



SAT as an effective solving technology
for constraint problems
Preliminary report

Marco Cadoli, Toni Mancini, Fabio Patrizi

Dipartimento di Informatica e Sistemistica

Università di Roma “La Sapienza”

Via Salaria 113, 00198 Roma

`cadoli|tmancini|patrizi@dis.uniroma1.it`

`http://www.dis.uniroma1.it/~cadoli/research/projects/NP-SPEC`

CILC 2005

Roma, June 22nd, 2005

Outline

- Technologies for solvers
- The NPSPEC project
- SPEC2SAT: system & algorithm
- Evaluation on the CSPLib
- Conclusions, current work

Current technologies for solving combinatorial problems

- **ad hoc** algorithms (complete or incomplete: tabu search, simulated annealing, ...),
- proprietary programs for **Integer Programming** (e.g., **cplex**), possibly interfaced through high level languages, e.g., **AMPL**,
- **libraries for constraint programming** (proprietary or not: ILOG, CLP(FD), ...) interfaced through general purpose programming languages (C++, Java, Prolog, ...),

Current technologies for solving combinatorial problems (cont.)

- general purpose **constraint modelling and programming** languages (typically, proprietary: OPL, Xpress, GAMS, ...),
- constraint modelling and solving systems based on **Answer Set Programming** (freeware: dlv, smodels, ASSAT, Cmodels, ...),
- ad hoc **translation into another problem** (often SAT),
- ...

Current technologies for solving combinatorial problems (cont.)

- **general purpose constraint modelling and programming** languages (typically, proprietary: OPL, Xpress, GAMS, ...),
- constraint modelling and solving systems based on **Answer Set Programming** (freeware: dlv, smodels, ASSAT, Cmodels, ...),
- ad hoc **translation into another problem** (often SAT),
- ...

Wish list for a CP system

- declarative approach,
- strong decoupling between:
 - instance representation,
 - constraint modelling, and
 - constraint programming,
- freeware,
- good quality (efficient, robust, ...),
- ...

Outline

- Technologies for solvers
- The NPSPEC project
- SPEC2SAT: system & algorithm
- Evaluation on the CSPLib
- Conclusions, current work

Our proposal: NP_{SPEC}/ SPEC_{2SAT}

- **declarative:** syntactic distinction between:
 - *instance*
 - *search space structure*
 - *constraints*
 - *constraint programming*
- **logic-based**
- **clear formal properties**
- **efficient**
- **freeware**



The NPSPEC project: Brief history

1998: formal properties [C & Palopoli]:

semantics: non-deterministic extension of DATALOG

data complexity: NP-complete

expression complexity: NE-complete

expressive power: captures NP (Fagin's theorem)

1999: pilot implementation (translation into *ECLⁱPS^e*) [C, Ianni, Palopoli, Schaerf, & Vasile]

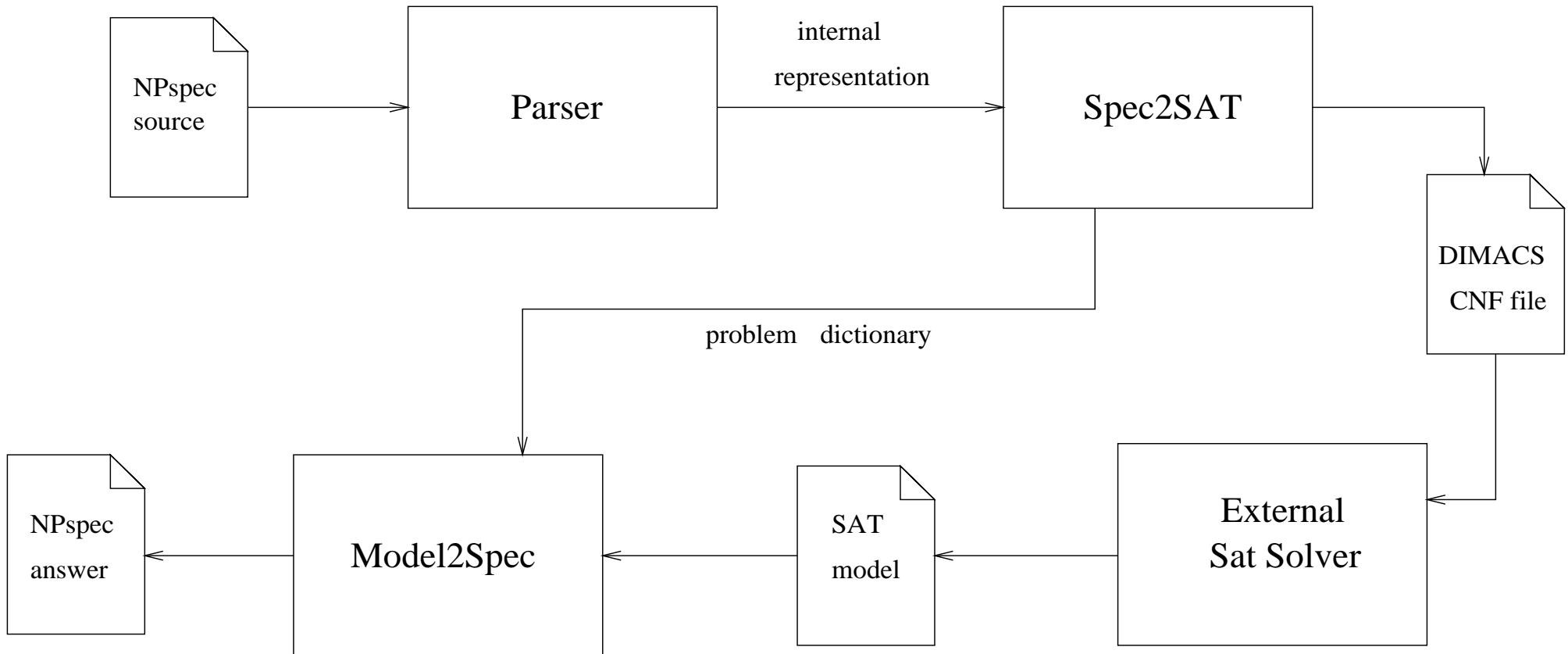
2001: translation into SAT [C & Schaerf]

2004: reengineering; available on the Web [C, Mancini & Patrizi]

Outline

- Technologies for solvers
- The NP_{SPEC} project
- SPEC2SAT: system & algorithm
- Evaluation on the CSPLib
- Conclusions, current work

System architecture



A typical NP-complete problem: 3-coloring

INSTANCE: A graph $G = (N, E)$

QUESTION: Is G 3-colorable, i.e., does there **exist a function**
 $col : N \rightarrow \{0, 1, 2\}$ such that $col(u) \neq col(v)$
for each $\{u, v\} \in E$?

Using NPSPEC for specifying 3-coloring

```
// 0. Instance
DATABASE
  NODE = {1..6};
  EDGE = {(1,2), (1,3), (2,3), (6,2), (6,5), (5,4), (3,5)};
SPECIFICATION
// 1. Search space (using a metapredicate)
  Partition(NODE,coloring,3).
// 2. Constraint (Datalog rules)
  fail <-- edge(X,Y), coloring(X,C), coloring(Y,C).
SEARCH
// 3. For efficient translation/solution (optional)
  inline edge;
  multivalued coloring;
```

Expected benefits of the SPEC2SAT approach

- + Get for free state-of-the-art Constraint Processing techniques.
They are embedded into freeware SAT solvers such as **zchaff**, **walksat**, ...
- + SAT solvers get more efficient every year.
- + New way to obtain benchmarks for SAT.

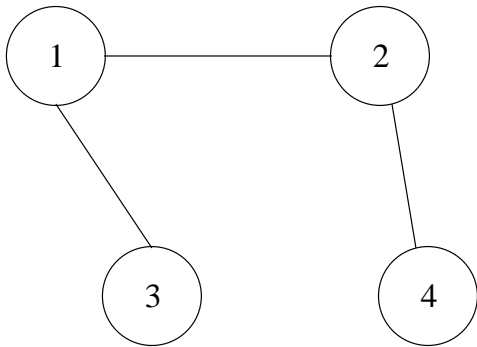
Translation into SAT is a popular technique for solving (efficiently) combinatorial problems.

It has been used for planning, scheduling, theorem proving in finite algebra, generation of test patterns for combinatorial circuits, cryptography ...

Difficulties we had

- Translation algorithm less intuitive than expected.
- Translation has a cost (always polynomial, but sometimes larger than solving the SAT instance)

Main idea of the algorithm: ground instantiation



(a) Problem Instance

```

DATABASE
  NODE = {1..4};
  EDGE = {(1,2), (1,3), (2,4)};
SPECIFICATION
// 1. Search space
  Partition(NODE,coloring,3).
// 2. Constraint
  fail <-- edge(X,Y), coloring(X,C),
           coloring(Y,C).

```

(c) Specification

NPSPEC atom	variable	kind
edge(1,2)	1	α
edge(1,3)	2	
edge(2,4)	3	
coloring(1,0)	4	β
coloring(1,1)	5	
coloring(1,2)	6	
coloring(2,0)	7	
coloring(2,1)	8	
coloring(2,2)	9	
coloring(3,0)	10	
coloring(3,1)	11	
coloring(3,2)	12	
coloring(4,0)	13	
coloring(4,1)	14	
coloring(4,2)	15	
fail	16	γ

(b) Dictionary

Structure of the SAT instance: vocabulary

- α . one variable for every fact in DB ;
- β . one variable for every ground instantiation of a guessed predicate on elements of the relevant domain;
- γ . one variable for every ground instantiation of other predicates.

As an example, for the *coloring* predicate we have $\#nodes \cdot \#colors$ atoms of the kind β .

Structure of the SAT instance: clauses

There are four kinds of clauses:

1. A clause $\{c\}$ for each variable c of the kind α .
2. Clauses using variables of the kind β encoding the meaning of the corresponding metarule.
3. Clauses using variables of the kind α , β , and γ encoding the meaning of the constraints of SP .
4. The clause $\{\neg fail\}$.

Clauses: example

+-----+-----+-----+-----+			
	-4 -5 0		
	-4 -6 0		
	-5 -6 0	16 -1 -4 -7 0	
	4 5 6 0	16 -2 -4 -10 0	
	-7 -8 0	16 -3 -7 -13 0	
1 0	-7 -9 0	16 -1 -5 -8 0	-16 0
2 0	-8 -9 0	16 -2 -5 -11 0	
3 0	7 8 9 0	16 -3 -8 -14 0	
	-10 -11 0	16 -1 -6 -9 0	
	-10 -12 0	16 -2 -6 -12 0	
	-11 -12 0	16 -3 -9 -15 0	
	10 11 12 0		
	-13 -14 0		
	-13 -15 0		
	-14 -15 0		
	13 14 15 0		
+-----+-----+-----+-----+			
1	2	3	4

Optimization of the translation

Several optimizations are needed to make the translation work in practice.

1. Reduce size of the SAT instance.
2. Reduce time for computing the SAT instance.
3. Instantiate rules (**inline**).
4. Eliminate predicates.

The first two techniques are now briefly explained.

Optimization 1: Size of the SAT instance

Problem: given the rule:

$$p(X, Y, Z) \leftarrow q(X, Y), s(Y, Z, W), r(Z, W).$$

a clause is generated for each of the $|U|^4$ ground instantiations.

Unpractical when the Herbrand universe is large, e.g., $|U| > 100$.

Proposed solution: notion of “alive” for ground instantiations of predicates. *alive*(p) (a subset of the Herbrand base) is:

- if p is a primitive predicate, *alive*(p) is the set of the ground instantiations corresponding to variables of the kinds α and β ;
- if p is a defined predicate, *alive*(p) is recursively defined as the set of atoms occurring in the head of ground instances of rules with p in the head, such that all positive literals in the body are alive.

Optimization 2: Time for computing the SAT instance

Problem: given the rule:

$$p(X, Y, Z) \leftarrow q(X, Y), s(Y, Z, W), r(Z, W).$$

if:

- Z and W have already been assigned to, e.g., 2 and 7, and
- $r(2, 7) \notin \text{alive}(\mathbf{r})$,

then it is useless to consider assignments to the other variables X and Y .

Proposed solution: The algorithm considers *partial* assignments to sets of variables, with a backtracking-based algorithm that for each rule explores a tree of depth n (one level for each variable).

Procedural aspects of translation

NPSPEC has two constructs for constraint programming, i.e., for providing the translation algorithm with procedural information to be used during the instantiation phase:

- **inline**: instantiates rules, generally beneficial,
- **multivalued**: disallows generation of *disjontness* constraints for **IntFunc** and **Partition** search spaces. It is a sound technique when such constraints are *safe-delay* [C & Mancini, 2004].

Outline

- Technologies for solvers
- The NP_{SPEC} project
- SPEC2SAT: system & algorithm
- Evaluation on the CSPLib
- Conclusions, current work

Experimental evaluation: starting point

- So far [C & Schaerf, 2001], limited experimentation (coloring, jobshop scheduling, Hamiltonian path, sport scheduling, university timetabling).
- Compilation faster than SAT for some problems, e.g., Hamiltonian path.
- Compilation time high for some problems, e.g., coloring, when:
 1. CNF is quite large, and
 2. has many models
- Overhead: SAT on SAT (specification of SAT, translation into SAT, solving SAT): 2.5 times slower than running a SAT solver on the original SAT instance.

Experimental evaluation: this paper

- Systematic experimentation on the CSPLib
- `www.csplib.org`: public repository of specifications:
 - 7 areas
 - 45 problems (mostly natural language)
- Current state of the experimentation in NPSPEC:
 - 6 areas covered
 - 12 NPSPEC specifications on the WWW (8 evaluated in this paper)
 - * “basic” model,
 - * “enhanced” model (symmetry breaking, delayed constraints)

NPSPEC on CSPLib: results

- Easy to specify problems (comparable to OPL)
- Approach works, on hundreds of instances
- Easy to include CP techniques at the specification level
- CP techniques typically effective
- Both complete (**zchaff**) and local search (**bg-walksat**) SAT solvers effective

Example: Balanced Academic Curriculum Problem (CSP030)

The BACP is to design a balanced academic curriculum by assigning periods to courses in a way that the academic load of each period is balanced, i.e., as similar as possible. The curriculum must obey the following administrative and academic regulations:

Academic curriculum: an academic curriculum is defined by a set of courses and a set of prerequisite relationships among them.

Number of periods: courses must be assigned within a maximum number of academic periods.

Academic load: each course has associated a number of credits or units that represent the academic effort required to successfully follow it.

Prerequisites: some courses can have other courses as prerequisites.

Minimum academic load: a minimum amount of academic credits per period is required to consider a student as full time.

Maximum academic load: a maximum amount of academic credits per period is allowed in order to avoid overload.

Minimum number of courses: a minimum number of courses per period is required to consider a student as full time.

Maximum number of courses: a maximum number of courses per period is allowed in order to avoid overload.

The goal is to assign a period to every course in a way that the minimum and maximum academic load for each period, the minimum and maximum number of courses for each period, and the prerequisite relationships are satisfied. An optimal balanced curriculum minimises the maximum academic load for all periods. Note that we could consider other types of balance criterion such as minimising the sum of the academic load of all periods.

Balanced Academic Curriculum Problem in NPSPEC

Partition(load, curriculum, PERIODS).

```
fail <-- SUM(curriculum(*,*,P), X:0..MAX_LOAD),
        period(P,M,_,_,_), X < M.
```

```
fail <-- SUM(curriculum(*,*,P), X:0..MAX_LOAD),
        period(P,_,M,_,_), X > M.
```

```
fail <-- COUNT(curriculum(*,*,P), X:0..MAX_LOAD),
        period(P,_,_,M,_), X < M.
```

```
fail <-- COUNT(curriculum(*,*,P), X:0..MAX_LOAD),
        period(P,_,_,_,M), X > M.
```

```
fail <-- prerequisite(Pre, Post), curriculum(Pre,_,P),
        curriculum(Post,_,Q), Q <= P.
```

Results with zchaff

Problem name	zchaff					
	Instances			SAT compil time (sec)	SAT solving time (sec)	Total time (sec)
	nr.	solved	unsolved			
Golomb Ruler	34	34	0	39,412.96	2.46	39,415.42
with safe delay	34	34	0	26,654.29	27.66	26,681.95
All-Interval Series	14	13	1	6.29	6,600.70	6,606.99
Social Golfer	168	110	58	64,467.93	212,527.78	276,995.71
with symm breaking	168	162	6	62,774.72	3,782.73	66,567.45
Schur's Lemma	164	164	0	2,412.57	0.08	2,412.65
with safe delay	164	164	0	2,510.13	0.12	2,510.12
with symm breaking	164	164	0	2,537.14	0.08	2,537.22
Ramsey problem	85	82	3	155.24	10,803.04	10,958.28
with safe delay	85	82	3	153.95	10,802.61	10,956.56
with symm breaking	85	82	3	9,099.76	484.017	9,583.78
Magic Square	3	3	0	281.16	128.59	409.75
with symm breaking	3	3	0	282.03	38.25	320.28
Langford's number	43	41	2	1,982.14	18,109.22	20,091.36
BACP	2	2	0	2,798.85	2.20	2,801.05

The “social golfer problem” (CSP010)

In a golf club, there are 32 (**Ngolfers**) social golfers, each of whom play golf once a week, and always in groups of 4 (**SizeGroup = Ngolfers / Nggroups**).

The coordinator would like you to come up with a schedule of play for these golfers, to last as many weeks as possible, such that no golfer plays in the same group as any other golfer on more than one occasion.

The best current solution is a 9 (**Nweeks**) week schedule.

Social golfer in NPSPEC

The basic model

SPECIFICATION

```
IntFunc({1..Ngolfers}><{1..Nweeks}, assignment, 1..Ngroups).
```

```
fail <-- assignment(G1,W1,Gr1), assignment(G2,W1,Gr1), G1 != G2,  
           assignment(G1,W2,Gr2), assignment(G2,W2,Gr2), W1 != W2.
```

```
fail <-- COUNT(assignment(*,W,Gr),X), X != SizeGroup.
```

Last constraint:

$$\forall W \in 1..Nweeks \quad \forall Gr \in 1..Ngroups \\ |\{G \in 1..Ngolfers \wedge assignment(G, W, Gr)\}| = SizeGroup$$

Some symmetries of the social golfer problem

Golfers, groups, and weeks can be permuted

- golfer 1 can play **every week** in group 1
- **first** week:
 - first group = (1,2,3,4)
 - second group = (5,6,7,8)
 - ...
 - eight group = (29,30,31,32)
- **second** week:
 - first group = (1,9,17,25)
 - second group = (2,10,18,26)
 - ...
 - eight group = (8,16,24,32)

Social golfer in NPSPEC: Symmetry breaking

```

fail <-- assignment(1,_,Gr), Gr != 1.
    // Golfer 1 must play every week in group 1

fail <-- assignment(G,1,Gr), Gr != ((G - 1)/SizeGroup) + 1.
    // First week groups:
    // (1,2,...,SizeGroup),
    // (1+SizeGroup,2+SizeGroup,...,SizeGroup*2),
    // ...

fail <-- assignment(G,2,Gr), Gr != ((G - 1) % Ngroups) + 1.
    // Second week groups:
    // (1,1+Ngroups,...,1+Ngroups*(SizeGroup-1)),
    // (2,2+Ngroups,...,2+Ngroups*(SizeGroup-1)),
    // ...

```

Outline

- Technologies for solvers
- The NPSPEC project
- SPEC2SAT: system & algorithm
- Evaluation on the CSPLib
- Conclusions, current work

Conclusions

- SAT approach for the execution of specifications of problems in NP.
- (Partial) evaluation on the CSPLib confirms that approach works.
- Further improvements of SAT solvers will reflect in an improvement of our system.

Current work

- Finish experimentation on the CSPLib, including complete specification of all problems.
- Compare with other approaches (OPL, ASP):
 - readability of specs
 - efficiency of method
- Incorporate Automated Reasoning techniques at the specification level, for:
 - detecting and breaking symmetries
 - detecting safe delay constraints