Chapter 5

On the First SAT/CP Integration Workshop

SAT and CP techniques are two problem solving technologies which share many similarities, and there is considerable interest in cross-fertilizing these two areas. The techniques used in SAT (propagation, activity-based heuristics, conflict analysis, restarts, etc.) constitute a very successful combination which makes modern DPLL solvers robust enough to solve large real-life instances without the heavy tuning usually required by CP tools. Whether such techniques can help the CP community develop more robust and easier-to-use tools is an exciting question. One limitation of SAT, on the other hand, is that not all problems are effectively expressed in a Boolean format. This makes CP an appealing candidate for many applications, like software verification, where SAT is traditionally used but more expressive types of constraints would be more natural.

The goal of the CP'2006 first workshop on SAT and CP integration was to boost the discussions between the SAT and CP communities by encouraging submissions at the border of these two areas [HAM 06]. This chapter gives a high level overview of the event with a special focus on the questions addressed to the participants during the panel session.

In the following, section 5.1 gives an overview of the technical program. Section 5.2 summarizes the discussion of the panel and section 5.3 gives a general conclusion along expected directions for the SAT and CP thematic.

Chapter written by Youssef Hamadi and Lucas Bordeaux.

5.1. The technical program

The day event started with one invited talk followed by eight 25 minute technical presentations. Robert Nieuwenhuis, the invited speaker presented SMT as a possible connection between SAT and CP. The morning session presented works related to pseudo-Boolean, interval constraints, interpolant-based decision procedure and local search. The afternoon session, presented works related to CSP and CP. This section quotes the summaries of each presentation. The full papers can be accessed in [HAM 06].

5.1.1. *The invited talk*

R. Nieuwenhuis, SAT Modulo Theories: A Possible Connection between SAT and CP

First we give an overview of SAT Modulo Theories (SMT) and its current state of the art. For this, we use our framework of Abstract DPLL and Abstract DPLL modulo Theories, which enables practical SMT algorithms to be expressed easily and enables us to formally reason about them. After this, we explain our DPLL(T) approach to SMT, a modular architecture for building SMT systems that has now been adopted in several state-of-the-art SMT tools. Experimental results and applications are given within the Barcelogic SMT system. In particular, we discuss its use for CP (including optimization) problems, thus going beyond the typical hard/software verification SMT applications.

5.1.2. *Contributions related to SMT and solver integration*

Shuvendu K. Lahiri and Krishna K. Mehra, Interpolant-based Decision Procedure for Quantifier-Free Presburger Arithmetic

Recently, off-the-shelf Boolean SAT solvers have been used to construct ground decision procedures for various theories, including Quantifier-Free Presburger (QFP) arithmetic. One such approach (often called the eager approach) is based on a satisfiability-preserving translation to a Boolean formula. Eager approaches are usually based on encoding integers as bit-vectors and suffer from the loss of structure and sometimes very large size for the bit-vectors. In this section, we present a decision procedure for QFP that is based on alternately under and over-approximating a formula, where Boolean interpolants are used to calculate the over-approximation. The novelty of the approach lies in using information from each phase (either under-approximation or over-approximation) to improve the other phase. Our preliminary experiments indicate that the algorithm consistently outperforms approaches based on eager and very lazy methods on a set of verification benchmarks.

Belaid Benhamou, Lionel Paris, and Pierre Siegel, Dealing with SAT and CSPs in a Single Framework

We investigate in this work a generalization of the known CNF representation which allows an efficient Boolean encoding for n-ary CSPs. We show that the space complexity of the Boolean encoding is identical to the one of the classical CSP representation and introduce a new inference rule whose application until saturation achieves arc-consistency in a linear time complexity for n-ary CSPs expressed in the Boolean encoding. Two enumerative methods for the Boolean encoding are studied: the first one (equivalent to MAC in CSPs) maintains full arc-consistency on each node of the search tree while the second (equivalent to FC in CSPs) performs partial arc-consistency on each node. Both methods are experimented and compared on some instances of the Ramsey problem and randomly generated 3/4-ary CSPs and promising results are obtained.

Martin Franzle, Christian Herde, Stefan Ratschan, Tobias Schubert, and Tino Teige, Interval Constraint Solving using Propositional SAT Solving Techniques

In order to facilitate automated reasoning about large Boolean combinations of non-linear arithmetic constraints involving transcendental functions, we extend the paradigm of lazy theorem proving to interval-based arithmetic constraint solving. Algorithmically, our approach deviates substantially from *classical* lazy theorem proving approaches in that it directly controls arithmetic constraint propagation from the SAT solver rather than completely delegating arithmetic decisions to a subordinate solver. From the constraint-solving perspective, it extends interval-based constraint solving with all the algorithmic enhancements that were instrumental to the enormous performance gains recently achieved in propositional SAT solving, such as conflict-driven learning combined with non-chronological backtracking.

5.1.3. *Contributions related to the use of SAT techniques to improve CSP/CP solvers*

Christophe Lecoutre, Lakhdar Sais and Julien Vion, Using SAT Encodings to Derive CSP Value Ordering Heuristics

In this section, we address the issue of value ordering heuristics in the context of a backtracking search algorithm that exploits binary branching and the adaptive variable ordering heuristic dom/wdeg. Our initial experimentation on random instances shows that (in this context), contrary to general belief, following the fail-first policy instead of the promise policy is not really penalizing. Furthermore, using SAT encodings of CSP instances, a new value ordering heuristic related to the fail-first policy can be naturally derived from the well-known Jeroslow-Wang heuristic. This heuristic, called min-inverse, exploits the bi-directionality of constraint supports to give a more comprehensive picture in terms of domain reduction when a given value is assigned to (resp. removed from) a given variable. An extensive experimentation on a wide range of CSP instances shows that min-inverse can outperform the other known value ordering heuristics.

Christophe Lecoutre, Lakhdar Sais, Sebastien Tabary, and Vincent Vidal, Nogood Recording from Restarts

In this section, nogood recording is investigated for CSP within the randomization and restart framework. Our goal is to avoid the same situations occurring from one run to the next one. More precisely, nogoods are recorded when the current cutoff value is reached, i.e. before restarting the search algorithm. A set of nogoods such as this is extracted from the last branch of the current search tree and managed using the structure of watched literals originally proposed for SAT. Interestingly, the number of nogoods recorded before each new run is bounded by the length of the last branch of the search tree. As a consequence, the total number of recorded nogoods is polynomial in the number of restarts. Experiments over a wide range of CSP instances demonstrate the effectiveness of this approach.

Guillaume Richaud, Hadrien Cambazard, Barry O'Sullivan, and Narendra Jussien, Automata for Nogood Recording in Constraint Satisfaction Problems

Nogood recording is a well-known technique for reducing the thrashing encountered by tree search algorithms. One of the most significant disadvantages of nogood recording has been its prohibitive space complexity. In this section we attempt to mitigate this by using an automaton to compactly represent a set of nogoods. We demonstrate how nogoods can be propagated using a known algorithm for achieving generalized arc consistency. Our experimental results on a number of benchmark problems demonstrate the utility of our approach.

5.1.4. *Other contributions*

Colin Quirke and Steve Prestwich, Constraint-Based Sub-search in Dynamic Local Search for Lifted SAT Problems

Many very large SAT problems can be more naturally expressed by quantification over variables, or *lifting*. We explore implementation, heuristic and modeling issues in the use of local search on lifted SAT models. Firstly, adapting existing local search algorithms to lifted models creates overheads that limit the practicality of lifting, and we design a new form of dynamic local search for lifted models. Secondly, finding a violated clause in a lifted model is an NP-complete problem called *subsearch*, and we show that subsearch benefits from advanced constraint techniques. Thirdly, lifting provides the opportunity for using SAT models that would normally be ignored because of their poor space complexities, and we use alternative SAT-encodings of a constraint problem to show that such lifted models can give superior results.

Jan-Georg Smaus, Representing Boolean Functions as Linear Pseudo Boolean Constraints

A linear pseudo-Boolean constraint (LPB) is an expression of the form $a_1 l_1 + ... + a_m l_m >= d$, where each l_i is a literal (it assumes the value 1 or 0 depending on whether a propositional variable xi is true or false) and the $a_1, ..., a_m$, d are natural numbers. The formalism can be viewed as a generalization of a propositional clause. It has been said that LPBs can be used to represent Boolean functions more compactly than the well-known conjunctive or disjunctive normal forms. In this section, we address the question: how much more compactly? We compare the expressiveness of a single LPB to that of related formalisms, and we give a statement that outlines how the problem of computing an LPB representation of a given CNF or DNF might be solved recursively. However, there is currently still a missing link for this to be a full algorithm.

5.2. The panel session

In an attempt to have a lively and controversial discussion, a list of questions, sometimes provocative, was addressed to the workshop participants. We present here for each question the starting viewpoints taken by the workshop organizers along with a summary of the standpoints taken by the most active participants. Each standpoint comes with the name of these participants who were asked to redevelop their own ideas in a short paragraph.

5.2.1. *Are SAT and CP different or similar?*

In order to support this interrogation, two different views were offered to the participants.

– The first view was taken by a reviewer commenting on [BOR 06]:

The two approaches could not be more different. SAT is a black-box using a minimal language; CP is a white-box with rich search and constraint languages. Both have advantages and drawbacks.

– The second view came from a colleague commenting on the SAT/CP workshop:

I think the frontier between SAT/CSP/ILP (and Max-SAT/Max-CSP) is becoming increasingly loose and permeable[1].

[**M. Leconte**] CP contains SAT as it is easy to express a SAT problem for a CP Solver. In CP, we could implement a global constraint handling SAT problems the very same way SAT Solvers perform unit-propagation. The issue is then to let inferences from this SAT constraint benefit other constraints. The DPLL(T) architecture of SMT (Sat Modulo Theory) can be seen as an extension of the CLP(X) schema but, as far as CP finite domain solvers are concerned, constraints communicate only via their domains seen as a set of possible values. This is called non-relational domain in the Abstract Interpretation field and one can wonder whether such communication channels would be enough for effective cooperation between the global SAT constraint and the usual CP ones. This makes SAT and CP very different as SAT uses clauses to record nogoods whereas finite domain CP is restricted to removing possible values for variables.

[**J. Marques-Silva**] SAT and CP differ extensively. CP provides rich modeling features, SAT does not. SAT algorithms can be extremely simple, CP algorithms cannot. CP has a number of strategic application areas, whereas SAT has other, mostly orthogonal, strategic application areas. It does not seem possible for SAT to easily replace CP in applications where CP has shown results. Similarly, CP is unlikely to replace SAT in the most well-known applications of SAT.

[**P. Stuckey**] SAT and CP are in my opinion very similar. Both have interest in complete and local search methods for tackling their problems. I'll concentrate on complete methods. In these terms search DPLL SAT is almost an instance of CP, unit propagation is one particular choice of propagators, and a particular type of search (selecting a literal to enforce). The perceived difference arises from the convergence of complete SAT solvers to a common set of particular techniques: restarts, nogoods and activity based search heuristics. All of these have been explored in the CP community, but since the problem space is broader, it it is less easy to see it converging to a single *best* solution. CP will converge with SAT by reinvestigating the SAT techniques that have been successful. The other lesson CP takes from SAT, is the importance of standard input language, a problem the CP community clearly admits, and the resulting engineering achievements that can be made once a standard is available. SAT is converging toward CP as well. Higher level modeling is clearly important to both communities and SMT drives SAT to effectively a form of global constraints. The more eager the communication between DPLL and theory T, the more it becomes analogous with propagation. The difference is

1. D. Le Berre, personal communication.

the concentration on learning. I am confident that with better understanding of complete approaches CP and SAT we will end in a single algorithmic toolbox for both classes.

[**R. Yap**] CP has more issues than SAT. For SAT, usually the modeling is not an important component, one assumes one already has an SAT model, i.e. in CNF form. In CP, the model itself might be the issue. The development of global constraints is driven by such modeling questions since global constraints are usually for specific kinds of uses.

5.2.2. *Why has SAT succeeded in reducing the tuning issue?*

Two possible answers were proposed:

– SAT researchers have just tried harder: whereas ease of use was never a fundamental concern in the design of CP tools, the SAT community has from the very beginning aimed at finding a universal SAT algorithm integrating many techniques in a transparent way (activity-based heuristics, restarts, learning, etc.); furthermore, by using large sets of real-life instances, the SAT community was very effective at evaluating and comparing the robustness of the algorithms;

– SAT researchers have *cheated*: the claim that SAT solvers are universal reasoning tools is over-stated, because they focus on particular classes of applications (essentially verification and transition systems), and their instances have a very specific structure (low tree-width; unsatisfiable instances have a small core, *etc.*).

[**M. Leconte**] Since SAT models are at a lower level than the CP ones, it is more difficult for the user to indicate an efficient search strategy to an SAT solver than a CP solver. This is because the variables of the original problem are generally not directly represented in the SAT formalism whereas they correspond to decision variables in CP. In a sense, this makes a specific search strategy much harder to express on a SAT model than on a CP model. This shows that generic solving is crucial for SAT solvers; this is a reason for the focus and the success of the SAT community in reducing the tuning issue. Note that the same story happened in the MIP community: modern MIP solvers are efficient enough to be used without specific tuning. We can hope for and believe that CP will follow this path.

[**J. Marques-Silva**] SAT solvers are solving a much simpler problem. This makes SAT solvers much easier to tune. The availability of representative problem instances contributed decisively to the development of optimized SAT solvers. I do not think SAT researchers have *cheated*! The performance results of modern SAT solvers are indeed impressive, but this is for problem instances from concrete application domains (e.g. model checking, etc.). In a good number of application areas SAT is not effective, and this is known.

[**R. Nieuwenhuis**] SAT researchers have not cheated. The restricted formalism of SAT has made it possible, after a lot of research, to come up with several key ideas:

a) good general-purpose heuristics based on literal activity. I see this as locally working from one constraint *cluster* at a time (extracting the right lemmas from it);

b) the right conflict analysis mechanism (called *1UIP*), that allows us to backjump and at the same time provides the lemmas (nogoods in the form of new clauses) that work well with the heuristic; cheap enough ways of generating smaller – and hence in principle stronger – lemmas are known, but tend to perform worse;

c) refined data structures for unit propagation (two-watched literals), clause representation, and bookkeeping for the heuristics.

For most real-world problems admitting a reasonable-sized reduction to SAT, this black-box setting works impressively well, and most attempts at problem-specific tuning only worsen the performance. This is also the case for problems from new application domains that keep coming up.

I believe that CP's generality and diversity makes it difficult to achieve something similar. Another disadvantage for CP is the fact that the community has been – and still is – being mislead by artificial problems. Even nowadays most CSP solver competition problems are artificial or are translations from artificial problems. SAT experts know very well that random or handcrafted SAT is completely different from real-world SAT. For example, in random SAT lemma learning is useless and the heuristics are completely different.

[**B. O'Sullivan**] The advantage of SAT is that it is based on a standard simple input language. The solvers tend to be regarded as black-boxes that can be tuned. This has provided a common basis for specifying industrial benchmarks in a standard way. Therefore, the solvers tend not to focus on modeling and reformulation concerns, but on how to tune the solver to best solve the instance. Researchers are therefore provided with an opportunity to focus on developing good solvers and not on developing a complex input language which has been challenging for CP. Of course, there are advantages with having a complex input language. Specifically, in CP we can be true to the actual structure of the problem and formulate a model that is more natural for humans to interpret and debug.

[**T. Walsh**] SAT is such a simple language. We typically only consider sets of clauses. The problem is coming up with an encoding into SAT that works well. Once we have a set of clauses, the language is very flat and uniform. The SAT competitions have also allowed us to come up with good uniform parameter settings, default heuristics, etc. This contrasts with CSP where we have a very rich language and tuning is correspondingly more difficult. In addition, CP has not had a long running competition so default heuristics are still relatively primitive (e.g. fail first, domain/degree).

5.2.3. *How long can the current generation of SAT solvers evolve?*

This question was illustrated by a recent statement from an SAT colleague:

> *I think that we found the right combinations of components for successful SAT solvers*[2].

It is also interesting to note for instance that, among the 37 papers presented in the technical tracks of the SAT 2006 symposium [BIE 06], very few presented direct improvements of the state-of-the-art DPLL architecture. The topics this year were: *proofs and cores*, *heuristics and algorithms*, *applications*, *Satisfiability Modulo Theories*, *structure*, *Max-SAT*, *local search and survey propagation*, *Quantified Boolean Formulas* and *counting and concurrency.* The *heuristics and algorithms* session included 4 papers, some related to resolution-based solvers or non-clausal formulas, and essentially *one single* paper proposed improvements to a central component of the standard DPLL solvers, namely the *restarts* aspect.

There are signs that DPLL with the standard combination (propagation based on 2-watched literals, conflict analysis based on variants of 1-UIP, activity-based heuristics and restarts) are the last word on complete, search-based SAT solvers. On the one hand this would be a considerable achievement; on the other hand this is a worrying sign of stagnation. Stagnation which may come from the propinquity effect generated by yearly SAT-competitions.

The question is therefore whether this situation is temporary, and new techniques will be found, or whether one can predict that the performance of SAT solvers will essentially remain stable during the next years.

[**I. Gent**] In 1993, I sat on a panel at a DIMACS workshop. I remember saying that I had been impressed how much SAT solvers had improved just in the last year or two, and I thought that might continue and so enable us

2. K. Sakallah, personal communication.

to solve much larger instances. I got the impression that few agreed with me, based on the obvious argument of the combinatorial explosion. In one sense I was wrong, since I was mainly thinking of Random 3-SAT instances, and for now the record is only at about 700 variables for a complete solver. However, I was right in the bigger picture, as SAT solvers have become technologically marvelous.

This is a long winded way of saying *maybe*. First of all, something surprising might come along and revolutionize the way we build SAT solvers. I think there was a slight feeling of stagnation before Chaff revolutionized SAT solvers around 2000, so some advance of that scale would get the field going again. Another answer is that the scale of effort needed may be increasing, which is a good thing and a bad thing. It is a good thing if the field is mature with well-known techniques which can be built on by a large team, and researchers should continue to go after the difficult big wins after the easy big wins have been achieved. On the other hand, it is a bad thing if researchers with good ideas have no chance of competing with the top teams.

I think it is a common experience amongst communities with competitions that they can lead to diminishing returns over a few years. It may be necessary to think of ways of avoiding this, for example running competitions on instances which current SAT solvers are bad at, to encourage experimentation. The problem with this is obviously that of getting successful entries!

[**M. Leconte**] Let us look at MIP solvers. Their expressive power is only between SAT and CP. As black-boxes, their efficiency is also only between these two. It took quite a long time for the MIP community to recognize the value of presolving. The common belief was that presolving techniques were useful only on bad models, and globally useless. As long as the efficiency of MIP solvers improved, the solved models became bigger and presolve gained more and more impact. Nowadays, all modern MIP solvers have a presolve phase. This is mainly what SAT solvers are missing compared to MIP solvers. Presolving could be the next subject to be (re)investigated.

[**J. Marques-Silva**] Recent years have not seen a dramatic improvement in SAT solvers. This suggests that improvements to the current organization of conflict-driven clause learning SAT solvers will become difficult to identify and justify. There is always the possibility of another paradigm shift in SAT solver technology, but performance data from the past couple of years suggests that the performance of SAT solvers is not improving significantly.

[**R. Nieuwenhuis**] I believe that there is still room for improvement, especially for finding models, i.e., for satisfiable problems. This is closely related to the issue of restarts, which appear to be not completely understood. There is of course also a lot of work to be done in extensions, such as Max-SAT and other optimization problems, all-SAT, proof generation, unsatisfiable cores, or, of course, SAT modulo theories.

5.2.4. *Were performance issues correctly addressed by CP?*

The CSP/CP community addressed performance issues by pushing on the inference side and by completely avoiding intelligent look-back techniques or nogood-recording strategies. The heavy commitment to global constraints is part of this quest for more inference, sometimes only possible when a whole sub-problem is analyzed at once by a specific method.

A legitimate question is whether this focus on propagation, was justified: is it the case that conflict analysis, activity-based heuristics and other techniques that are so useful in SAT simply do not pay for CP? Or has the CP community neglected these techniques because the standards it followed for evaluating algorithms was different (and admittedly *poorer*) from the ones used in SAT?

[**C. Bessiere**] Conventional wisdom on conflict-directed techniques differs in SAT and CP essentially because of the inherent differences in the encoding of problems and in the characteristics of standard solvers.

Let us recall that an encoding (or model) for a problem is a SAT formula or a constraint network that makes a number of nogoods explicit via its clauses or constraints. These explicit nogoods subsume many other nogoods that are implicit. Implicit nogoods can cause thrashing if they are not detected early by the search procedure. Inference (or constraint propagation) and conflict-directed techniques are two orthogonal methods of trying to avoid thrashing. Inference uses explicit nogoods to detect implicit ones *before* committing to inconsistent subspaces. Conflict-directed techniques use information contained in a dead-end to extract a culprit nogood and to use it to discard future subspaces inconsistent because of this nogood. SAT and CP solvers all use some form of inference. The standard level of inference, called unit propagation in SAT and arc consistency in CP, is to remove a value v for variable X (unary nogood) only if there exists a clause/constraint c for which all valid assignments of the variables of c containing v for X are forbidden by c. Now, a SAT clause represents a single nogood whereas a CP constraint is a compilation of several nogoods. As a result, in most cases, standard inference prunes more inconsistencies in CP than in SAT. Since conflict-directed techniques are useful only if inference has missed a nogood, there are more chances that they improve a DPLL-like search procedure in SAT than an arc consistency-based backtracking in CP. This can explain in part the discrepancy between the respective conventional wisdom in SAT and CP about conflict-directed techniques. Furthermore, in CP, if we increase the number of nogoods considered jointly during propagation (by propagating more than arc consistency or by using global constraints), we increase the amount of implicit nogoods found by CP solvers. The CP community observed that the more we increase the level of propagation during search, the smaller the number of problems on which conflict-directed techniques pay off [BES 96, CHE 01]. However, it was shown in [JUS 00] that we can build instances of problems on which standard arc consistency always thrashes whereas adding some form of backjumping allows a smooth solving.

There are other characteristics of SAT solvers that can explain the success of conflict-directed techniques such as nogood learning. Recent SAT solvers use the *restart* technique. Restart has been shown to combine extremely well with nogood learning. The fact that nogood learning can greatly pay off when maintained over several searches was observed in dynamic constraint problems, where the problem is repeatedly solved after small changes in the constraints [SCH 93]. However, it has not been applied to standard CP solvers. One reason for this is perhaps that non-unary nogoods are much easier to handle in SAT than in CP. A nogood is extremely easy to express as a clause to be used by a SAT solver. On the contrary, non-unary nogoods are not constraint objects that can easily be handled by the current generation of CP solvers.

[**I. Gent**] There is certainly a lot of research on backjumping and learning in CP. However, there is one important reason it has not yet taken off in practical solvers. Whenever a constraint propagates in a backjumping system, it is required to provide a conflict set responsible for the propagation. In SAT, this is trivial: when a unit clause is created the conflict set is the other literals in the original version of that clause. In CP, propagators can be very complicated and (even worse) the global nature of some makes it hard to construct a useful conflict set, i.e. of a size much less than the number of variables being searched on. This does not mean that backjumping and learning is doomed, but making it pay off in practice is hard.

I agree that the CP community has very low standards in benchmarking, and I have been guilty of this as many others have. I do not agree that the standard in SAT is necessarily better. SAT does have advantages of a standard input format and a large number of instances in that format, but this still allows for poor experiments poorly done, just as CP practitioners can perform good experiments very well.

It is true that it has been perceived to be hard to construct an outstanding constraint solver, and this has inhibited the number of world-class solvers constructed by research teams. In fact this perception is false, as we recently showed with the Minion constraint solver, the first version of which (almost all by Chris Jefferson) was the result of only a few man-months of work. A major lesson we have learned is that people can build high-performance solvers just as we did, and we would encourage people to do so if it would help them incorporate novel techniques and prove their worth. If they don't want to spend the time building their own solver, they can add code to one of the other open source solvers: not just Minion but also Gecode, Choco or Eclipse.

[**M. Leconte**] In the early days, performance issues of *automatic* systems such as Alice from J-L. Laurière in 1978 were addressed by *opening the box.* Ten years later, the *white-box* approach or *control on search*, was first highlighted by the CHIP system from ECRC. Still for efficiency reasons, to let users implement global constraints, the glass-box approach or *implementing your own constraints* was introduced by ILOG in 1995. These answers effectively bring efficiency to CP systems but at the cost of user-specified control. Use of CP systems became more and more difficult as well as time-consuming. ILOG have been working on the ease of use of our CP solver for a few years. The next ILOG CP solver (due out in 2007) provides an efficient default search strategy generic enough to remove (or reduce) the need for specific tuning. We have been trying to repeat the MIP solvers' story which also succeeded in reducing the tuning issue while maintaining good performances. The next step, as in MIP, would be to incorporate presolving techniques such as model reformulations or simplifications.

[**J. Marques-Silva**] In the early 1990s the evaluation of CP solvers was not convincing, with some emphasis on randomly generated problem instances. This may in part justify the emphasis on propagation/inference observed in CP. I believe things have changed over the last few years. The CP competition is a good example, and has been important for motivating the development of better CP tools. In contrast, from the mid 1990s the development of SAT solvers was motivated by concrete applications (e.g. test pattern generation, circuit delay computation, hardware model checking, software testing and model checking, etc.). This enabled the selection of the best techniques for solving these concrete problems.

[**P. Stuckey**] I suspect performance issues have always been of interest to the CP community, but a CP solving engine (here I mean for complete search) is a large software undertaking. A competitive SAT solver may be less than a thousand lines of code. Hence SAT performance is an easier thing to investigate, by an order of magnitude. A standard language for specifying CP problems is obviously missing, but this won't fundamentally change the problem that investigating design choices for performance in CP will always be much harder work than in SAT. The focus on propagation has been justified I think, by large industrial CP/MIP solutions, where while the MIP solver is doing the majority of the solving, the propagation of CP is key to modeling and solving some parts of the problem. Many of these industrial solutions do not have large search trees, but involve substantial work at each node, so conflict analysis and other techniques have less of a role to play.

[**T. Walsh**] CP researchers have not agreed on a common input language format (unlike SAT where everyone accepts DIMACS). Therefore comparing different solvers is difficult. This must be fixed in the near future if CP is to progress rapidly.

CP has also neglected issues like nogoods, learning and non-chronological backtracking. This is partly technological. The older solvers around today were engineered before we had understood these methods and it would be difficult to re-engineer them with nogoods. However, it is also because the research community was misled by looking at random problems. With random problems, techniques like nogood learning are of limited value. The SAT community missed out this trap as nogoods are clearly of great value in domains like hardware verification.

[**R. Yap**] The CP community has not addressed the performance issues as seriously as in SAT. This is related to question 1. We do not even have agreement on common benchmarks, different solvers support different constraints, etc. Thus, it is less clear how to even carry out proper benchmarking.

5.2.5. *Was CP too ambitious?*

Modern CP toolkits come from a long tradition of *General Problem Solving*, a quest that was pursued from the early days of AI (see e.g., [NEW 60]). It is worth remembering that all attempts at general problem solving failed, because they were too ambitious.

Constraint programming's response to the apparent impossibility of building a completely general and automatic general problem solver was to open the solver and to let the user have some control: with this *glass-box* approach, CP allows the user to tailor the system to their application space – generality is preserved, but not the full automation. The approach of the SAT community was different: by focusing on a restricted language and on a restricted class of applications for which this language is a natural fit, SAT solvers follow the dream of being fully automated, but abandon the dream of being fully general.

The question is therefore whether there is a way to reconcile these apparently orthogonal approaches, whether some of the original ambition of CP should be given-up, whether the dream of general problem solving is still worth pursuing, and, if so, whether CP or SAT is the best candidate.

[**C. Bessiere**] The best candidate for a general problem solver is neither SAT nor CP, it is AI.

This is true that all the attempts at a general problem solver have failed in the past. SAT has a limited expressiveness but thanks to its simplicity, it is good at automatically adapting the model to improve the search. Thanks to its very high expressiveness CP allows representations very close to the nature of the problem, but the rigid separation between modeling and solving forces the user to go back and forth from modeling to solving to improve their model. The positive side is that these strengths and weaknesses are now quite well understood in both communities. There has been some work in SAT to integrate into the model some of the lost structure of the problem [KAS 90, KAU 99, GEN 02]. In CP, there were preliminary attempts at adapting the solver to the instance to be solved [LOB 98, EPS 01]. If the general problem solver exists, it will need help from other AI fields. Case-based reasoning, machine learning, or knowledge compilation can certainly be used to draw lessons from previously solved instances, for self-adaptivity of the solver to the instance, or for model reformulation.

[**J. Marques-Silva**] Generality often comes at a price. Work on SAT has always been quite focused. In contrast, the CP quest for general problem solving may have come at a cost. Nevertheless, I do not think that SAT will ever be a strong candidate for general problem solving. In contrast, SMT solvers are much more general than SAT solvers, and may well successfully challenge CP in the near future.

5.2.6. *Do we still need CP?*

With this provocative question we wanted to stress the fact that several important research fields consider SAT solvers as trouble-free problem solvers, e.g. knowledge representation [GIU 06], biology [LYN 06], planning [KAU 92], etc.

On the contrary, few application domains naturally consider using CP. One possible explanation is that people understand more easily what solvers for the famous SAT problem are about. After all, propositional logic is widely known; building a modeling on it and using a solver able to understand it may look like an easy path for many end-users. Another possible explanation is that people assume (perhaps wrongly) that CP is less efficient, or more difficult to use.

An interesting indicator of this fact is the current interest of the computer aided verification community for satisfiability modulo theories (SMT). This is an application domain where people need to solve problems that have a large

Boolean part, but also include constraints on other domains, such as integers and arrays. The most successful attempts to build solvers for this large class of problems started from an SAT solver and extended it with other theories [NIE 07]. Another natural approach would be to start from a CP solver and specialize or adapt it. This approach has simply not been considered by the CAV community. This may come from their long-time commitment to DPLL solvers, and to the importance of the Boolean part in their formulations.

It is therefore important for the CP community to assess its position against SAT: which advantages does CP offer, in which cases should it be used, and why is it not considered in more of the aforementioned application areas?

[**J. Marques-Silva**] I am not sure the CP community should assess its position against SAT. SAT technology is useful in a number of contexts, CP in other contexts. CP is more general, SAT is more focused. In contrast, the CP community should probably pay more attention to the work being done in SMT. SMT solvers can be extremely general, and so could compete with CP in a number of contexts. I believe the contributions of CP justify the future of CP. I would like to see CP people involved in the development of the next generation of SMT solvers. This might be beneficial for both communities.

[**B. O'Sullivan**] A significant difference between constraint programming and SAT is that the former tends to focus more on optimization problems. Typically, in constraint programming we wish to differentiate between alternative solutions in terms of some measure of overall cost, reliability or cost. These can be incorporated more easily, in some sense, in a constraint-based approach since we can exploit the structure of the problem to give good bounds on the objective function. Also, there are many tools available to hybridize with traditional operations research techniques. Systematic SAT approaches have typically focused on verification and satisfiability checking, but there are specialized optimization problems such as Max-SAT that can be solved well in this framework. Local search techniques give rise to new opportunities for optimization in both SAT and CP. However, CP offers greater opportunities for expressing the real structure in an optimization problem which is lost in the traditional CNF encodings used in SAT. Therefore, in the case of optimization I would argue that CP is a more appropriate technology.

[**R. Yap**] The work in SMT could be thought of as an extension of SAT. However, I think it seems to be quite analogous to the development of CLP. Constraints were the natural way of making the logic programming languages and formalism more powerful. So SMT also uses CP in the same way.

5.3. Summary, future directions and conclusion

In any new thematic, low-hanging fruits represent the first catches, and this SAT/CP event verified this traditional saying. Not surprisingly, the first wave of SAT/CP integration research tried to blend some of the most successful SAT techniques into current CSP/CP frameworks. We can see this in some of the sections presented here [HAM 06] but also in some recently published conference papers [GEN 06]. In this respect it is interesting to remark that in SAT it is not a particular technique *in isolation* that explains the success of current DPPL solvers, but the miraculous alchemy between the fast 2-watch propagation (which is not slowed-down too much when clauses are added, thus making clause-learning practical), the 1UIP conflict-analysis (which uses the fact that the *implication graph* can be recovered cheaply from the data-structures used for propagation), and the activity-based heuristics (which is nourished by the learning), and the other components (e.g. restarts). It is fair to say that we are still far from such a finely-tuned integration of techniques in CP, and that finding the right alchemy will be much more challenging than in SAT.

It seems reasonable to forecast the integration of a SAT solver as part of a CP engine. We see two potential consequences on the future of CP tools:

– The impressive achievements of the SAT community will very probably impact the CP community positively: the next generation of CP tools will be able to handle the Boolean part of CP problems much more efficiently thanks to the lessons learned from SAT solvers. Many combinatorial problems include a significant Boolean part together with other types of constraints. It was long assumed in CP that the Boolean part could be handled just as the rest (a Boolean variable is just a normal discrete variable with domain 0/1, it is propagated using the general-purpose propagation techniques of the solver). There is now every evidence, as shown by Minion, that specialized methods to deal with the Boolean part do make a difference.

– If the integration of SAT techniques within CP is successful, one can conceive that the interest for purely Boolean solvers will diminish. At the time of writing, using a general CP tool to solve a purely Boolean problem is out of question: the performance would simply not be acceptable compared to a specialized SAT solver. If CP solvers reach a performance that is reasonably competitive to SAT on purely Boolean instances, then many more users might feel tempted to consider CP solvers as their default problem solving tool. This would, in our opinion, improve on a situation which is currently paradoxical: in too many cases, problems are nowadays encoded to SAT even though part of their constraints are originally non-Boolean. The reason is twofold: first SAT solvers are so easy to use compared to CP tools that the effort of encoding the non-Boolean part into CNF appears acceptable; second, the speed-up on the Boolean part is so considerable that it often outweighs the possible slow-down incurred by the non-natural encoding of the non-Boolean part. In a sense, the success of SAT has been a lesson whose positive outcome for CP might be that it would force us to analyze why SAT solvers are used for problems where CP solvers would appear to be more natural candidates, and how we can correct that.

Another interesting thing to come of this first workshop is the growing importance of the satisfiability modulo theories formalism and its potential connection with existing CLP/CP research. It seems reasonable to forecast a possible integration CP/SMT where existing CP results would be reused in specific SMT theories. We can think of specific propagators for constraints over integers, lists, reals, which have been well studied in CLP/CP. One possible challenge in using these CP results for SMT is that in SMT the algorithms for each theory do not communicate solely by propagation. In CP, the communication between constraints is typically done by deductions of only one form: *variable x cannot be assigned to value v*. In most existing frameworks for satisfiability modulo theories, such as the traditional Shostak and Nelson-Oppen methods or the recent DPLL(X) framework [NIE 07], the decision procedures typically exchange richer information, for instance equalities (each decision procedure is able to generate deductions of the form *variables x and y are equal*) or proofs (each decision procedure is able to generate an explanation of which hypothesis was really used to produce a particular deduction). In his invited talk, R. Nieuwenhuis explained the need for this rich information exchange by saying that [each component] *guides* the search instead of only *validating* it. For many types of constraints considered in CP, such a rich communication remains to be studied. In addition to improving the impact of CP results on SMT, the philosophy summarized by R. Nieuwenhuis might inspire new perspectives in solver cooperation for CP itself.

The workshop was the first event devoted specifically to the issues of cross-fertilization between SAT and CP; it can be safely predicted that it will not be the last, and that more papers on this issue will also be published in mainstream conferences in the near future. Our personal guess is that the contributions related to the advancement of SAT/SMT from a CP perspective will be among the most exciting of these forthcoming contributions.

5.4. References

[BES 96] Bessiere C., Régin J., "MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems", *Proc. of Int Conf. on Principles and Practice of Constraint Programming (CP)*, p. 61-75, 1996.

[BIE 06] Biere A., Gomes C., *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference*, LNCS 4121, Springer, 2006.

[BOR 06] Bordeaux L., Hamadi Y., Zhang L., "Propositional Satisfiability and Constraint Programming: a Comparative Survey", *ACM Computing Surveys*, vol. 38, num. 4, p. 1-54, 2006, Article 12.

[CHE 01] Chen X., van Beek P., "Conflict-directed backjumping revisited", *J. of Artificial Intelligence Research*, vol. 14, p. 53–81, 2001.

[EPS 01] Epstein S., Freuder E., "Collaborative learning for constraint solving", *Proc. of Int Conf. on Principles and Practice of Constraint Programming (CP)*, p. 46–60, 2001.

[GEN 02] Gent I., "Arc consistency in SAT", *Proc. of European Conf. on Artificial Intelligence (ECAI)*, p. 141-145, 2002.

[GEN 06] Gent I. P., Jefferson C., Miguel I., "Watched Literals for Constraint Propagation in Minion.", *Proc. of Int Conf. on Principles and Practice of Constraint Programming (CP)*, p. 182-197, 2006.

[GIU 06] Giunchiglia F., "Managing Diversity in Knowledge", *Proc. of European Conf. on Artificial Intelligence (ECAI)*, Page 4, 2006.

[HAM 06] Hamadi Y., Bordeaux L., Proceedings of the First Workshop on the Integration of SAT and CP techniques, Report num. MSR-TR-2006-153, Microsoft Research, Nov 2006.

[JUS 00] Jussien N., Debruyne R., Boizumault P., "Maintaining arc consistency within dynamic backtracking", *Proc. of Int Conf. on Principles and Practice of Constraint Programming (CP)*, p. 249–261, 2000.

[KAS 90] Kasif S., "On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks.", *Artificial Intelligence*, vol. 45, num. 3, p. 275-286, 1990.

[KAU 92] Kautz H. A., Selman B., "Planning as Satisfiability", *Proc. of European Conf. on Artificial Intelligence (ECAI)*, p. 359-363, 1992.

[KAU 99] Kautz H., Selman B., "Unifying SAT-based and Graph-based Planning", *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI)*, p. 318-325, 1999.

[LOB 98] Lobjois L., Lemaître M., "Branch and Bound Algorithm Selection by Performance Prediction", *Proc. of Nat. Conf. on Artificial Intelligence (AAAI)*, p. 353-358, 1998.

[LYN 06] Lynce I., Marques-Silva J., "SAT in Bioinformatics: Making the Case with Haplotype Inference", *Proc. of Int. Symp. on Theory and Applications of Satisfiability Testing (SAT)*, p. 136-141, 2006.

[NEW 60] Newell A., Shaw J. C., Simon H. A., "Report on a general problem-solving program", *Proc. of IFIP World Computer Congress*, UNESCO, p. 256-264, 1960.

[NIE 07] Nieuwenhuis R., Oliveras A., Tinelli C., "Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)", *J. of the ACM*, 2007, forthcoming.

[SCH 93] Schiex T., Verfaillie G., "Nogood recording for static and dynamic CSPs", *Proc. of Int. Conf. on Tools for Artificial Intelligence (ICTAI)*, IEEE, p. 48-55, 1993.