On Translating MiniZinc Constraint Models into Fitness Functions for Evolutionary Algorithms: Application to Continuous Placement Problems

Thierry Martinez and François Fages

Inria Paris-Rocquencourt, Team Lifeware, France

Abstract. MiniZinc is a solver-independent constraint modeling language which is increasingly used in the constraint programming community. It can be used to compare different solvers which are currently based on either constraint programming, Boolean satisfiability or mixed integer linear programming. In this paper we show how MiniZinc models can be compiled into fitness functions for evolutionary algorithms. More specifically, we describe the translation of FlatZinc models into fitness functions over the reals and their use in the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) solver. We illustrate this approach, and evaluate it, on the modeling and solving of complex shape continuous placement problems.

1 Introduction

MiniZinc [11] is a medium-level constraint modeling language. It is high-level enough to express most constraint problems easily, but low-level enough to be mapped onto existing solvers easily and consistently. It is increasingly used as a standard by the Constraint Programming community. FlatZinc is a low-level solver input language that is the target language for MiniZinc. It is designed to be easy to translate into the form required by a solver. Currently, there exist FlatZinc parsers for mixed integer linear programming, finite domain constraint programming and SAT solvers.

In [1], Björdal et al. present a constraint-based local search backend for MiniZinc and show that it produces competitive results on the 2010 MiniZinc challenge. There has been related work on the design of high-level constraint-based modeling languages for local search and genetic algorithms. In particular the seminal work of Van Hentenryck and Michel on Comet [9,5] showed how a finite domain constraint model can be compiled into an objective function for local search metaheuristics, such as Tabu search, with default neighborhoods derived from the constraint model. In these systems, the local search solver is limited to finite domain constraints and use neighborhoods derived from the finite domains of the variables.

In this paper, we generalize this approach to constraints over real valued variables and show how the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) solver [4] can be used as backend for MiniZinc for both finite domain

and real-valued variable constraint satisfaction problems. More specifically, we show how a FlatZinc model can be translated into a fitness function for CMA-ES, and illustrate this approach by the modeling and solving of complex continuous placement problems for square and curved shapes.

In [8], we have shown that the non-overlap constraint between squares, cubes, rectangles, boxes, triangles, polygons circles and spheres, can be associated with a *measure of overlap* which can be used as a fitness function in CMA-ES for packing mixed shapes in a bin of minimal area (or volume), with an interesting trade-off between generality and efficiency. On Korfs benchmark of discrete consecutive sizes square packing problems, for which the optimal costs are known up to 32 squares, we have shown that the solutions computed by CMA-ES are typically at 14% from the optimal cost, with a computation time comparable to the time used by the best exact methods for finding an optimal solution [6]. On a benchmark of consecutive sizes circle packing problems, CMA-ES finds solutions at 2% of the best known costs obtained by running the three global optimization methods reported in Castillo et al. [2]. In [10], Salas and Chabert show that the overlap measures which were defined in an *ad hoc* manner in [8], can be defined with a numerical algorithm that automatically measures the *penetration depth* of two objects of virtually any shape defined by conjunction and disjunction of non-linear inequalities.

In this paper, we give MiniZinc definitions for the penetration depths between polygons and circles, and use them to compute the fitness function associated to a FlatZinc model. The rest of the paper is organized as follows. In the next section we show how circle packing and general mixed shape continuous placement problems can be modeled in MiniZinc. More specifically, we use ClpZinc, a higher-level extension of MiniZinc with records and Horn clauses to ease modeling, including the expression of search strategies [7] and here the complex definition of penetration depths between objects. In Section 3 we describe the translation of a FlatZinc model into a fitness function over the reals, using simple formula for circles and Minkowski sums for the penetration depth between polygons [3]. Then in Section 4, we report on the performance results obtained through the compilation chain from ClpZinc, MiniZinc, FlatZinc to CMA-ES, on some complex shape packing problems of [8]. Finally, we conclude on the general perspective opened by this approach for compiling MiniZinc constraint models to stochastic continuous optimization solvers.

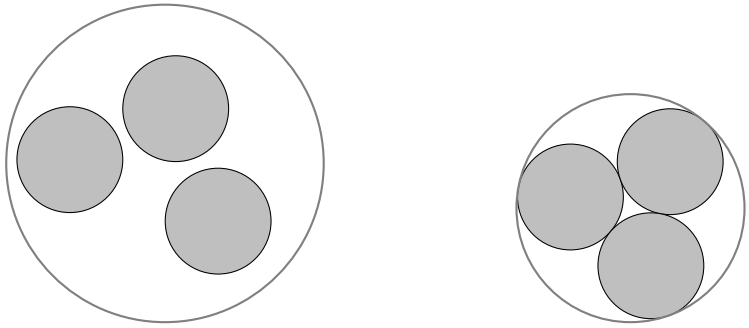
2 Modeling Continuous Placement Problems in MiniZinc

2.1 Packing Circles in MiniZinc

MiniZinc has support for float variables and is therefore a suitable language for modeling continuous problems such as continuous placement problems. Variables in a continuous placement problem are the positions of the object, i.e. (x, y) on a plane, (x, y, α) if we consider rotations, (x, y, z) in the space, *etc.*. For instance, the following MiniZinc model describes the problem of packing three circles of radius 1 in a circle of radius 3.

```
array[1..3] of var float: x;
array[1..3] of var float: y;
var float: radius;
constraint pow(x[2] - x[1], 2.0) + pow(y[2] - y[1], 2.0) > 4.0;
constraint pow(x[3] - x[1], 2.0) + pow(y[3] - y[1], 2.0) > 4.0;
constraint pow(x[3] - x[2], 2.0) + pow(y[3] - y[2], 2.0) > 4.0;
constraint pow(x[1], 2.0) + pow(y[1], 2.0) < pow(radius - 1.0, 2.0);
constraint pow(x[2], 2.0) + pow(y[2], 2.0) < pow(radius - 1.0, 2.0);
constraint pow(x[3], 2.0) + pow(y[3], 2.0) < pow(radius - 1.0, 2.0);
radius = 3;
solve satisfy;
```

$(\texttt{x[}i\texttt{]}, \texttt{y[}i\texttt{]})$ are the coordinates of the centers of the circles. The three first constraints impose that the three circles do not overlap pairwise. This is modeled by imposing that for each pair of circles, the (square of the) distance between their centers is greater than (the square of) the sum of their radii (i.e. $4 = (1+1)^2$). The last three constraints impose that all the three circles fit in the circle of radius 3 centered on the origin, *i.e.* for each circle, the (square of the) distance from the origin to the center of the circle is lesser than (the square of) 3 minus the radius of the circle (i.e. $4 = (3-1)^2$).



The bounding circle radius can also be specified as objective to minimize rather than given as input, by replacing the two last lines in the model by

```
constraint radius>0;
solve minimize(radius);
```

2.2 Packing Complex Shapes in ClpZinc

The previous example has in fact been automatically generated by ClpZinc from the following model.

```
include "packing.plz";

array [1..3] of var float: x;
array [1..3] of var float: y;
var float: radius;

:- non_overlap([circle(x[i], y[i], 1.0) | i in 1..3]),
   bounding_circle_lt([circle(x[i], y[i], 1.0) | i in 1..3], radius),
   radius > 0,
   minimize(radius).
```

ClpZinc[1] extends MiniZinc with records and the possibility to define search strategies [7] and relations by (terminating) Horn clauses. ClpZinc models are expanded into MiniZinc models and can thus benefit from the compilation to FlatZinc and a variety of constraint solvers. The ClpZinc packing library implements

- `non_overlap(Shapes)` for stating that the shapes in the list `Shapes` do not overlap pairwise.
- `bounding_circle_lt(Shapes, Radius)` for stating that the shapes fit into a bounding circle centered on the origin with the given radius.
- `bounding_box(Shapes, X0, Y0, X1, Y1)` for stating that shapes fit into the given bounding box.

`Shapes` is a list of terms of the form

- `circle(X, Y, Radius)`,
- `triangle(X0, Y0, X1, Y1, X2, Y2)`,
- users can define their custom shapes from these two elementary shapes by adding a new clause to the predicate `shape(S, L)` that unifies `L` with a list of elementary shapes that compose `S`. *E.g.*, squares can be described as the combination of two triangles with the following clause.

```
shape(square(X, Y, Size, Angle), [
  triangle(
    X, Y,
    X + cos(Angle) * Size, Y + sin(Angle) * Size,
    X + cos(Angle) * Size + cos(Angle + pi / 2) * Size, Y + sin(Angle)
* Size + sin(Angle + pi / 2) * Size),
  triangle(
    X, Y,
    X + cos(Angle - pi / 2) * Size, Y + sin(Angle - pi / 2) * Size,
    X + cos(Angle) * Size + cos(Angle + pi / 2) * Size, Y + sin(Angle)
* Size + sin(Angle + pi / 2) * Size)]).
```

3 Compiling MiniZinc Models into Fitness Functions

In order to use a continuous optimization solver such as CMA-ES, a FlatZinc model needs be compiled into a fitness function over the reals. The fitness function associates a cost to the violation of each constraint of the FlatZinc model for any given valuation of the variables. The optimization procedure finds values for the variables which minimize the cost at least locally.

The FlatZinc model is first transformed to a pair (c, e) where c is a constraint expression over the following grammar, with the obvious semantics, and e is an expression carrying the objective.

$$c ::= c \wedge c \mid c \vee c \mid c \Rightarrow c \mid e < e \mid e \leq e \mid e = e \mid \text{let } x := e \text{ in } c$$

For instance, let us consider the model below.

[1] `http://lifeware.inria.fr/~tmartine/clp2zinc/`

```
var float: x;
var float: y;

constraint x + y > 4.0;
constraint x < 3.0;
constraint y < 2.0;

solve minimize x * y;
```

This model is transformed by the `mzn2fzn` compiler to the following FlatZinc model.

```
array [1..2] of float: X_INTRODUCED_2 = [-1.0,-1.0];
var float: x:: output_var;
var float: y:: output_var;
var float: X_INTRODUCED_3 ::var_is_introduced :: is_defined_var;
constraint float_lin_lt(X_INTRODUCED_2,[x,y],-4.0);
constraint float_lt(x,3.0);
constraint float_lt(y,2.0);
constraint float_times(x,y,X_INTRODUCED_3):: defines_var(X_INTRODUCED_3);
solve  minimize X_INTRODUCED_3;
```

We associate to this model the following constraint expression, which displays the conjunction of the three first constraints, together with the objective $\texttt{x} \times \texttt{y}$.

$$-.1.0 \times \texttt{x} + -1.0 \times \texttt{y} < -4.0 \wedge \texttt{x} < 3.0 \wedge \texttt{y} < 2.0$$

The constraint expression is then transformed into a violation degree inductively over the structure of the expression.

$$v(c \wedge c') = v(c) + v(c') \quad v(c \vee c') = \min(v(c), v(c')) \quad v(c \Rightarrow c') = \mathbf{1}_c \times v(c')$$

Each elementary hard constraint is associated to a specific cost c_i that turns it into a soft constraint. The cost of $e < e'$ and $e \leq e'$ is $\max(e - e', 0)$. The cost of $e = e'$ is $|e - e'|$. In the context of continuous packing, if the non-overlapping constraint is expressed in terms of penetration depth, the associated cost will naturally guide the continuous solver towards a better solution with less overlap [8,10].

A fitness function is then generated and compiled together with the CMA-ES optimization loop, in C. $\mathbf{1}_c \times v(c')$ is mapped to the production of conditional `if` block, for models that use reification.

The fitness function is expected to return a violation cost for the constraints. If there is no hard constraint, this cost is the value of the objective in case of minimization, or its opposite in case of maximization. We provide several methods for aggregating the costs that result from the transformation of hard constraints into soft constraints through MiniZinc annotations.

- weighted constraints: the default method, which may be explicited with the annotation `solve satisfy :: weighted;`. Individual constraint costs c_i are summed up with multiplicative coefficients: $\sum_i w_i c_i$. Weights can be tuned by marking constraints with the annotation `:: weight(w)`.
- fuzzy: the maximum of all constraint costs c_i is considered for minimization ($\min_i w_i c_i$).

– probabilistic: the product of all constraint costs c_i is considered for minimization ($\prod_i w_i c_i$).

It is worth noticing that these methods are a mere syntactic sugar for expressing the objective function: the modelling language is expressive enough for expressing the calculation of the cost directly into the objective function. For example, here is a reformulation of the "three circles" example as a pure minimization problem with no hard constraints, with a weight of 1000 for each hard constraint:

```
array[1..3] of var float: x;
array[1..3] of var float: y;
var float: radius;
solve minimize (radius
  + 1000 * max(0, 4.0 - pow(x[2] - x[1], 2.0) + pow(y[2] - y[1], 2.0))
  + 1000 * max(0, 4.0 - pow(x[3] - x[1], 2.0) + pow(y[3] - y[1], 2.0))
  + 1000 * max(0, 4.0 - pow(x[3] - x[2], 2.0) + pow(y[3] - y[2], 2.0))
  + 1000 * max(0, pow(x[1], 2.0) + pow(y[1], 2.0) - pow(radius - 1.0, 2.0))
  + 1000 * max(0, pow(x[2], 2.0) + pow(y[2], 2.0) - pow(radius - 1.0, 2.0))
  + 1000 * max(0, pow(x[3], 2.0) + pow(y[3], 2.0) - pow(radius - 1.0, 2.0)));
```

In the context of continuous packing problems, we need to express the penetration depth between two objects. Between circles, the penetration depth can be expressed with simple formulas as shown in Section 2.1. Between polygons, the definition is harder. For instance, the penetration depth between two convex polygons is the distance between the origin and the Minkowski sum of the two convex polygons [3]. We don't know simpler method for computing the penetration depth between two triangles. The Minkowski sum between two convex polygons is a convex polygon which can be obtained by sorting the edges of the two input polygons by increasing slope. This can be defined in ClpZinc as follows

```
minkowski_sum(Polygon0, Polygon1, PolygonMinkowski) :-
  edges(Polygon0, Edges0),
  edges(Polygon1, Edges1),
  permutation(Edges0 ++ Edges1, EdgesMinkowski),
  increasing_slope(EdgesMinkowski),
  polygon_from_edges(EdgesMinkowski, PolygonMinkowski).
```

ClpZinc models with choice-points are compiled into MiniZinc models via the use of reification [7]. Such a ClpZinc definition generates a reified constraint in MiniZinc which enumerates over the permutations of the edges. The reified constraints are translated into cost functions as described above in this section.

4 Evaluation Results with the CMA-ES Solver

We show in this section how the packing examples of [8] can be reimplemented declaratively in ClpZinc. Here is the code for packing 10 circles of radii $i^{-1/2}$ for i from 1 to 10.

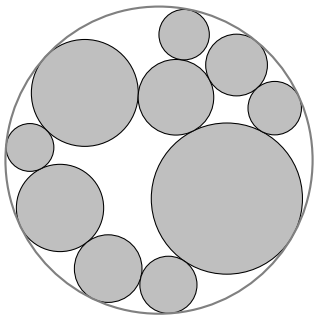
```
include "packing.plz";

int: n;
array [1 .. n] of var float: x;
array [1 .. n] of var float: y;
var float: radius;
```

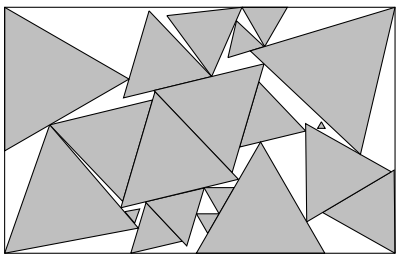
```
:-
   Shapes = [circle(x[i], y[i], pow(i, -1/2)) | i in 1 .. n],
   non_overlap(Shapes),
   bounding_circle_lt(Shapes, radius),
   radius > 0,
   minimize(radius).

n = 10;
```

Here is the code for packing 20 equilateral triangles with sides of length i for i from 1 to 20.

```
include "packing.plz";

float: pi;
int: n;
array [1 .. n] of var float: x;
array [1 .. n] of var float: y;
array [1 .. n] of var float: a;
var float: x0;
var float: y0;
var float: x1;
var float: y1;

shape(equilateral_triangle(X, Y, Size, Angle), [
  triangle(
    X, Y,
    X + cos(Angle + pi / 6.0) * Size, Y + sin(Angle + pi / 6.0) * Size,
    X + cos(Angle - pi / 6.0) * Size, Y + sin(Angle - pi / 6.0) * Size)]).

:-
   Shapes = [equilateral_triangle(x[i], y[i], i, a[i]) | i in 1 .. n],
   non_overlap(Shapes),
   bounding_box(Shapes, x0, y0, x1, y1),
   minimize((x1 - x0) * (y1 - y0)).

n = 20;
pi = 4.0 * atan(1.0);
```

The following pictures illustrate two solutions found by CMA-ES for packing the 10 circles and the 20 equilateral triangles, obtained in 25s and 19min respectively (on a Intel(R) Xeon(R) CPU E5-1620 0 @ 3.60GHz), similarly to [8].





5 Conclusion and Perspectives

In this paper we have shown how a MiniZinc constraint model over not only integer but also real-valued variables can be translated into a fitness function over the reals and can be solved using a stochastic optimization procedure such as CMA-ES [4].

This approach has been illustrated by the modeling of complex mixed shape continuous placement problems and their solving using CMA-ES. More specifically, we have used ClpZinc, an extension of MiniZinc with records and Horn clauses, to define the penetration depths between objects, using Minkowski sums for polygons, and the compilation chain from ClpZinc to MiniZinc and FlatZinc using reified constraints. First evaluation results have shown the absence of significant overhead when compared to their direct encoding in CMA-ES as done in [8].

A classical difficulty in the definition of error function for a conjunction of constraints is the normalization of the error function for each constraint. This has been solved here by letting the model specify the objective function but dynamic strategies inspired from previous work on Comet for instance could be developed [9].

Our approach is general. We have focused on continuous placement problems, but our MiniZinc/CMA-ES can be applied *in principle* to any continuous constraint model and also discrete domains through integrity relaxation. This will be the topic of future work.

Acknowledgements. This work has been funded by the ANR Blanc Net-WMS-2 grant. We would like to thank all the partners of this project for fruitful discussions.

References

1. G. Björdal, J.-N. Monette, P. Flener, and J. Pearson. A constraint-based local search backend for minizinc. *Constraints*, 20(3):325–345, 2015.
2. I. Castillo, F. J. Kampas, and J. D. Pintér. Solving circle packing problems by global optimization: Numerical results and industrial applications. *European Journal of Operational Research*, 191(3):786–802, 2008.
3. D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.
4. N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.
5. P. V. Hentenryck and L. Michel. Synthesis of constraint-based local search algorithms from high-level models. In *Proc. AAAI*, pages 273–278, 2007.
6. E. Huang and R. E. Korf. Optimal rectangle packing: An absolute placement approach. *Journal of Artificial Intelligence Research*, 46:47–87, 2012.
7. T. Martinez, F. Fages, and S. Soliman. Search by constraint propagation. In *Proceedings of the 17th International Conference on Principles and Practice of Declarative Programming, PPDP'15*. ACM, 2015.

8. T. Martinez, L. Vitorino, F. Fages, and A. Aggoun. On solving mixed shapes packing problems by continuous optimization with the cma evolution strategy. In *Proceedings of the first Computational Intelligence BRICS Congress BRICS-CCI'13*, pages 515–521. IEEE Press, Sept. 2013.
9. L. Michel and P. V. Hentenryck. The comet programming language and system. In *Proc. Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 881–881, 2005.
10. I. Salas and G. Chabert. Packing curved objects. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI'15*, Buenos Aires, Argentina, 2015.
11. The Zinc team. MiniZinc web page. `http://www.minizinc.org/`.