TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science | Institute of Artificial Intelligence | Knowledge Representation and Reasoning

PBLib — A C++ Toolkit for Encoding Pseudo–Boolean Constraints into CNF

Peter Steinke Norbert Manthey

KRR Report 14-01

Mail to
Technische Universität Dresden
01062 Dresden

Bulk mail to
Technische Universität Dresden
Helmholtzstr. 10
01069 Dresden

Office
Room 2006
Nöthnitzer Straße 46
01187 Dresden

Internet
http://www.wv.inf.tu-dresden.de

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

PBLib – A C++ Toolkit for Encoding Pseudo-Boolean Constraints into CNF

Peter Steinke and Norbert Manthey
peter.steinke@tu-dresden.de

October 30, 2013

Abstract

Many different encodings for PB constraints into conjunctive normal form (CNF) have been proposed in the past. The PBLib project starts to collect and implement these encodings to be able to encode PB constraints in a very simple, but effective way. The aim is not only to generate as few clauses as possible, but also using encodings that maintain generalized arc consistency by unit propagation, to speedup the run time of the SAT solver, solving the formula.

A major issue of the implementation is a high flexibility for the user. Consequently it is not required to bring a PB constraint into a certain normal form. The PBLib automatically normalizes the constraints and decides which encoder provides the most effective translation.

The user can also define constraints with two comparators (less equal and greater equal) and each PB constraint can be encoded in an incremental way: After an initial encoding it is possible to add a tighter bound with only a few additional clauses. This mechanism allows the user to develop SAT-based solvers for optimization problems with incremental strengthening and to keep the learned clauses for incremental SAT solver calls.

1 Implemented Encodings

Table 1 shows the encodings are currently implemented in the PBLib. The label *todo* denotes encodings that are planned for the (near) future.

2 Overview

The overview in Figure 1 shows the sequence of encoding a PB constraint. A PB constraint is given to the PB2CNF class, where the constraint is simplified and normalized by the PreEncoder. Next PB2CNF selects a suitable encoder for the simplified constraint. The clauses generated by the encoder are added to a clauses database and auxiliary variables are provided by an instance of the AuxVarManager class.

Figure 2 shows the encoding process of an incremental constraint. After this initial encoding the user can encode a tighter bound with the incremental data (without calling PB2CNF again), using the information that the initial bound is already in the clause database.

At most one	At most K	PB
sequential[9]*	BDD[6, 4]**	BDD
bimander[5]	cardinality networks[2]	adder networks
commander[7]	adder networks[4]	watchdog[8]
k-product[3]	*todo: perfect hashing*[10]	sorting networks[4]
binary[1]		
pairwaise		
nested		

* equivalent to BDD, latter and regular encoding
** equivalent to sequential counter

Table 1: Implemented encodings

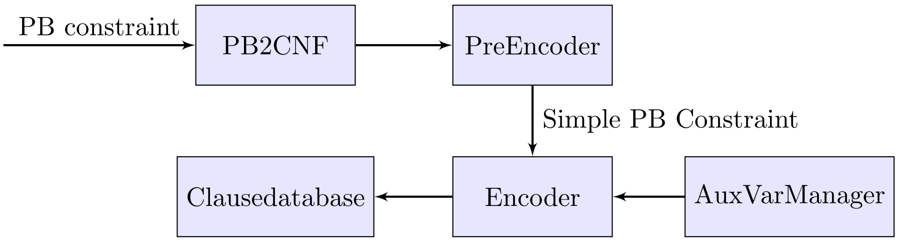

Figure 1: Encoding a PB constraint to CNF

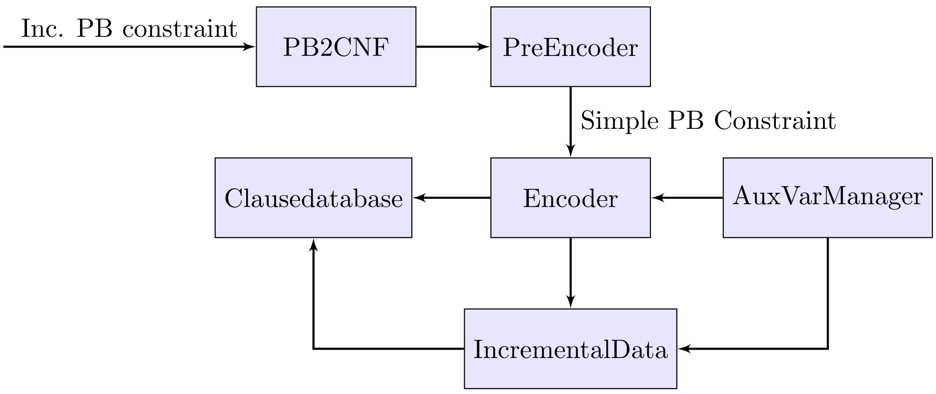

Figure 2: Encoding an incremental PB constraint to CNF

3 Using the PBLib

3.1 Including the PBLib

The following includes are needed for the PBLib:

```
#include "pb2cnf.h"       // encoding interface
#include "VectorClauseDatabase.h" // basic clause database
```

Optional includes are:

```
#include "PBParser.h"     // parser for opb files
#include "SATSolverClauseDatabase.h"
// a clause database that adds all clauses
// directly to a SAT solver (minisat)
```

The project has to be linked against the PBLib (libpblib.a or libpblib.so). Note that PBLib uses the `C++11` standard .

3.2 PB Constraints

For an instance of a PB constraint a vector of weighted literals is used:

WeightedLit (int32_t literal, int64_t weight);

In addition to the weighed literals a comparator is needed. This could be less equal, greater equal or a combination of both.

enum Comparator { LEQ, GEQ, BOTH };

Depending on the comparator one or two bounds are needed:

PBConstraint (vector< WeightedLit > const & literals, Comparator comparator, int64_t bound);

PBConstraint (vector< WeightedLit > const & literals, Comparator comparator, int64_t less_eq, int64_t greater_eq);
`// in case comparator == BOTH and less\_eq != greater\_eq`

3.3 Incremental PB Constraints

In addition to the PB constraints presented above, the incremental version allows an incremental encoding of a sequence of tighter bounds. Note that an incremental PB constraint is not a subclass of PB constraint (in the sense of object-oriented programming).

After the initial encoding of an incremental PB constraint *incPbConstraint*, it is possible to encode new (tighter) bounds with the methods:

incPbConstraint.encodeNewGeq (int64_t newGeq, ClauseDatabase & formula, AuxVarManager & auxVars);

incPbConstraint.encodeNewLeq (int64_t newLeq, ClauseDatabase & formula, AuxVarManager & auxVars);

The clause database *formula* has to contain the initial encoding of *incPbConstraint*.

3.4 Clause Database

The clause database is a container class for clauses – the CNF formula. Every implementation of this class provides the following methods:

addClause (vector<int32_t> const & clause);

The PBLib contains one default implementation of this class: VectorClauseDatabase. Every clause that is added is saved into a vector of clauses.

3.5 Auxiliary Variable Manager

The Auxiliary Variable Manager, returns fresh variables to the encoder. Therefor it is initialized with the first fresh variable. Hence it is assumed that all variables in the original constraints are smaller than this first fresh variable.

AuxVarManager (int32_t first_free_variable);

Every `getVariable` call returns the successor variable of the last returned variable. With the method **int32_t getBiggestReturnedAuxVar()** the biggest returned variable can be obtained. Hence every variable between this (including) number and *first_free_variable* (probably) occurs in some clause database.

3.6 Encoding PB constraints into CNF

The class *PB2CNF* is the main interface to encode an (incremental) PB constraint.

PB2CNF(PBConfig & config);

Where *config* is a configuration object explained in the next subsection. Besides the constraint, a clause database and an auxiliary variable manager is needed to encode a PB constraint with the *encode* method:

encode(PBConstraint const & pbconstraint, ClauseDatabase & formula, AuxVarManager & auxVars);

encodeIncInital(IncPBConstraint & incPbconstraint, ClauseDatabase & formula, AuxVarManager & auxVars);

3.7 Configuration

An instance of the *PBConfig* class contains the configuration for all options in the PBLib. Since PBConfig is a shared pointer you have to initialize it in the following way:

```
PBConfig config = make_shared<PBConfigClass>();
```

The following options (with the given default values) are currently available:

```
PB2CNF_PB_Encoder pb_encoder = PB_ENCODER::BEST;
PB2CNF_AMK_Encoder amk_encoder = AMK_ENCODER::BEST;
PB2CNF_AMO_Encoder amo_encoder = AMO_ENCODER::BEST;
BIMANDER_M_IS bimander_m_is = BIMANDER_M_IS::N_HALF;
int bimander_m = 3;
```

```
int k_product_minimum_lit_count_for_splitting = 10;
int k_product_k = 2;
int commander_encoding_k = 3;
int64_t MAX_CLAUSES_PER_CONSTRAINT = 1000000;
bool use_formula_cache = false;
bool print_used_encodings = false;
bool check_for_dup_literals = false;
bool use_gac_local_watch_dog = true;

enum PB2CNF_AMO_Encoder {BEST, NESTED, BDD,
                         BIMANDER, COMMANDER,
                         KPRODUCT, BINARY, PAIRWAISE};
enum PB2CNF_AMK_Encoder {BEST, BDD, CARD};
enum PB2CNF_PB_Encoder  {BEST, BDD, WATCHDOG,
                         SORTINGNETWORKS, ADDER};
enum BIMANDER_M_IS      {N_HALF, N_SQRT, FIXED};
```

Note that if the maximum number of clauses per constraint is (approximately) exceeded, the adder encoding is used as a fallback.

4 Example PB Solver

The PBLib source code contains also a folder *BasicPBSolver* with the implementations of an example PB Solver. It uses minisat 2.2 as a back-end SAT solver and solves optimization instances with the help of incremental PB constraints. Hence minisat keeps the learned clauses during each search iteration and only a few clauses are added after the initial solving run.

```
usage ./pbsolver inputfile
```

5 PB Fuzzer

Included is also a PB fuzzer. The program generates a random PB problem that is solved by two different PB Solvers and compares the results. If the results differ, an error is reported and the generated (error) instance is saved in the "tmp_fuzzing.opb" file.

This program helps to find bugs in new or customized implementations of the PBLib.

```
usage: ./fuzzer solver1 solver2
```

References

[1] Anthony J. Doggett Alan M. Frisch, Timothy J. Peugniez and Peter W. Nightingale. Solving Non-Boolean Satisfiability Problems with Stochastic Local Search: A Comparison of Encodings. *Journal of Automated Reasoning*, 35, 2005.

[2] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality Networks and Their Applications. Lecture Notes in Computer Science. Springer, 2009.

[3] Jingchao Chen. A new sat encoding of the at-most-one constraint. In *The Twelfth International Workshop on Constraint Modelling and Reformulation*, 2010.

[4] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2, 2006.

[5] Steffen Hölldobler and Van-Hau Nguyen. On SAT-Encodings of the At-Most-One Constraint. In *The Twelfth International Workshop on Constraint Modelling and Reformulation*, 2013.

[6] Albert Oliveras Ignasi Abío, Robert Nieuwenhuis and Enric Rodríguez-Carbonell. BDDs for Pseudo-Boolean Constraints – Revisited. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011.

[7] Will Klieber and Gihwon Kwon. Efficient CNF Encoding for Selecting 1 from N Objects. In *The fourth Workshop on Constraints in Formal Verification*, 2007.

[8] Yacine Boufkhad Olivier Bailleux and Olivier Roussel. New Encodings of Pseudo-Boolean Constraints into CNF. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009.

[9] Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming – CP 2005*, 2005.

[10] Oded Margalit Yael Ben-Haim, Alexander Ivrii and Arie Matsliah. Perfect Hashing and CNF Encodings of Cardinality Constraints. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012.