Changing the rules of business™

Using Genetic Algorithms in ILOG Solver

CPAIOR '05

Outline

Order of business

- EAs in ILOG Solver
- Example problem: bin packing
- Direct and indirect representations
- Creating initial populations
- Selection and evolution
- CP-based intensification
- Some results

What is Solver?

- A C++ Library for constraint programming
  - Many constraint types with powerful propagation
  - Extensible
    - New constraints
    - New search strategies
- Solver IIM (Iterative Improvement Methods)
  - Local search & evolutionary algorithms
  - Extensible
    - New local search neighborhoods
    - New evolutionary operators

EAs in ILOG Solver IIM 6.1

Changing the rules of business™

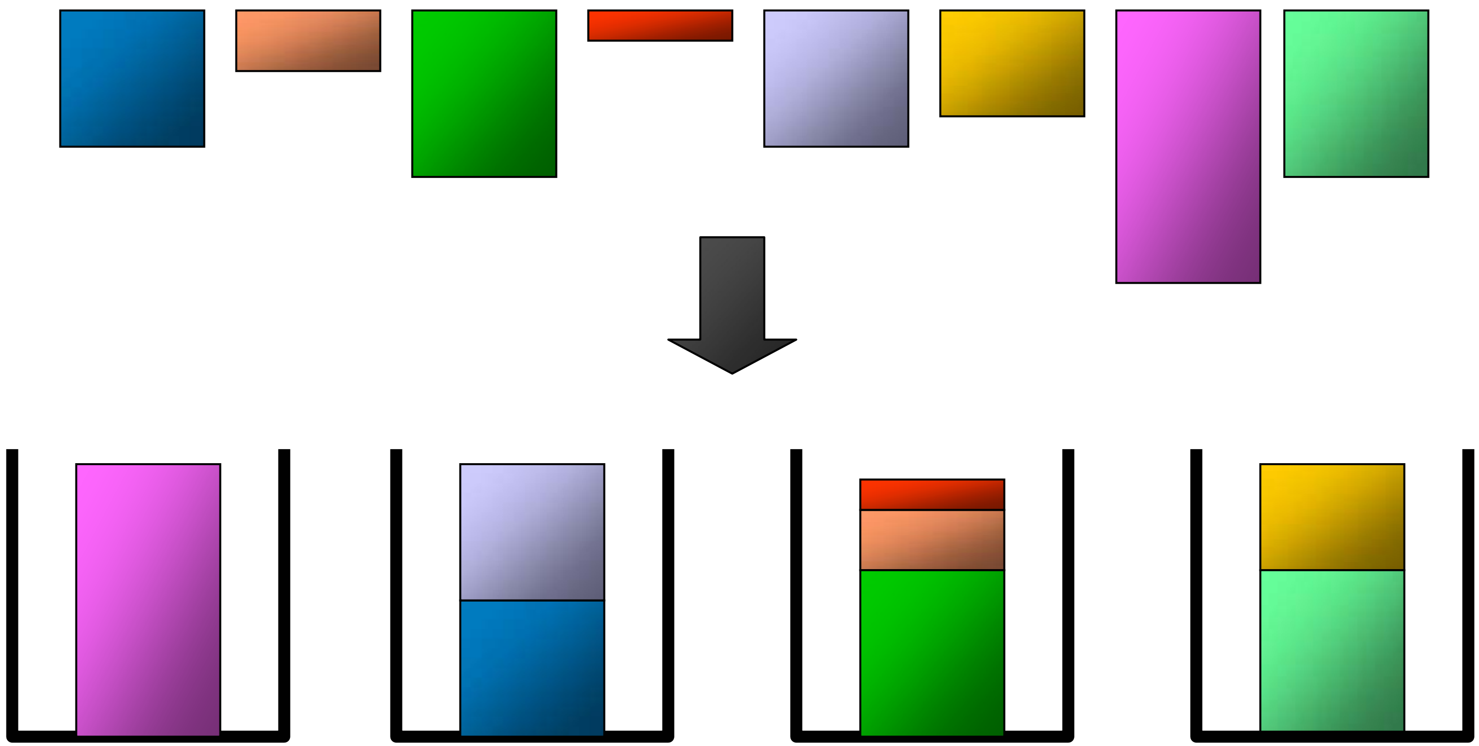
Basics

- For constrained optimization problems
- Solution and solution pool objects
  - One solving engine, many solution objects
- Solution pool operator
  - Takes a set of solutions and instantiates constrained variables according to operator
- Solution pool processor
  - As operator, but generates solution objects
- Solution selectors, comparators, evaluators

Example: Bin Packing

Problem

- **Pack a set of *n* items of weight *w(i)* into a minimum number of bins, each of capacity *C***



Example: Bin Packing

Model

```
IloEnv env
IloIntArray weight
IloInt cap
```

- `IloEnv env` — Environment: serves as a global object manager
- `IloIntArray weight` — Array of object weights (sorted non-increasing)
- `IloInt cap` — Capacity of the bins

```
(1) IloInt numItems = weight.getSize();
(2) IloInt numBins = numItems;
(3) IloIntVarArray where(env, numItems, 0, numBins - 1);
(4) IloIntVarArray load(env, numBins, 0, cap);
(5) IloInt lb = (IloSum(weight) + cap - 1) / cap;
(6) IloIntVar used(env, lb, numBins);
(7) IloModel model(env);
(8) model.add(IloPack(env, load, where, weight, used));
```

Example: Bin Packing

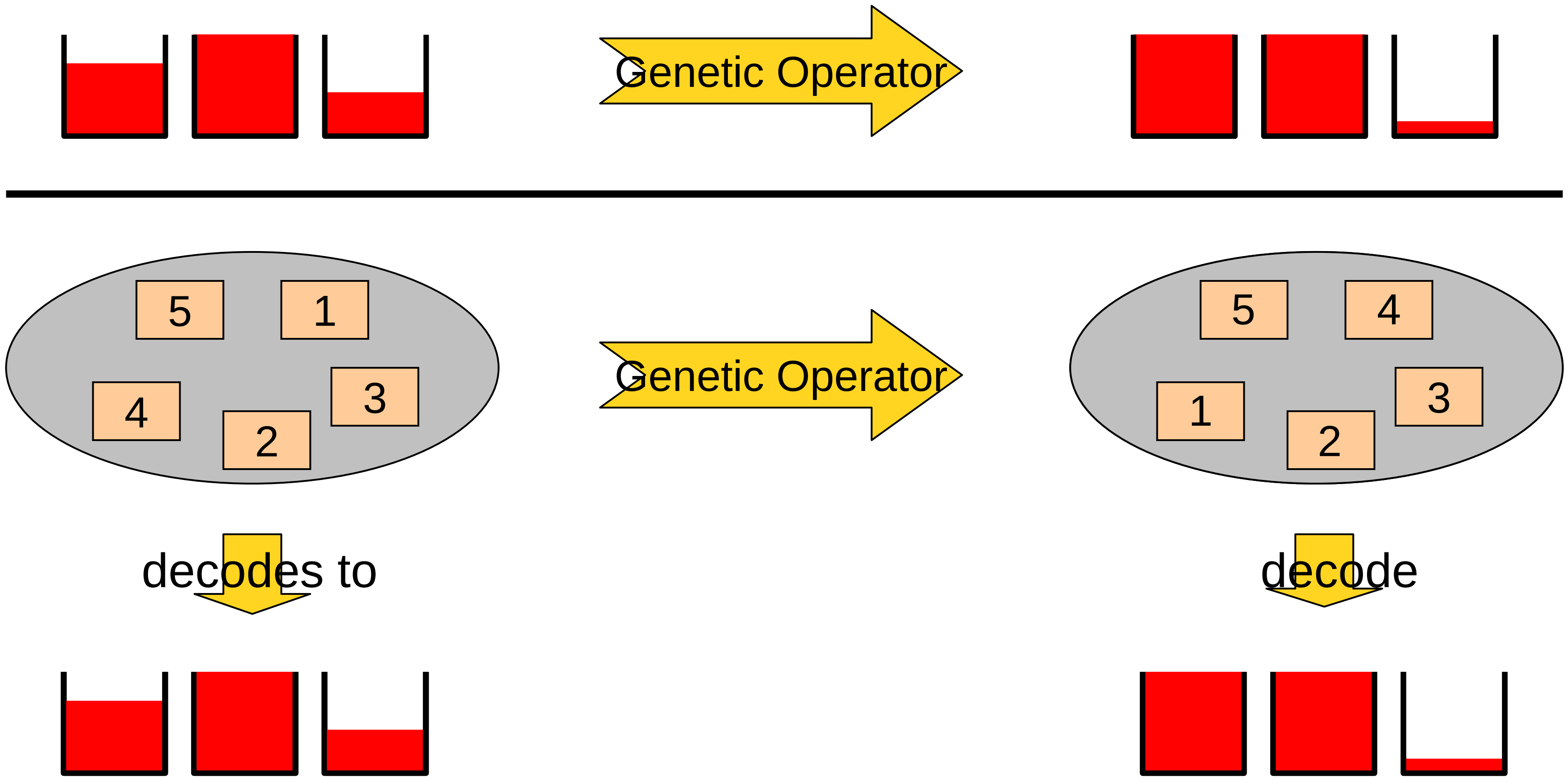
Constructive method

- Decreasing first fit
  - **Consider items according to decreasing weight**
  - **Place each item in the first bin with enough space**
  - **On failure, disallow item from this first bin**
- Implementation
  - **Sort “where” vars. by decreasing object weight**
  - **Instantiate these variables in the given order**
    - Choose minimum value for variable (first available bin)
  - **Implemented using `IloGenerate` of Solver**

Representations

- Direct: work directly on decision variables
  - chromosome = decision variable (“where” var.)
  - Normally use *custom* genetic operators
  - For bin packing, use a “grouping operator”
- Indirect: work on new additional variables *p*
  - Here, these vars. assign a priority to each item
  - Run EA on priorities using classic operators
  - *Decode* priorities to “where” variables
  - Decode = DFF except priorities replace weights

Representations



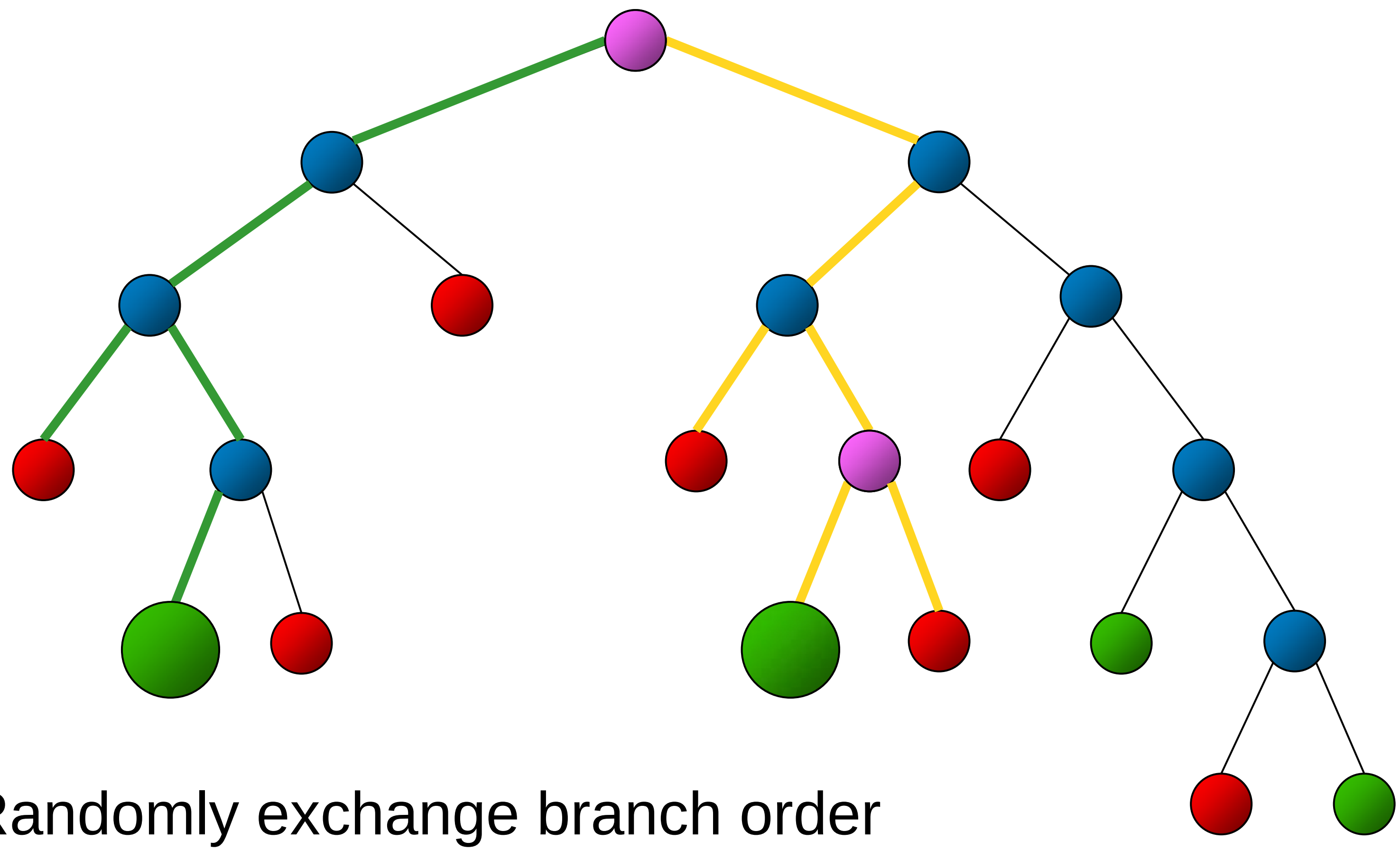
Initial Population

Direct representation

- Constraints discount random solutions
- Use a constructive search for first solution
  - Diversity: use a *random* variable / value choice
  - Restart search after limit is node reached
  - Downside: random solutions often poor
- Can we trade quality vs. diversity?
  - Yes. If we have a “good” search strategy
  - Mix a “good” CP search with a random one

Initial Population

Tree search perturbation



Randomly exchange branch order

Initial Population

Direct Representation

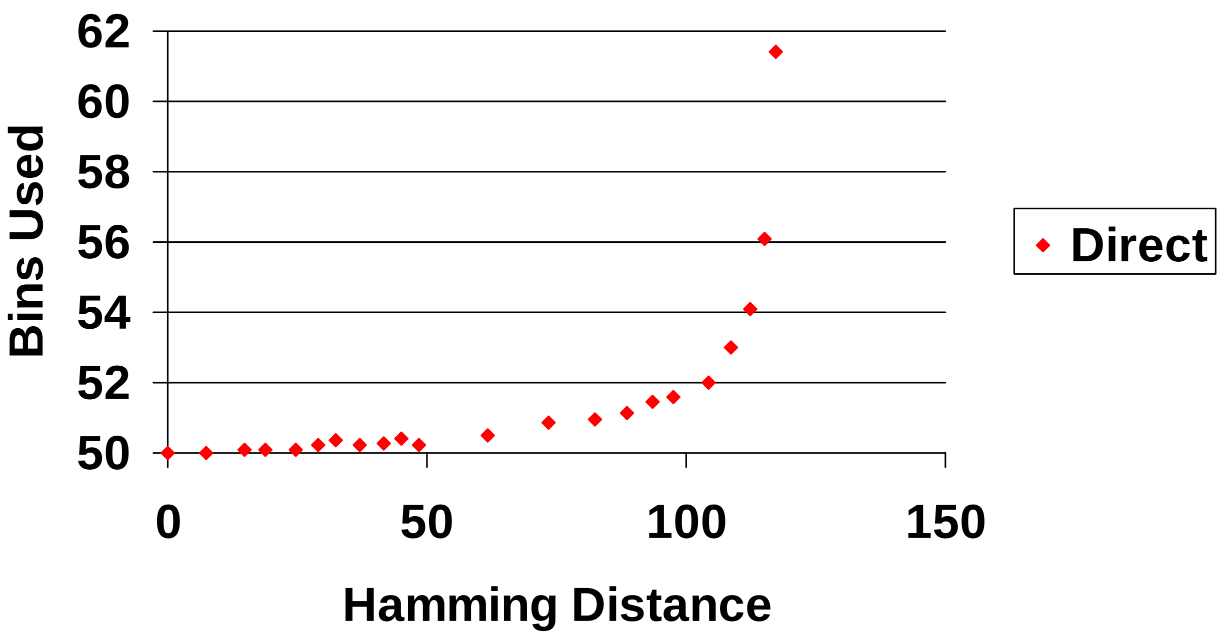
```
 (1) IloSolution prototype(env);
 (2) prototype.add(where);
 (3) prototype.add(load);
 (4) prototype.add(used);
 (5) prototype.add(IloMinimize(env, used));

 (6) IloSolutionPool popn(env);

 (7) IloGoal dff = IloGenerate(env, where);
 (8) IloGoal packGoal = IloRandomPerturbation(env, dff, 0.3);

 (9) IloPoolProc source = IloSource(env, packGoal, prototype);
(10) IloPoolProc initPop = source(popSize) >> popn;
(11) IloSolver solver(model);
(12) solver.solve(IloExecuteProcessor(env, initPop));
```

Initial Population Diversity

Changing the rules of business™

Controlling diversity

Diversity / Cost Correlation



Evolve the population

Direct representation

```
IloPoolOperator xo = XOOperator(env, load, where, weight, used);
xo = IloLimitOperator(env, xo && packGoal, IloFailLimit(env, 100));

IloRandomSelector<IloSolution,IloSolutionPool> rndSelector(env);
IloPoolProc evolve = popn
                   >> IloSelectSolutions(env, rndSelector)
                   >> xo(popSize / 3)
                   >> IloReplaceSolutions(env, popn)
                   >> IloDestroyAll(env);

IloInt best = popn.getBestSolution().getObjectiveValue();
for (IloInt i = 0; i <= maxGen && best > lb; i++) {
  env.out() << "Generation " << i << ": " << best << endl;
  solver.solve(IloExecuteProcessor(env, evolve));
  best = popn.getBestSolution().getObjectiveValue();
}
```

Crossover operator

Direct representation

- **Idea: child inherits well packed bins from a parent**
  - **All items not in well-packed bins are repacked**
  - **Suppose we have a solution using *m* bins**
  - **For *better* solutions, bins need a mean load of *W / (m - 1)***
- **Algorithm**
  - **Consider bins in a random order**
  - **Inherit *all* items from the bin of one (random) parent if none of these items has been added to the child**
  - **Stop when “enough” bins have been inherited**
  - **(Complete the packing using Deceasing First Fit)**

Crossover operator: code

Direct representation

```
ILOIIMOP4(XOOperator, 2, IloIntVarArray, load, IloIntVarArray, where,
                         IloIntArray, weight, IloIntVar, used) {
  IloSolver solver = getSolver();
  IloInt numBins = load.getSize();
  IloSolutionPool parents = getInputPool();
  IloInt numBinsUsed = IlcMax(parents[0].getValue(used), parents[1].getValue(used)) - 1;
  IloInt reqdMeanLoad = (IloSum(weight) + numBinsUsed - 1) / numBinsUsed;
  IlcRandom rnd = solver.getRandom();
  IlcIntArray order(solver, numBins);
  MakeRandomOrder(order, rnd);
  IloInt toDo = rnd.getInt(numBinsUsed);

  for (IlcInt i = 0; i < numBins && toDo > 0; i++) {
    IlcInt bin = order[i];
    IlcBool good0 = parents[0].getValue(load[bin]) >= reqdMeanLoad &&
                    AllItemsUnpacked(solver, parents[0], bin, where);
    IlcBool good1 = parents[1].getValue(load[bin]) >= reqdMeanLoad &&
                    AllItemsUnpacked(solver, parents[1], bin, where);
    if (good0 || good1) {
      IloSolution soln = parents[!good0 || (good1 && rnd.getInt(2))];
      PackAllItems(solver, soln, bin, where);
      toDo--;
    }
  }
  return 0;
}
```

Initial Population

Indirect representation

- Priorities are unconstrained
  - Random solutions will do just fine!
  - Decode each set of priorities into a packing
    - Backtracking decoder respects constraints
- Again, can we trade off quality vs. diversity?
  - Yes. If we know a good set priorities
  - Can prioritize items according to weight
  - Mix “good priorities” with random ones

Decoder

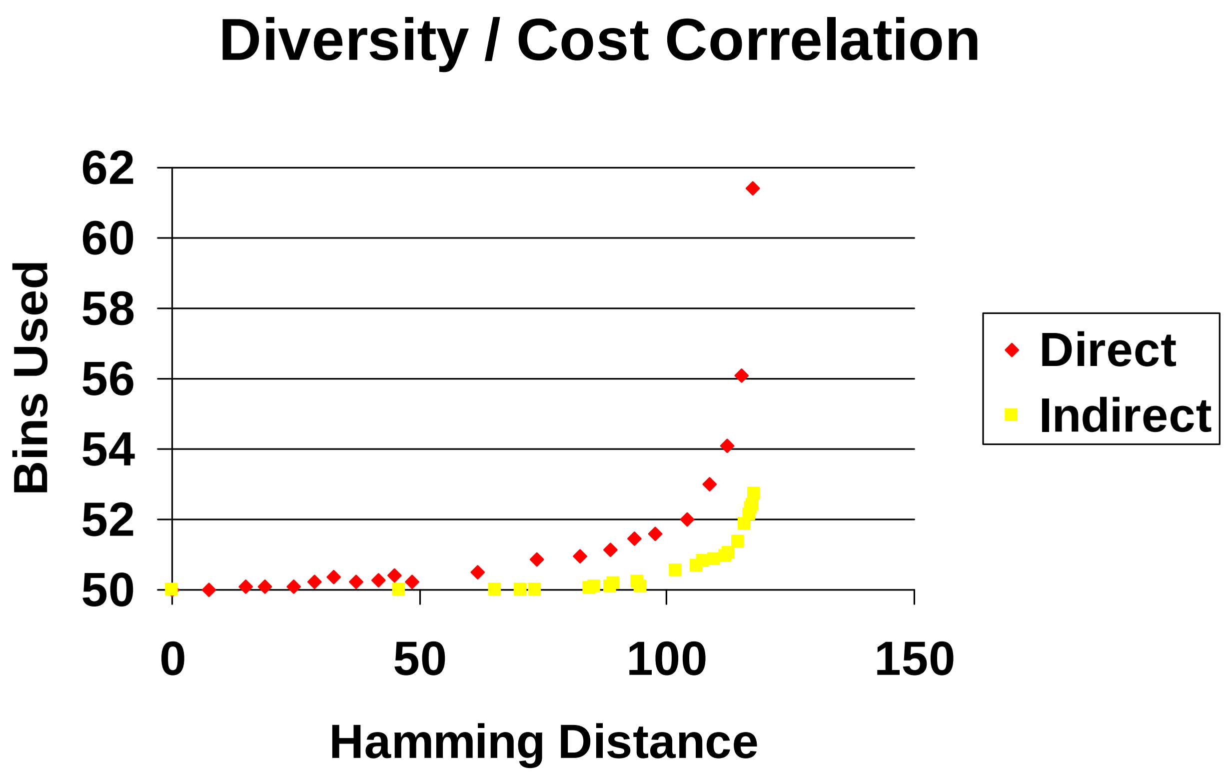
Indirect representation

- **Decoder: pack items in priority order**

```
ILCGOAL2(Decode, IlcIntVarArray, prio, IlcIntVarArray, where) {
  IlcInt best = -1;
  for (IlcInt i = 0; i < where.getSize(); i++) {
    if (!where[i].isBound() &&
       (best < 0 || prio[i].getValue() > prio[best].getValue()))
      best = i;
  }
  if (best >= 0)
    return IlcAnd(IlcInstantiate(where[best]), this);
  return 0;
}
```

Initial Population Diversity

Controlling diversity



Evolve the population

Indirect representation

```
... IloIntVarArray priority(env, 0, maxPriority);
    IloGoal decode = Decode(env, priority, where); ...
IloEAOperatorFactory f(env, priority);
f.setSearchLimit(IloFailLimit(env, 100));
f.setAfterOperate(decode);

IloPoolProcArray ops(env);
ops.add(f.uniformXover());
ops.add(f.mutate(4.0 / numItems);
IloRandomSelector<IloPoolProc,IloPoolProcArray> rndSelector(env);
IloPoolProc breed = IloSelectProcessor(env, ops, rndSelector);
IloPoolProc gen = popn
               >> IloSelectSolutions(env, rndSelector)
               >> breed(popSize / 3)
               >> IloReplaceSolutions(env, popn)
               >> IloDestroyAll(env);
```

Packing quality measure

Indirect representation

- **For equal numbers of bins, better quality packings have bigger chunks of free space**

```
ILOEVALUATOR2(PackQuality, IloSolution, s,
              IloIntVarArray, load, IloInt, cap) {
  IloNum q = 0;
  for (IloInt i = 0; i < load.getSize(); i++) {
    IloInt ld = s.getValue(load[i]);
    if (ld != 0)
       q += (cap - ld) * (cap -ld);
  }
  return q;
}
```

Finer replacement method

Indirect representation

```
IloEvaulator<IloSolution> quality = PackQuality(env, load, cap);
IloEvaluator<IloSolution> cost = IloSolutioEvaluator(env, used);
IloComparator<IloSolution> compareWorst = IloComposeLexical(
   cost.getGreaterThanComparator(),
   quality.getLessThanComparator()
);
IloSolutionPoolSelector selectDead =
  IloBestSelector<IloSolution, IloSolutionPool>(compareWorst);

 IloPoolProc gen = popn
               >> IloSelectSolutions(env, rndSelector)
               >> breed(popSize / 3)
               >> IloReplaceSolutions(env, popn, selectDead)
               >> IloDestroyAll(env);
```

CP-based intensification

Using quality constraints

- **Cost UBs can help tree search find better solutions**
  - **When completing the child in a direct representation**
  - **When decoding an indirect representation**
  - **Child is forced to be at least as good as worst parent**

```
xo = IloImproveOn(env, used, 0.5) && xo;

ILOIIMOP2(ImproveOn, 1, IloIntVar, used, IloNum, p) {
  IloInt lim = getInputPool().getWorstSolution().getObjectiveValue();
  IloSolver solver(getSolver());
  solver.add(used <= limit - (solver.getRandom().getFloat() < p));
  return 0;
}
```

Some results

Changing the rules of business™

- **Population size = 30, 1 minute time limit**
- **Falkenauer's 120 item problems (20 instances)**

Representation	+cp intens	+ quality	# optima
Direct	N	N	15
Direct	N	Y	19
Direct	Y	N	20
Direct	Y	Y	20
Indirect	N	N	9
Indirect	N	Y	15
Indirect	Y	N	17
Indirect	Y	Y	19

Summary

- **ILOG Solver IIM has EA for constraint optimization**
  - **Use standard CP search as a base**
- **Support for direct and indirect representations**
- **Selectors/evaluators/comparators**
- **You can define your own operators**
  - **Operators can perform a local search, or LNS**
- **CP brings fundamental advantages**
  - **Can *directly* tackle constrained problems**
  - **Decoding simplicity in the presence of side constraints**
  - **Constraint-based acceptance criteria**

Create an initial population

Changing the rules of business™

Indirect representation: code

```
const IloInt maxPriority = IloMax(weight);
IloGoal decode = Decode(env, priority, where);
for (IloInt i = 0; i < popSize; i++) {
  IloSolution soln = prototype.makeClone(env);
  for (IloInt j = 0; j < numItems; j++) {
    IloInt prio = weight[j];
    if (env.getRandom().getFloat() < 0.1)
      prio = 1 + env.getRandom().getInt(maxPriority);
    soln.setValue(priority[j], prio);
  }
  IloGoal evaluate = IloRestoreSolution(env, soln)
                  && decode
                  && IloStoreSolution(env, soln);
  solver.solve(evaluate);
  popn.add(soln);
}
```