Implementing Tuple Variables in Gecode

UPPSALA UNIVERSITET

Bachelor Thesis

Patrik Broman

Gecode

- Library for constraint programming
- Open source – MIT licence
- Implemented in C++

UPPSALA UNIVERSITET

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 assigns 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 Tuple Variables?

- A tuple is an ordered list of elements. The elements can be of any type, and do not need to be of the same type.
- We restrict ourselves to 2-dimensional integer tuple variables.

UPPSALA UNIVERSITET

What about Tuple Variables?

Consider the following example:

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

What about Tuple Variables?

- Without tuples variables, it is impossible to prune a combination of values
- Tuples variables move work from searching to propagation

Pair Variables

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

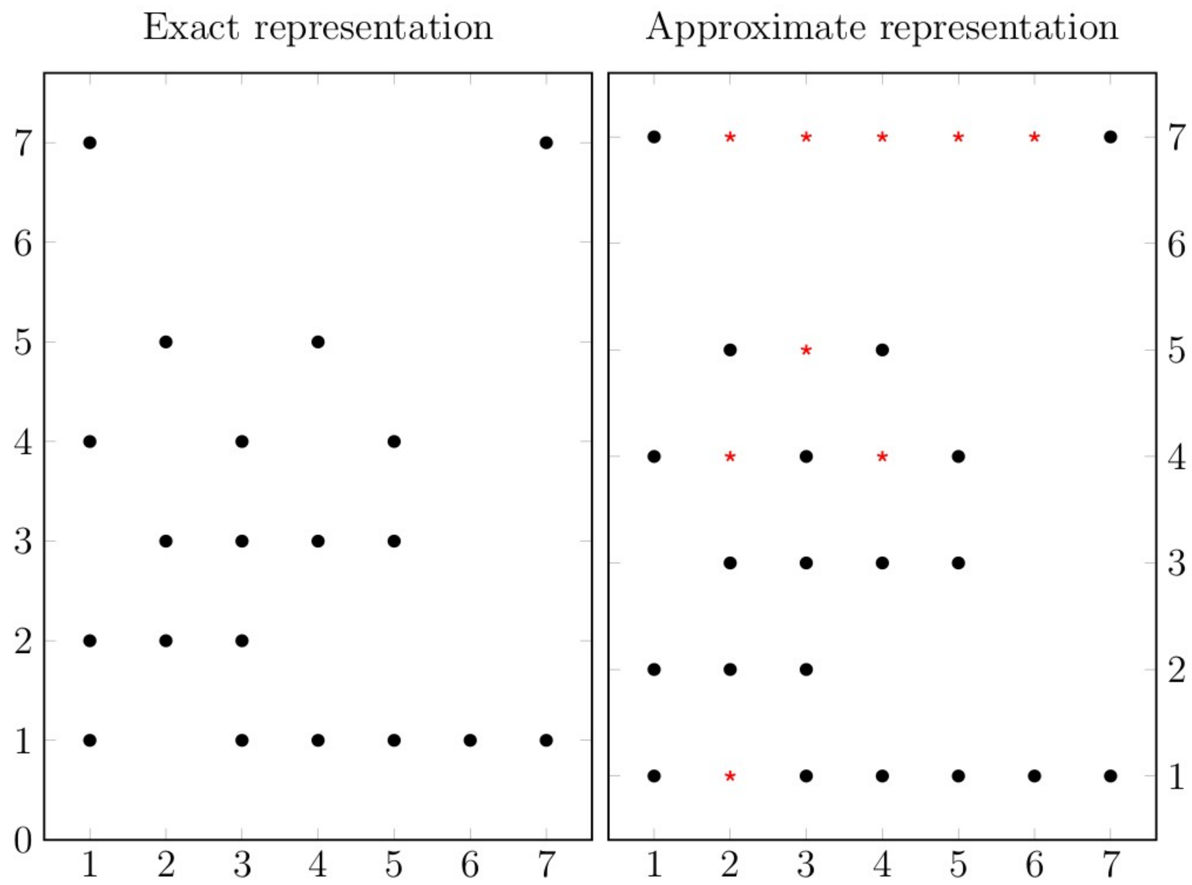
Exact Pair

- Stores a long list of ALL combinations
- Requires more memory
- Some operations are slower
- More values can be pruned

Approximate Pair

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

Comparison


Exact representation
Approximate representation
0
1
2
3
4
5
6
7

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

UPPSALA UNIVERSITET

The cDFA constraint

```
cDFA(ps, pc, qs, qc, x, S, C)
- qs: State before the transition
- qc: Cost before the transition
- ps: State after the transition
- qs: Cost after 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
- P: State and cost after 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

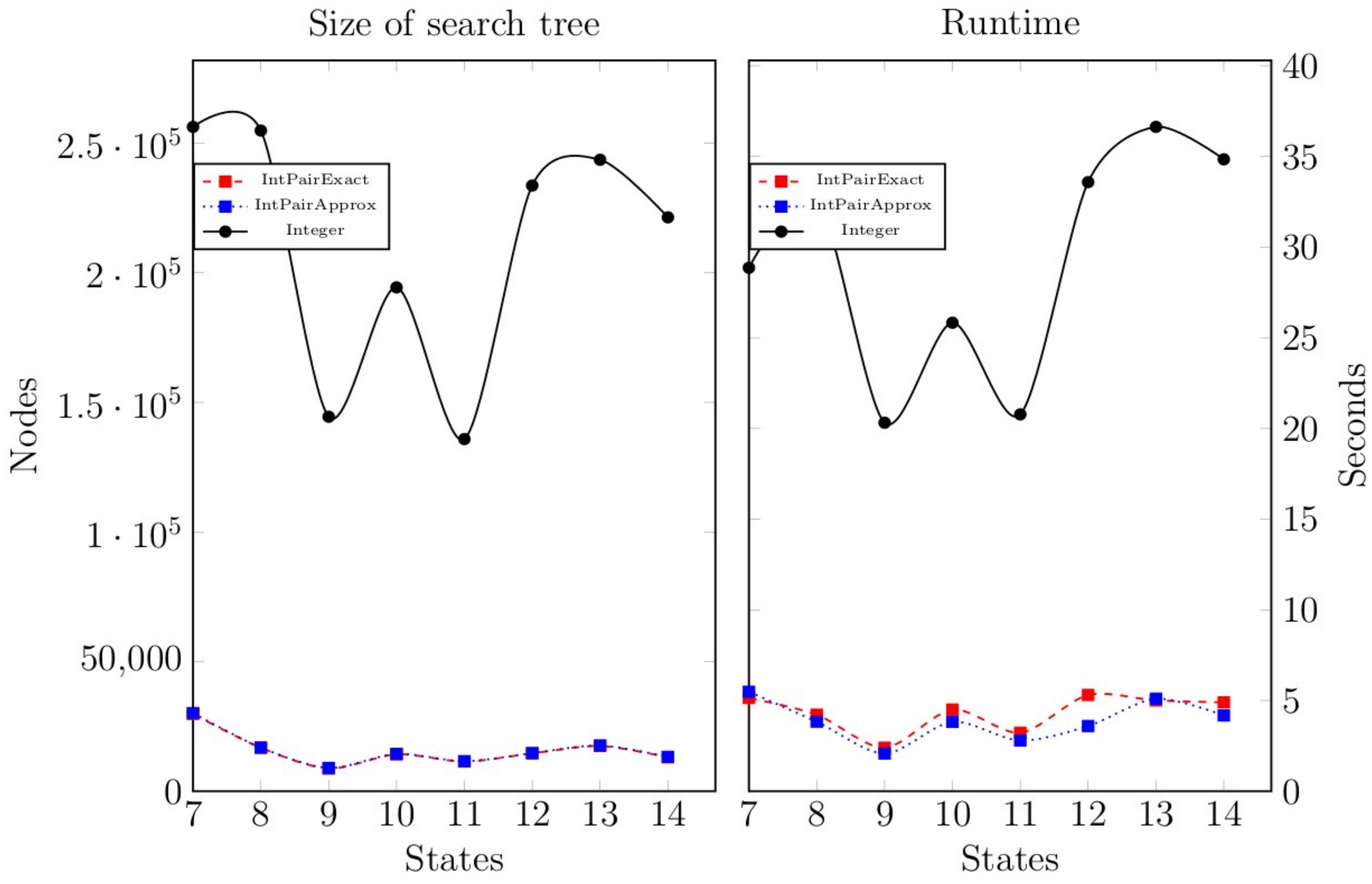
Performance is measured in two ways:

- Size of search tree
- Total runtime

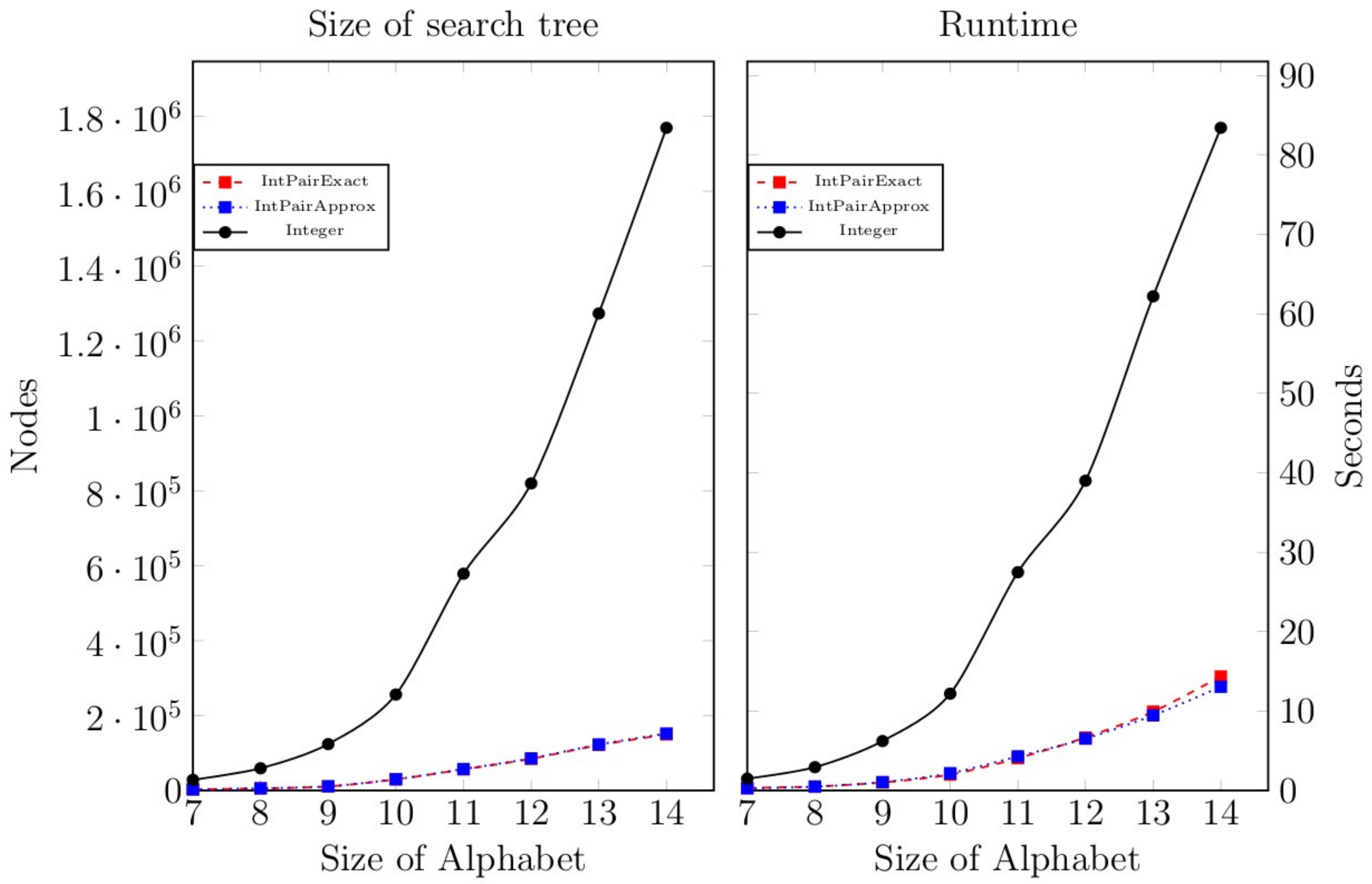
Performance is compared by varying four parameters:

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

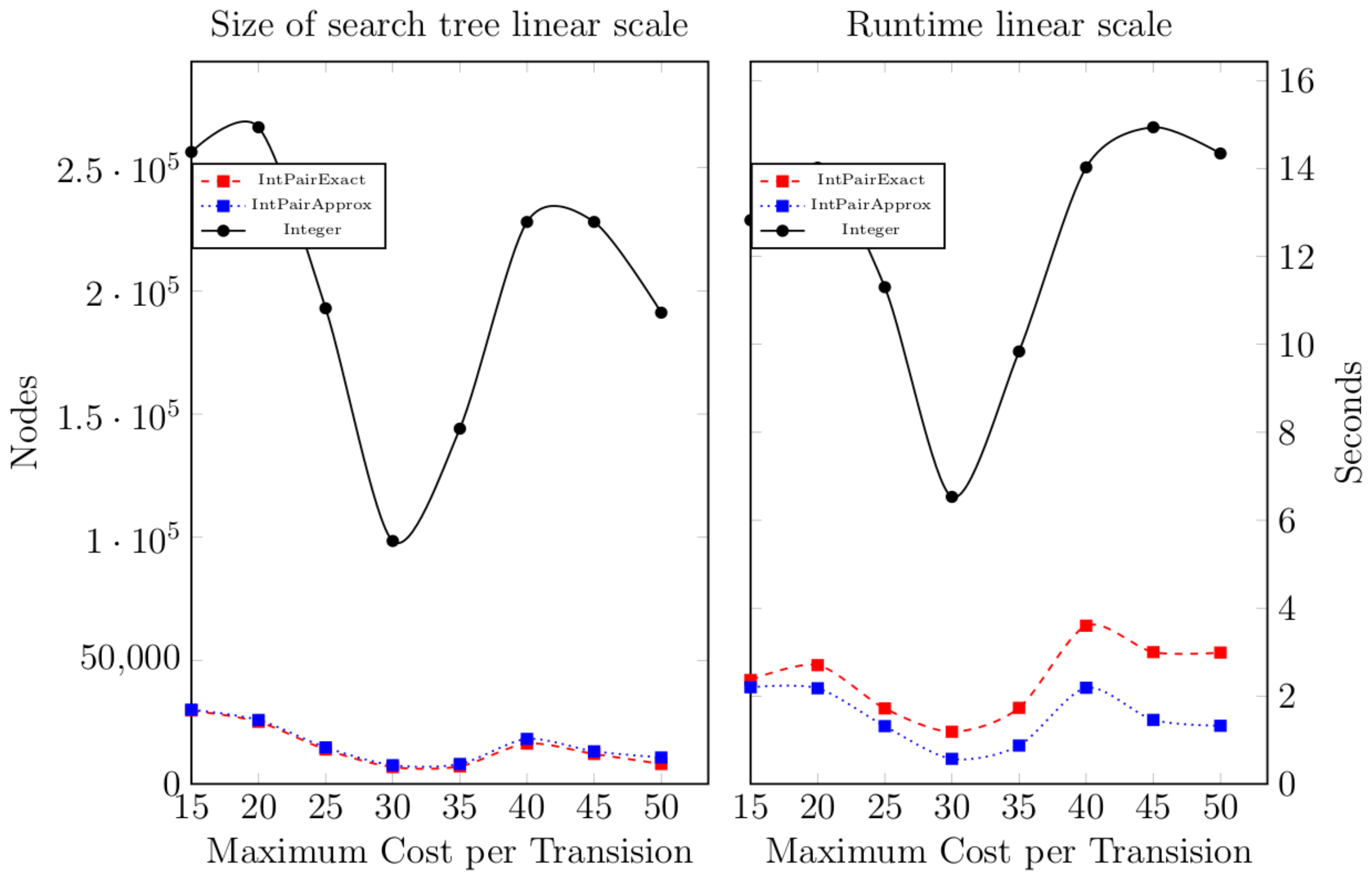
Varying number of states



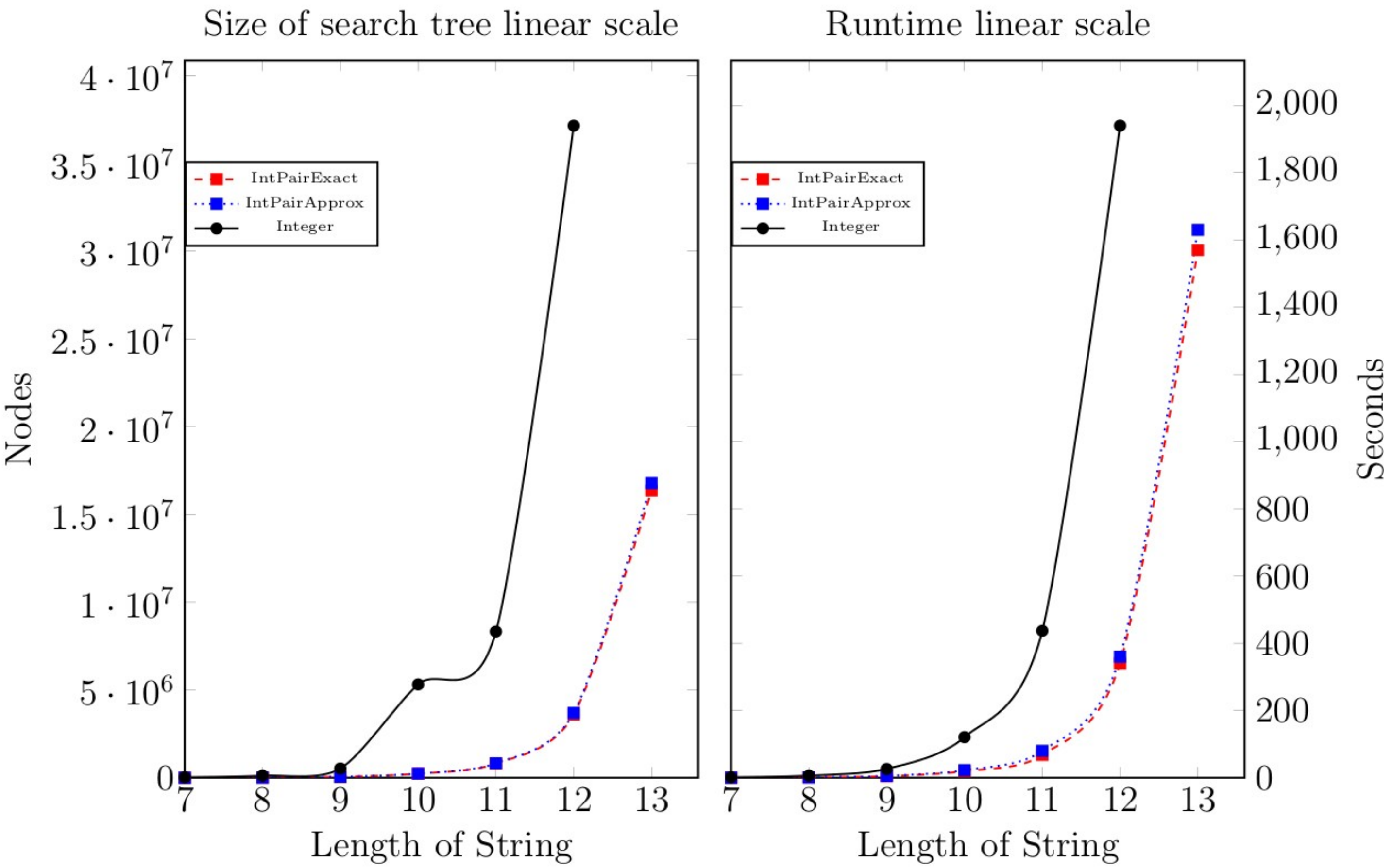
Varying size of alphabet



Varying cost per transition



Varying length of string



Conclusions

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