

Abstract

This is the abstract

Contents

1	Acknowledgements	1
2	Introduction	2
2.1	Constraint Programming	2
2.2	Gecode	2
3	Background	2
3.1	The need for tuple variables	2
4	Tuple Variable	2
4.1	Creating a new Variable	2
4.1.1	Specification file	2
4.1.2	Variable implementation	3
4.1.3	Deltas	3
4.1.4	Views	3
4.1.5	Propagators	3
4.1.6	Branchers	3
4.2	Implementation of exact IntPair	3
4.3	Implementation of rectangle IntPair	3
4.4	Performance test	4
5	Related work	4
5.1	Implementation of bit-vector variables in a CP solver	4
5.2	A propagator design framework for constraints over sequences	4
6	Conclusions and future work	4
6.1	Write the implementation in such a way that it may be accepted in a new Gecode release	4
6.2	Test different implementations with different domain representations	4
6.3	Extend the tuples to arbitrary dimensions and not just pairs	4
6.4	Reach better performance for a specific problem with an array of pair variables than with two arrays of integer variables	4
7	Git repository	4
8	Legal	4

1 Acknowledgements

Thanks to Joseph Scott

2 Introduction

2.1 Constraint Programming

Constraint programming (CP) is a method to solve problems by first modeling the problem with different types of variables and then adding constraints that the variables must fulfill. The variables have an initial domain of possible values. The CP solver then uses the constraints to remove impossible values. A very typical problem suitable for constraint programming is sudoku. The most intuitive model is a 9x9 matrix of integer variables. The general constraints for all standard sudokus is that every cell has a domain of 1..9, and that all cells in a row, column and box must be different. For a specific sudoku constraints are added to require specific cells to have specific values.

The available data types limits the options to model a specific problem. There is a famous problem called Queens. The problem is to place eight queens on a chess board in such a way that no queen threatens another. If only boolean variables were available we would have to model the problem with 64 boolean variables. If we instead use integer variables we can use the fact that all rows will have exactly one queen when we model the problem.

2.2 Gecode

Gecode is an open source CP solver blablabla.

3 Background

3.1 The need for tuple variables

When using a CP solver, one wants to avoid the brute force searching. This is done by intelligently choosing model and constraints. By doing this the variable domains can be pruned to smaller sizes. For example, if two variables x and y have domains $x = \{1, 2, 3\}$ $y = \{2, 3, 4\}$ and the constraint $y < x$ is added we can prune the domains to $x = \{3\}$ and $y = \{2\}$. If we instead have the same variables but add the constraint $x \neq y$ nothing would be pruned, since for all values in both domains there exists solutions. In this example the problem was modeled with two integer variables, but if the problem instead would be modeled with one tuple variable pruning would be possible. Then we would have a tuple variable t with domain $t = \{< 1, 2 >, < 1, 3 >, < 1, 4 >, < 2, 2 >, < 2, 3 >, < 2, 4 >, < 3, 2 >, < 3, 3 >, < 3, 4 >\}$ and pruning with the distinct condition would yield $t = \{< 1, 2 >, < 1, 3 >, < 1, 4 >, < 2, 3 >, < 2, 4 >, < 3, 2 >, < 3, 4 >\}$. This shrinks the size of the search tree from 9 to 7.

If we instead look at the case where $x, y = \{1..1000\}$ and the constraint $x = y$ the benefits is more obvious. Using integer variables pruning can not remove anything. If tuple variables were used, the size of the domain would shrink from 10^6 to 10^3 . This indicates that tuple variables can be quite useful.

4 Tuple Variable

4.1 Creating a new Variable

4.1.1 Specification file

Creating the specification file is the first thing that needs to be done when designing a new variable. Caution should be taken, because this step may be very hard to modify later. A simple specification file could look like this.

```

[ General ]
Name:      IntTuple
Namespace: MPG::IntTuple

[ ModEvent ]
Name:      FAILED=FAILED

[ ModEvent ]
Name:      VAL=ASSIGNED

[ ModEvent ]
Name:      NONE=NONE

[ PropCond ]
Name:      NONE=NONE
[ End ]

```

As can be seen, there are three sections. The specification file must start with the [General] section and end with [End]. There are also sections for modification events and propagation conditions. When the specification file is written a configuration script is used to generate base classes from the specification file.

The [General] section is pretty straight forward. This is where the name and namespace of the variable is specified. Both name and namespace is arbitrary. In Gecode the standard variables have namespace Gecode::VarName and in the examples in the documentation they instead have the namespace MPG::VarName.

The [ModEvent] section describes the modification events. For all variables modification events for NONE, FAILED and ASSIGNED are required.

The [PropCond] section describes how the propagators are scheduled.

4.1.2 Variable implementation

The variable implementation class would with the specification file in the previous section be called IntTupleImp and inherit from the generated class IntTupleVarBase. It has to implement a function called assigned that returns true if the variable is assigned.

4.1.3 Deltas

4.1.4 Views

4.1.5 Propagators

4.1.6 Branchers

4.2 Implementation of exact IntPair

The exact version does not approximate the domain.

4.3 Implementation of rectangle IntPair

This version of IntPair approximates the domain by just storing four pairs. The benefit is vastly less use of memory, but the drawback is that it can't be pruned as efficient as the exact version.

4.4 Performance test

5 Related work

5.1 Implementation of bit-vector variables in a CP solver

5.2 A propagator design framework for constraints over sequences

6 Conclusions and future work

6.1 Write the implementation in such a way that it may be accepted in a new Gecode release

6.2 Test different implementations with different domain representations

6.3 Extend the tuples to arbitrary dimensions and not just pairs

6.4 Reach better performance for a specific problem with an array of pair variables than with two arrays of integer variables

7 Git repository

<http://github.com/patwotrik/gecode-tuples>

8 Legal

This document is licensed under GPL v3 and may be redistributed as GPL v3 or later.

References

- [1] Calculus, Adams
- [2] Operating systems and its concepts, Galvin