Evolutionary Multi Point Search In CPLEX Studio's Constraint Programming Engine

Renaud Dumeur presenting CPLEX Optimization Studio Team
Who are we?

- ILOG solver circa 1988
- ILOG has been acquired by IBM in 2008
- The CP Optimizer team is led by Paul Shaw
- It is distributed over multiple countries
- The product we develop is CP Optimizer, the constraint programming solver embedded in IBM CPLEX Optimization Studio.
Applications of CP Optimizer:
- Scheduling: Project scheduling, aircraft assembly, Rostering (often in conjunction with CPLEX), hospital staffing, maintenance scheduling, aircraft or train maintenance, scheduling of transport zones, seaport scheduling, manufacturing, production line scheduling
- Configuration: configuration of new cars
- Timetabling: sports timetable creation

CP Optimizer includes a search method (MultiPoint) which belongs to Evolutionary Algorithm (EA) family

We will describe how this EA has been hybridized with Constraint Programming and the various challenges we met
What is Constraint Programming?

- Constraint Programming (CP) is an optimization technology which is complementary to Math Programming. While taking a different approach to optimization, it has similarities to MP.
- Major strength is in scheduling and highly combinatorial problems.
- CP has concepts such as variables, constraints, and an objective function.
- Variables: unknowns to be decided.
  - Are typically finite domain.
  - Can have enumerated domains, for example {2, 3, 5, 7, 11, 13, 17, 19}.
- Constraints can capture structure and can have rich semantics. e.g.
  - all-different ($x_1...x_n$)
  - $y = \text{count}(x_1...x_n, v)$
  - $(y^2 \leq z) \lor (w(y+z) \geq 10)$
- Also, the objective function can be a general expression. e.g.
  - minimize $\sum_{i=1}^{n} c_i |x_i - d_i|$
How Constraint Programming Works

- CP is a constructive approach
- Values are assigned to variables one at a time to extend a partial solution until a complete solution is produced.
- When assigning a value, a constraint violating the current partial solution can trigger backtracking to explore other variable values.
- Domain filtering happens before each value-variable assignment
- Variable values which are unusable in the current partial assignment can be removed using a specialized algorithm.
What are Evolutionary Algorithms?

- Evolutionary Algorithms are general search methods inspired by Darwin's theory of natural evolution of species. (Holland 1978, Rechenberg 1973)
- As natural evolution solves the problem of adaptation to a natural environment, why not apply its underlying mechanics to solve our own problems?
- What helps is that natural evolution works on bits as we do
- Mutants, new species appear due to changes in DNA, and DNA molecules can be represented as strings of letters (ATCG).
- Changes in those letters may result in changes in organism metabolism, morphology or behavior better adapted to the natural environment.
- Instead of molecules, Evolutionary Algorithms work on populations of objects in computer memory and attempt to produce data structures adapted to user-specified requirements.
- The model is represented as a solution “fitness” function
- Evolutionary algorithms are known to be particularly robust on problems of high dimensionality
How evolutionary algorithms work

- Evolutionary algorithms are *perturbative* search methods
- Darwinian search space exploration
- Uses a population of search space points
- Can start from random search points
- Typical generation cycle:
  - Evaluate search point fitnesses
  - Select parents based on fitness
  - Produce offspring (e.g. using mutation)
  - Replacement: store offspring in population

Mutation

Hybridisation
Hybridizing Evolutionary Algorithms and Constraint Programming

- Classically, Evolutionary Algorithms requires relaxing feasibility problems by pushing constraints as objective function components
  - E.g. Constraints violations are linearly combined
    - How to choose violation weights?
    - How to relax global constraints?
    - How to mix weighted violations with actual problem objectives

- Evolved data structure and evolutionary operators are specific to the solved problem

- CP Optimizer MultiPoint search avoids any model reformulation
- Effortlessly reap benefits of search diversification brought by an EA
- The EA can be applied to pure integer, pure scheduling or hybrid models
EAs were initially designed to solve unconstrained problems
Hybridizing Evolutionary Algorithms and Constraint Programming

CP search spaces have infeasible regions (hard constraints)
What to do with infeasible assignments?

- Use an indirect, messy representation (genome)
- Decode the genome to produce an assignment (phenotype)

What if the decoder still fails to produce valid assignments?
- CPO's MultiPoint search will attempt to evolve invalid assignments until valid ones arise, by favoring genomes that maximize the number of valid assignments obtained during decoding.
Genome Decoding Process

Genes can be silenced by model constraints

Model Constraints

Propagation Engine

Genome based constraints

Decoder

Constraint Generation

Completion

Filled By Propagation

Variable Domains

© 2013 IBM Corporation
Can we learn from known feasible assignments?

- Specify an existing solution as a starting point.
- CPO MultiPoint search is able to “genetically engineer” genomes.

Specified as starting point by user

a

g

z

Decoder chooses to encode preferences for variables a, g and z

Sparse genome describing preferences on variables a, g and z

- Improve convergence by tweaking genomes from existing solutions
Benchmarking

- Default search method (Restart) is robust
- When Workers=4, MultiPoint is active and allows to complement the default method
- Tests made with between 10 and 30 random seeds depending on the instance.
- Using our integer problem tests, 13% of test cases work better with MultiPoint, when 17% show similar performance.
- Using scheduling models 14% of test cases show improvements when using MultiPoint
Enabling MultiPoint in CP Optimizer

- In C++, set the SearchType parameter

```cpp
IloEnv env;
IloCP cp(env);
// build your model here
cp.setParameter(IloCP::SearchType, IloCP::MultiPoint);
```

- In OPL, use the GUI or script:

```opl
execute {
    cp.param.SearchType = "MultiPoint";
    cp.param.MultiPointNumberOfSearchPoints = 10; // small population, faster convergence.
}
subject to constraints {
    // define your model here
}
```