Towards a lightweight standard search language

Horst Samulowitz, Guido Tack, Julien Fischer, Mark Wallace, Peter Stuckey

ModRef 2010
September 6th 2010, St. Andrews, Scotland
Goals

• Define a **search language** for MiniZinc

• **Lightweight**: Balance expressiveness with ease of implementation

• Basis for discussion and (eventually) wide adoption
Why custom search?

• Standard labeling sometimes not good enough
• Exploit problem structure
  - problem decomposition
• Combine search procedures
  - restarts, warm starts, backdoors, portfolios...
Why standard language?

• **Compare** different solvers and searches
• **Exchange** models (e.g. CSPLib)
• **Communicate** search strategies (e.g. papers)

• **Fix good names**
  (independent of adoption as a standard!)
Approach

- **Not:** fully programmable search (too complex)
- Language for **combining** predefined search strategies
- Library of **search templates** that define the strategies
Simple labeling

Template:

\[ \text{varsel} \equiv \text{input_order, random_order, } \{\text{min, max}\}_{\{\text{lb, ub}\}}, \{\text{min, max}\}_{\text{dom size}}, \{\text{min, max}\}_{\text{dom size weighted degree}}, \ldots \]

\[ \text{domsplit} \equiv \{\text{assign, exclude}\}_{\{\text{lb, ub}\}}, \text{bisect}_{\{\text{low, high}\}}, \{\text{assign, exclude}\}_{\text{impact}_{\{\text{min, max}\}}}, \ldots \]
limit_search($measure, limit, search$)

fails, nodes, solutions, time, discrepancies

once($search$) $\equiv$ limit($solutions, 1, search$)
lds($d, search$) $\equiv$ limit($discrepancies, d, search$)

restart_geometric($inc, init, measure, search$)
restart_luby($init, max, measure, search$)
Composition

Sequential search:
seq_search([search1, ..., searchN])

Parallel search:
par_search([search1, ..., searchN])

Diagram:
- Sequential search: search1 and search2
- Parallel search: search1 | or | search2
Example: Job Shop

constraint
    forall(i in 1..size) ( 
        forall(j in 1..size-1) (s[i,j]+d[i,j] <= s[i,j+1]) 
        /
        s[i,size] + d[i,size] <= end 
        /
        forall(j,k in 1..size where j < k) ( 
            no_overlap(s[j,i], d[j,i], s[k,i], d[k,i])
        ) 
    );

solve :: search minimize end;
Example: Job Shop

Simple dom/wdeg search:

\[
\text{search} \equiv \text{int\_search}(s, \\
\quad \text{min\_dom\_size\_weighted\_degree,} \\
\quad \text{bisect\_low})
\]

Find first solution with LDS, then prove optimality with IBS:

\[
\text{search} \equiv \text{par\_search}( [ \\
\quad \text{lds}(3, \text{int\_search}(s, \text{min\_lb, assign\_lb})), \\
\quad \text{int\_search}(s, \text{max\_impact, assign\_impact\_min}) ])
\]
Example: Radiotherapy

\begin{verbatim}
var 0..Ints_sum: Beamtime;
var 0..m*n: K;
array[BTimes] of var 0..m*n: N;
array[Rows, Columns, BTimes] of var 0..m*n: Q;

constraint
    Beamtime = sum(b in BTimes) (b * N[b])
    K = sum(b in BTimes) (N[b])
    forall(i in Rows, j in Columns)
        ( Intensity[i,j] = sum([b * Q[i,j,b] | b in BTimes]) )
    forall(i in Rows, b in BTimes)
        ( ub_i(N[b], [Q[i,j,b] | j in Columns]) );

predicate ub_i(var int: N_b, array[int] of var int: L) =
    N_b >= L[1] + sum([ max(L[j] - L[j-1], 0) | j in 2..n ]); 

solve :: search minimize (ub(K) + 1) * Beamtime + K;
\end{verbatim}
**Observation:** after labeling the N, each row in the Q is independent.

\[ \text{search} \equiv \text{seq\_search} \]

- First search N
- If one row fails, backtrack into N

```
int\_search N
```

```
\text{int}\_search \ Q[1] \\
\text{int}\_search \ Q[2] \\
\text{int}\_search \ Q[3] \\
\text{int}\_search \ Q[4]
```
Observation: after labeling the N, each row in the Q is independent

search ≡ 
seq_search(
  [int_search(N, min_dom_size_weighted_degree, bisect_low)]
  ++
  [once(int_search(
    [Q[i,j,b] | j in Cols, b in BTimes],
    max_activity, bisect_activity_min)) | i in Rows])

if one row fails, backtrack into N

first search N
**Observation:** after labeling the N, each row in the Q is independent

```
search ≡
  seq_search(
      [int_search(N, min_dom_size_weighted_degree, bisect_low)] ++
      [once(int_search(
        [Q[i,j,b] | j in Cols, b in BTimes],
        max_activity, bisect_activity_min))
      | i in Rows])
```
Implementation

• Two prototypes for FlatZinc/Gecode
  - code generator
  - C++ library

• Many templates implemented

• Generic approach, (hopefully) easy to adapt to other CP solvers
Future work

- Full implementation
- Define interaction with concurrent search
- Symmetry breaking?
- Shaving?
- Local search?
Conclusions

• Combinators and templates are expressive enough for useful, complex custom searches

• Proposed language can be implemented

• Useful as a standard: compare, exchange, communicate search strategies

• Independent of concrete modeling language: let's fix good names