solvers are also available (such as GNU prolog – `gprolog.inria.fr` and `choco` – `choco.sf.net`) making the technology free of charge for testing purpose.

2.1. A success story

Those good results can be explained by the following features of constraint programming:

- one of the basic assumption of constraint programming, stating that search can be performed in a bounded domain, is very often naturally verified in practice where unknown values correspond to physical values;
- tools and techniques are very general, they are never restricted to particular class of problems;
- in essence, constraint programming techniques are well fitted for distributed implementations, leading to improved performance;
- developed techniques were designed to complement other techniques more prone to the number of unknowns or to the structure of the set of equations defining the problem (as in algebraic systems).

2.2. The general framework

A constraint satisfaction problem (CSP [Tsa93]) is modelled by a triplet (V, D, C) where V is a set of variables taking their value in a specified domain for each variable (the set of those domains is D) and upon which constraints (in set C) are defined. A solution for a CSP is a complete assignment of the variable (each variable is assigned a value from its domain) such that all the constraints in C simultaneously hold.

Solving a CSP involves the interleaving of filtering steps (efficient removal of values from the domains of the variable in order to prune portions of the search space that cannot possibly contain a solution of the problem) and enumeration steps (taking decision – often assigning a value to a variable – whenever the filtering step is not sufficient to design a solution). A constraint solver is a programmable system controlling this interleaving through the call to filtering components (the constraints themselves) and enumeration control components. The core of the constraint programming research field is devoted to those two crucial steps in order to provide as generic as possible constraint solvers.

3. Extending constraint programming

For a long time, constraint programming focused solely on highly combinatorial problems for decision support systems. Recently, the field opened up to new horizons. Extensions to classical constraint programming do exist to take into account new problems, new applications, etc.

3.1. Explanations and constraint program analysis

Introducing the concept of *explanation* within constraint programming is quite recent [Jus01]. An explanation, originating from works on ATMS [Kle86] is actually an efficient way to record a limited and readable trace of the solver's activity. Integrating explanations within constraint programming paves the way to several new applications:

- unifying a vast majority of existing techniques for solving CSP in a generic framework that allows an easy characterization of those methods in order to identify the appropriate technique for a given instance of a problem;
- developing new algorithms able to dynamically handle problem alteration before, during and after solving the instance while keeping a given stability to the set of successive solutions (being able to use that feature in a user interacted system);
- informing the user (or developer) in a understandable and readable manner about the inner activity of the constraint solver.

This extension defines a new paradigm of constraint programming.

3.2. Preferences for constraint programming

Valued CSP (VCSP [BMR$^+$99]) represent an extension of CSP design to handle over-constrained problems or problems for which some solutions are preferred from others. A VCSP can be modelled as a CSP (V, D, C) to which is added a valuation structure S and an evaluation function ϕ. The valuation structure is a triplet $(E, >, \times)$ where E is a set of valuations, $>$ a total ordering of E ($\top$ is the maximal element and $\bot$ is the minimal element) and $\times$ is a binary operator closed on E. The valuation function ϕ defined from C to E associates a valuation to each constraints. It represents its relative importance.

Let A be a complete assignment of the variables and $C_{unsat}(A)$ the set of constraints that do not hold in A. The valuation of A is the aggregation of all valuations of the constraints in $C_{unsat}(A)$: $\phi(A) = \times\phi(c)$ for all $c \in C_{unsat}(A)$.

A solution to a VCSP is complete assignment of variables with a minimal valuation. Specific techniques are used based upon the nature of the selected aggregation operator: $\wedge$ for classical CSP, $\max$ for possibilistic CSP, $\cup$ for multi-sets in lexicographic CSP, $\sum$ for additive CSP, etc. The latter is the more often studied extension. It is considered as the hardest when looking for algorithms.

3.3. Hybrid techniques

The extension of constraint programming to hybrid algorithms tris to combine both tree-based search techniques dans local search techniques. Tree-based techniques are mainly used to produce optimal solutions and to prove the optimality. But, because of their potentially exponential behavior, they can prove heavily costly in computation effort. Moreover, such techniques often cannot quickly provide good solutions. Conversely, local search techniques (such as simulated annealing or tabu search) are able to provide quite quickly quite good solutions thanks to their opportunistic way of exploring the search space. But, those techniques may be trapped in local optima or spend many time in an uninteresting neighborhood.

Hybrid approaches [JL04], are a compromise between the two approaches. They efficiently combine advantages from constraint propagation provided by tree-based search techniques and opportunistic wandering of local search techniques. Such hybrid techniques often combining two or more different approaches in order to solve a given have very good results and allowed the consideration of much broader real-life problems.

4. A first example: solving an anytime problem

It has now been proved that constraint programming is a valuable tool to handle industrial off-line problems (specified problem and unlimited solving time). However, in a the context of on-line problems, problems are prone to unexpected modifications due to unscheduled external modifications of the environment and their solving is subject to strong temporal constraints.

Even though classical constraint programming does not seem appropriate to handle this kind of problems, using some of its extensions may be of interest. Indeed, using extensions of constraint programming leads to the definition of an algorithm which may be interrupted at all times and nevertheless providing the following features:

- quick computation of good solutions;
- gradual improvements of those first solutions taking into account a time contract.

This is considered as an *anytime* context. The objective is to compute a good performance profile for the interruptible algorithm.

In this contexte, using a VCSP (in order to model the notion of quality of a partial solution) and an hybrid algorithm seems quite appropriate. For example, the VNS/LDS+CP algorithm [LB01] is a recent hybrid algorithm that is based on a local search algorithm which uses variable neighborhoods whose exploration is performed using an efficient LDS-like algorithm [HG95] (a partial but well spread tree-based search) helped with propagation in order to solve VCSP. Its use on real instances of radio frequency assignment problems [CdGL$^+$99] show that such an hybridation is a valuable answer for solving problems in an anytime context.

5. A second example: the PTIDEJ system

Explanation-based constraint programming has been used to to design the PTIDEJ system (Pattern Trace Identification, Detection and Enhancement in Java [AACGJ01]). PTIDEJ is an automated system designed to identify micro-architectures looking like *design patterns* in object oriented source code. A micro-architecture defines a subset of classes in an objected oriented program.

Producing quality source code is an important objective for the software industry. A quality source code eases evolution and maintenance: adding new functionalities, correcting bugs, adapting to new execution environments, integrating into classes libraries, etc. In object oriented programming, quality source code can be identified in two dimensions: efficient, clearly written, documented, conventions and idiom-compliant algorithms; and, an *elegant* class architecture. Design patterns [GHJV94] are examples of elegant micro-architecture that embed the experience of skillful developers.

Nevertheless, producing at once such elegant and well written code is not a obvious task. But, design patterns are often present in existing source code in an approximate state (*i.e.* similar but not exactly to those proposed in design patterns). No tool exist that is able to identify in existing source code those approximate micro-architecture in order to inform the developer of possible improvement of her code (making them compliant to the aforementioned design patterns).

PTIDEJ is such a tool. Its objective is to identify micro-architecture in existing source code. The key idea of PTIDEJ is to define a constraint satisfaction problem whose structure (variables dans constraints) represents the design pattern one is looking for and whose semantics (the domaine of the variables and the actual semantics of the constraints) is deduced from the provided code. Solving such a problem leads to the computation of micro-architectures exactly matching the desired design pattern. In order to look for approaching micro-architectures, an explanation-based constraint solver is used to identify why the exact micro-architecture cannot be found and therefore informing the user of constraints that cannot be met in the pattern.

The main interest of PTIDEJ, not withstanding it is the first tool able to identify approached micro-architecture [GJ01], is that it is able to provide explanations for its answers. This is really interesting since coding and software engineering is often considered a form of art and where fully automated systems are not always appreciated by potential users: programmers.

6. A third example: solving an hard real time allocation problem

Real-time systems are at the heart of embedded systems and have applications in many industrial areas: telecommunication, automotive, aircraft and robotics systems, etc. Today, applications (*e.g.* cars) involve many processors to serve different demands (cruise control, ABS, engine management, etc.). These systems are made of specialized and distributed processors (interconnected through a network) which receive data from sensors, process appropriate answers and send it to actuators. Their main characteristics lie in functional as well as non-functional requirements like physical distribution of the resources and timing constraints. Timing constraints are usually specified as deadlines for tasks which have to be executed. Serious damage can occur if deadlines are not met. In this case, the system is called a *hard* real-time system and timing predictability is required. In this field, some related works are based on off-line analysis techniques that compute the response time of the constrained tasks. Such techniques have been initiated by Liu and al. [Liu73] and consist in computing the worst-case scenario of execution. Extensions have been introduced later to take into account shared resources, distributed systems [TC94] or precedence constraints [GKL91].

Our problem consists in assigning periodic and preemptive tasks with fixed priorities (a task is periodically activated and can be preempted by a higher priority task) to distributed processors. A solution is an allocation of the tasks on the processors which meets the schedulability requirements. The problem of assigning a set of hard preemptive real-time tasks in a distributed system is NP-Hard [Law83]. It has been tackled with heuristic methods [FCO99,Ram90], simulated annealing [TBW92,BM94] and genetic algorithms [FCO99,San96]. However, these techniques are often incomplete and can fail in finding any feasible assignment even after a large computation time. New practical approaches are still needed.

We introduced in [CHDJT04] a decomposition based method which separates the allocation problem itself from the scheduling one. It is related to the Benders decomposition and especially to the logic Benders based decomposition. On the one hand, constraint programming offers competitive tools to solve the assignment problem, on the other hand, real time scheduling techniques are able to achieve an accurate analysis of the schedulability. Our method uses Benders decomposition as a way of generating precise nogoods in constraint programming leading to a new powerful hybrid technique. It implements a *logical* duality to infer nogoods, tries to enforce the constraint model and finally performs an incremental resolution of the master problem. It is also strongly related to a class of algorithms which intends to learn from mistakes in a systematic way by managing nogoods.

Our approach shows the extreme interest and efficiency of hybrid techniques for solving problems arising in real life contexts.

7. Conclusion

In this paper, we introduced constraint programming and three of its extensions. Constraint programming is not limited to combinatorial optimisation in decision making processes. The three applications of its extensions described here clearly show constraint programming could be of great use in software engineering.

References

[AACGJ01] Hervé Albin-Amiot, Pierre Cointe, Yann-Gaël Guéhéneuc, and Narendra Jussien. Instantiating and detecting design patterns: Putting bits and pieces together. In *16th IEEE conference on Automated Software Engineering (ASE'01)*, pages 166–173, San Diego, USA, November 2001. IEEE Computer Society Press.

[BMR$^+$99] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie and H. Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4(3):199–240, 1999.

[BM94] G. Borriello and D. Miles. Task Scheduling for Real-Time Multiprocessor Simulations. In *11th Workshop on RTOSS*, pages 70–73, 1994.

[CdGL$^+$99] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners.W.K. Radio link frequency assignment. *Constraints*, 4(1):79–89, 1999.

[CHDJT04] Hadrien Cambazard, Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Narendra Jussien, Yvon Trinquet Decomposition and learning for a real time task allocation problem. In *Principles and Practice of Constraint Programming* (CP 2004), Lecture Notes in Computer Science, vol. 3258, pp. 153-167, Springer-Verlag, 2004.

[FCO99] E. Ferro, R. Cayssials, and J. Orozco. Tuning the Cost Function in a Genetic/Heuristic Approach to the Hard Real-Time Multitask-Multiprocessor Assignment Problem. In *Proceedings of the Third World Multiconference on Systemics Cybernetics and Informatics*, pages 143–147, 1999.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[GJ01] Yann-Gaël Guéhéneuc and Narendra Jussien. Using explanations for design-patterns identification. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, pages 57–64, Seattle, WA, USA, August 2001.

[GKL91] M. González Harbour, M.H. Klein, and J.P. Lehoczky. Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority. In *Proceedings of the IEEE Real-Time Systelms Symposium*, pages 116–128, December 1991.

[HG95] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of IJCAI'95*, Montreal, August 1995.

[Jus01] Narendra Jussien. e-constraints: explanation-based constraint programming. In *CP01 Workshop on User-Interaction in Constraint Satisfaction*, Paphos, Cyprus, 1 December 2001.

[JL04] Narendra Jussien, François Laburthe. Hybrid Optimization Techniques. In *Annals of Operations Research*, vol. 130, Kluwer, Special Issue following CP-AI-OR'02, 2004

[Kle86] Johan de Kleer. An Assumption-based TMS. In *Artificial Intelligence*, vol. 28, pages 127-162, 1986

[Law83] E. L. Lawler. Recent Results in the Theory of Machine Scheduling. In *Mathematical Programming: The State of the Art*, pages 202–233, 1983.

[LB01] Samir Loudni and Patrice Boizumault. A new hybrid method for solving constraint optimization problems in anytime contexts. In *Proceedings of the Thirteenth IEEE International Conference on Tools with Artificial Intelligence (ICTAI'2001)*, pages 325–332, Dallas, USA, November 2001. IEEE Computer Society.

[Liu73] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real Time Environment. In *Journal ACM*, 20(1):46–61, 1973.

[Ram90] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. In *10th International Conference on Distributed Computing Systems*, pages 108–115, 1990.

[San96] F. E. Sandnes. A hybrid genetic algorithm applied to automatic parallel controller code generation. In *8th IEEE Euromicro Workshop on Real-Time Systems*, 1996.

[TBW92] K. Tindell, A. Burns, and A. Wellings. Allocation Hard Real-Time tasks: An NP-Hard Problem Made Easy. In *The Journal of Real-Time Systems*, 4(2):145–165, 1992.

[TC94] K. Tindell and J. Clark. Holistic scheduling Analysis for Distributed Hard Real-Time Systems. In *Euromicro Journal*, pages 40–117, 1994.

[Tsa93] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.