Implementing Tuple Variables in Gecode

Bachelor Thesis

Patrik Broman

Constraint Programming

- Declarative style programming
- The programmer states what must hold, without stating how it is achieved
- A constraint programming variable has more in common with mathematical variables than regular variables used in programming

Constraint Programming

```
while not done:
    propagators prune as much as possible
    a brancher assignes a value to a variable
```

Constraint Programming

Sudoku is a good example of a problem suitable for constraint programming.

Constraint Programming

```
IntVar s[9][9] = {1, … ,9}

for x=1 to 9:
   distinct(s[x][1], s[x][2], … , s[x][9])

for y=1 to 9:
   distinct(s[1][y], s[2][y], … , s[9][y])

for x=1 to 3:
   for y=1 to 3:
      distinct( s[3x-2][3y-2], s[3x-1][3y-2], s[3x][3y-2],
                s[3x-2][3y-1], s[3x-1][3y-1], s[3x][3y-1],
                s[3x-2][3y], s[3x-1][3y], s[3x][3y])
```

What data types exist in Gecode?

- Boolean
- Integer
- Integer set
- Float

What about Tuples?

What about Tuples?

Consider the following example:

```
Intvar a,b = {1, …, 1000}
rel(a ≠ b)
```

What about Tuples?

- Without tuples, it is impossible to prune a combination of values.
- Tuples move work from branchers to propagators.

Pair Variables

- We restrict tuples to 2-tuples and call them pairs
- Two variants are created
  - Exact domain representation
  - Approximate representation

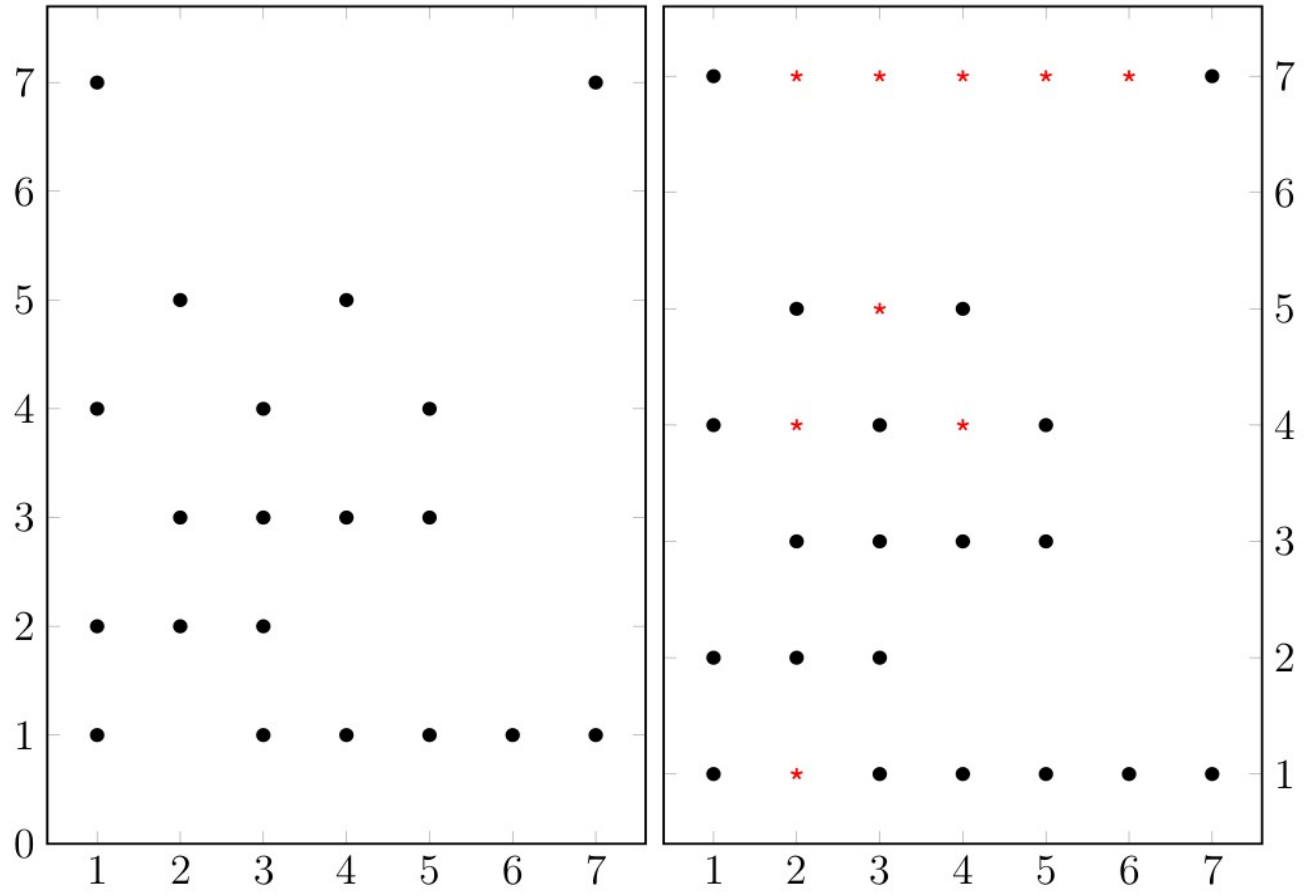
Exact Pair

- Stores a long list of ALL combinations
- Requires more memory
- Some operations are slower
- Can be pruned more efficiently

Approximate Pair

- Does not store all possible combinations
- Requires less memory
- Some operations are faster
- Can not be pruned as efficiently as the exact representation

Comparison



The cDFA constraint

- A cDFA is a DFA with individual costs for each transition
- A cDFA does not only tell if a string is accepted or not, but also the cost

The cDFA constraint

cDFA(ps, pc, qs, qc, x, S, C)

- qs: State before the transition
- qc: Cost before the transition
- x: Next symbol in the string
- S: State function
- C: Cost funtion
- ps=S(qs, x)
- pc=qc+C(qs, x)

The cDFA constraint

cDFA(P, Q, x, S, C)

- Q: State and cost before the transition
- x: Next symbol in the string
- S: State function
- C: Cost funtion
- P=<S(Q.s, x), Q.c + C(Q.s, x)>

Comparing performance

```
PairVar P[n+1]
IntVar x[n]
for i=1 to n:
  cDFA(P[n+1], P[n], x[n], S, C)
```

```
IntVar s[n+1], c[n+1], x[n+1]
for i=1 to n:
  cDFA(s[n+1], c[n+1], s[n], c[n],
       x[n], S, C)
```

Comparing performance

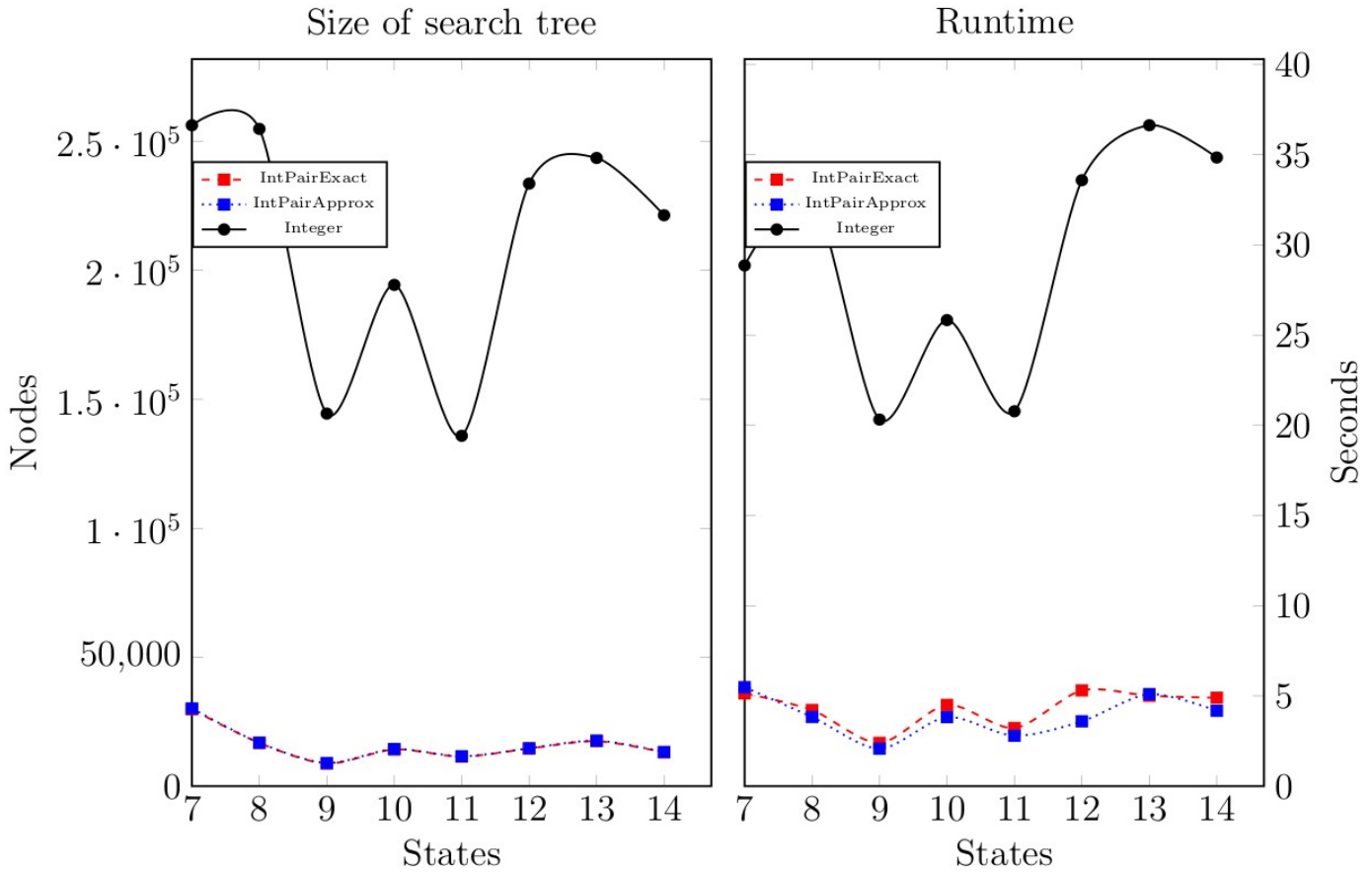
Performance is measured in two ways:

- Size of search tree
- Total runtime

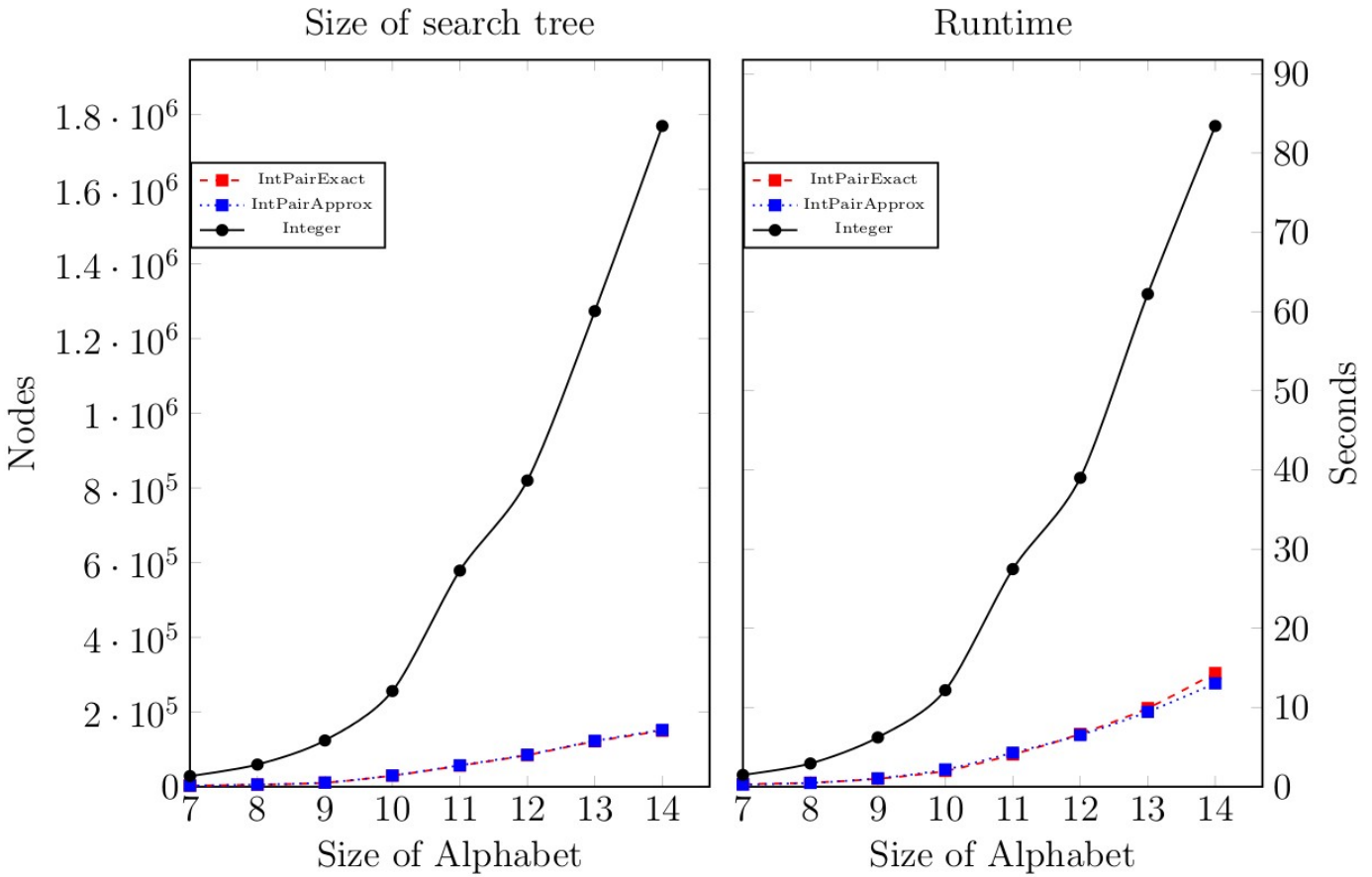
Performance is compared by varying four parameters:

- Number of states
- Size of alphabet
- Cost per transition
- Length of string

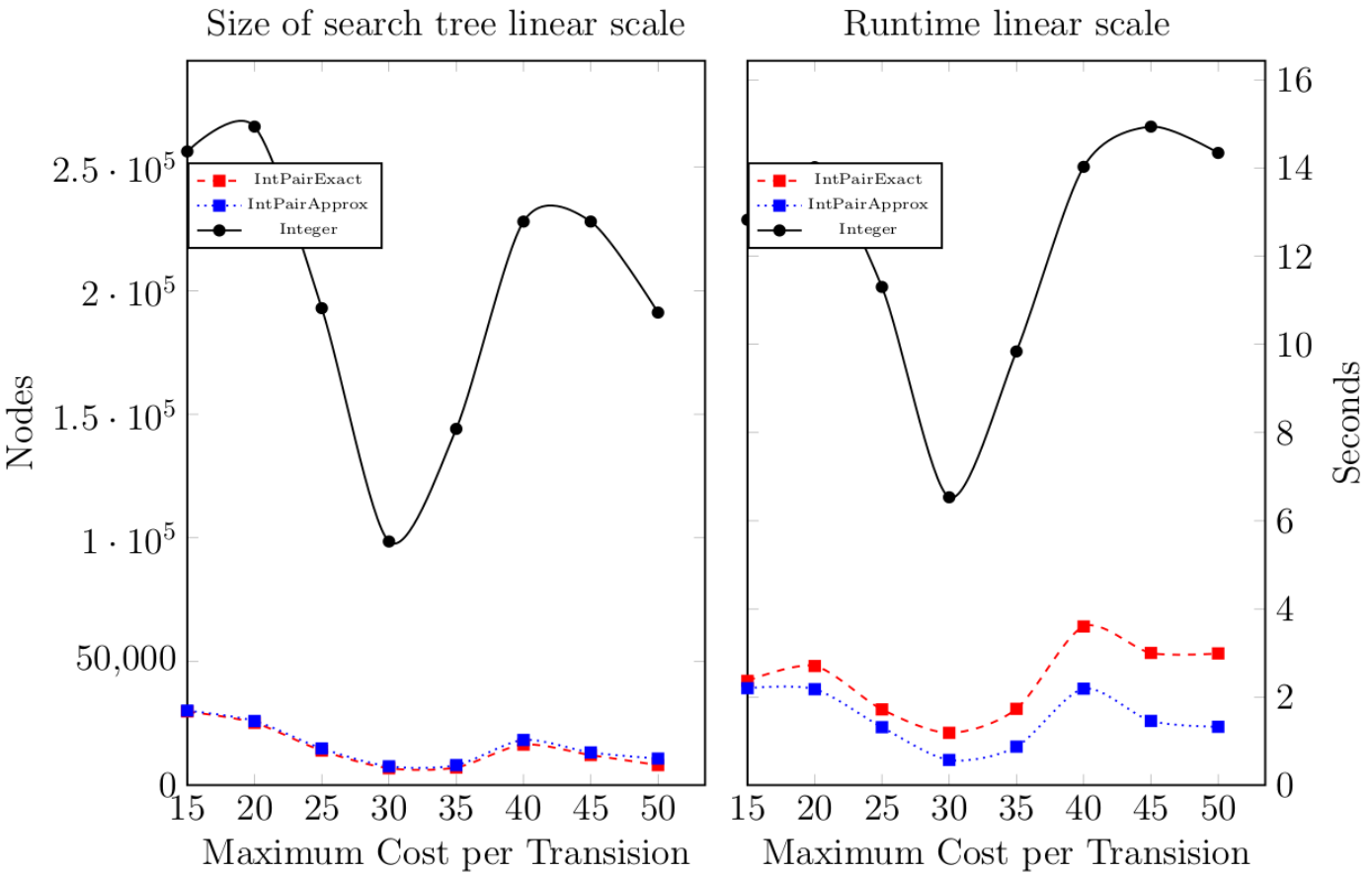
Varying number of states



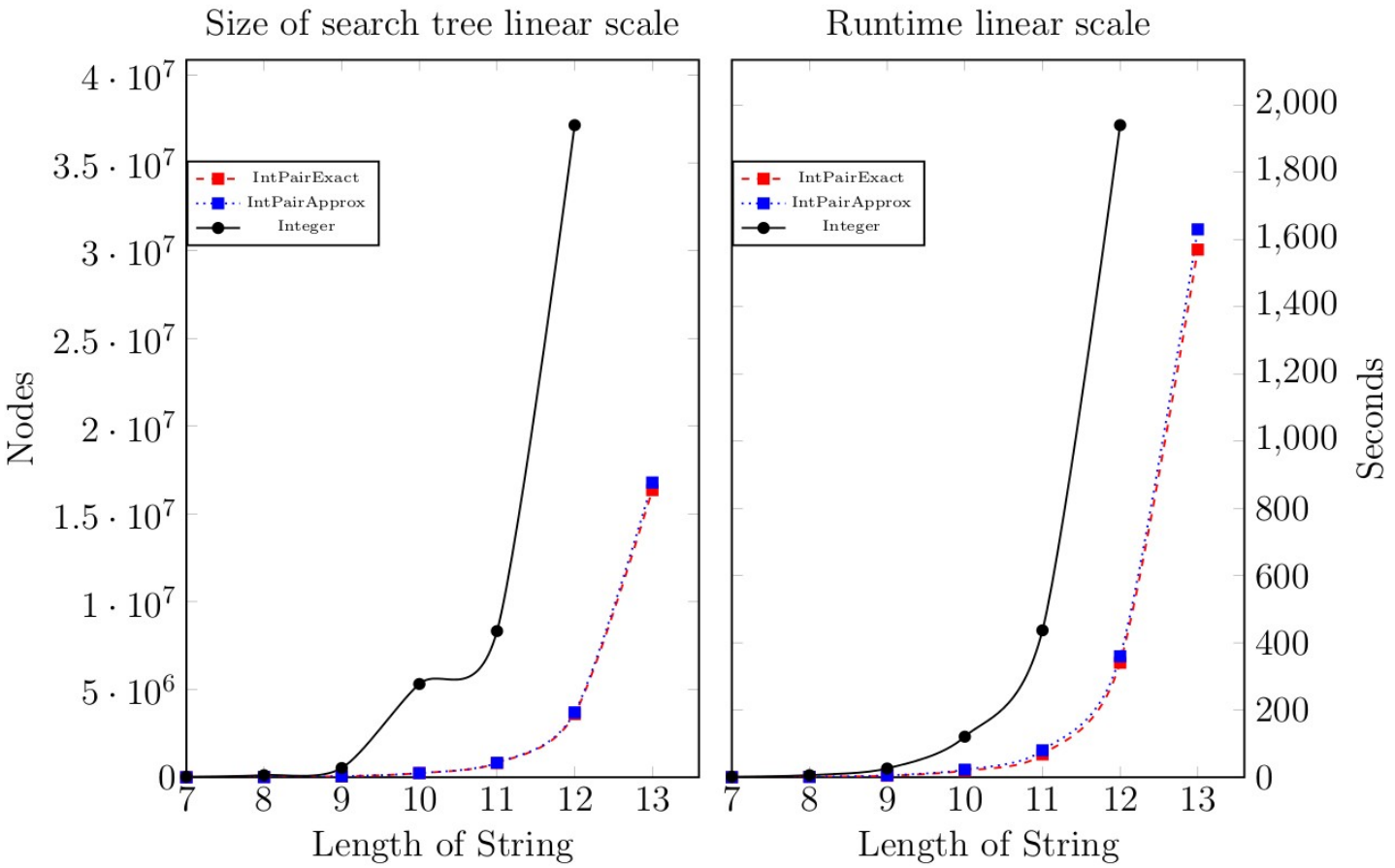
Varying size of alphabet



Varying cost per transition



Varying length of string



Possible Problems

The propagator for regular integer variables may be optimised further. However, much more work have been spent on optimising this.

Conclusions

Tuples seem very promising. In these tests, they perform far better than regular integer variables. This is true for both the size of search tree and the total execution time.