CSG2341
Intelligent Systems

Evolution Strategies

Evolution Strategies

- The first examples of ES were done without the use of computers, using real physical models.
- Early examples:
  - a jet nozzle (1968)
  - using wind tunnel experiments to evolve the shape of a kinked plate with minimal drag (1964).
- Distinguishing features of ES include:
  - Populations are often small;
  - Different selection mechanism from GAs;
  - Often aimed at designing physical objects.

ES Representation

- Often use sequences of real valued parameters. For example:
  - in the jet nozzle, the numbers represent the radii of the nozzle segments;
  - in the kinked plate, the numbers represent the angles between sections of the plate.
- For combinatorial optimisation problems, candidate solutions are permutation vectors (e.g. 1,3,2 representing the ordering item 1, then item 3, then item 2).

ES Operators

Often, the only operator is mutation (no crossover).

Mutation for real values

- adding normally distributed independent random values, N(0,σ), to each parameter. The value σ determines the size of the changes and is called the *mutation strength*. (Other distributions may also be used.)
- using fixed mutation strength does not work well. There are several schemes for adapting σ

ES self-adaptation

Self-adaptation is one way of changing the mutation strength σ:

- evolve the value of σ along with the solution;
- each candidate solution has its own mutation strength
  - inherited by its offspring,
  - must specify how it is to be updated,
  - one method is to update σ by multiplying by $(1+\tau N(0,1))$, where τ is some value between, say 0.2 and 0.5.

ES Operators (cont)

Permutations Example mutation operators :

Inversion:

1	2	3	4	5	6	7	8	9

becomes

1	5	4	3	2	6	7	8	9

ES Operators (cont)

Insertion:

1	2	3	4	5	6	7	8	9

becomes

1	3	4	5	2	6	7	8	9

ES Operators (cont)

Exchange:

1	2	3	4	5	6	7	8	9

becomes

1	5	3	4	2	6	7	8	9

ES Operators (cont)

Shift:

1	2	3	4	5	6	7	8	9
	↑			↑		↑		

becomes

5	1	6	7	8	9	2	3	4

ES Operators

- If recombination is used :
  - Discrete: each component copied from one parent or the other at random
  - Intermediate: each component is a linear combination of the two (or more) parents
- Recommended to use discrete recombination for object parameters, intermediate recombination for strategy parameters.

ES Selection

In ES, two populations– a parent population having μ members, and an offspring population having λ members. There are two kinds of selection strategies:

- **(μ, λ) selection**:
  - also called comma selection;
  - a non-elitist strategy;
  - the best μ members of the offspring pool become the new generation of parents.
- **($\mu + \lambda$) selection**:
  - also called plus selection;
  - an elitist strategy;
  - the best μ members of the offspring pool plus the parent pool become the new generation of parents;
  - should be used for combinatorial optimisation.

Designing Crushers with a Multi-Objective Evolutionary Algorithm

Luigi Barone, Lyndon While — The University of Western Australia

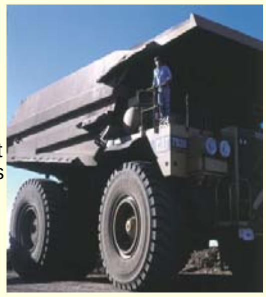
Philip Hingston — Edith Cowan University

Overview

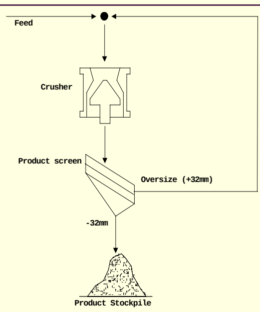
- Iron ore mining in Western Australia
- Ore crusher terminology
- A genetic representation of crushers
- Measuring crusher performance
- The multi-objective EA
- Experimental results
- Conclusions

The Iron Ore Industry in Western Australia

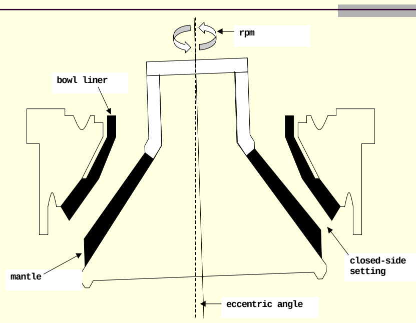
- Iron ore mining was worth AUD$3.4b to the state in 2000/01
- State has 30b tonnes in reserves (third largest in the world)
- Raw ore varies from dust particles to ~5m boulders
- Export size is <32mm



A Simple Comminution Circuit



A Schematic of a Cone Crusher



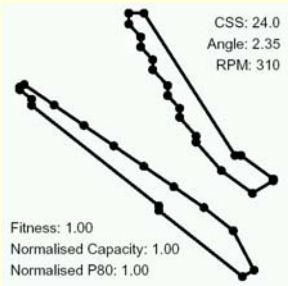
The Problem

- Given: a specification of a circuit and models for simulating comminution components
- The task: design a tool for automatically creating better crusher designs for a variety of different scenarios
- The approach: use an evolutionary algorithm to search the space of possible crusher designs

Genetic Representation

- Represent the machine settings (*CSS*, eccentric angle, and rotational speed) as real-valued variables in the EA
- The end-points of both liners are fixed, but the internal shape may vary
- Represent each liner as a series of line segments using a variable-length list of coordinates
- Use an ES to evolve the population

The Base Crusher



Measuring Crusher Performance

- Crusher performance is measured by two (potentially conflicting) objectives:
  - maximise the capacity of the circuit containing the crusher
  - minimise the size of the product
- Define *P80* as a measure of the size of the 80th percentile in the product
- *P80* is to be maximised

Capacity Constraints

- The capacity of a circuit may be limited by any one of three factors:
  - the capacity and throughput of the crusher
  - the power requirements of the crusher
  - the capacity of recirculation conveyors
- Define the capacity (*CAP*) of the circuit as the minimum of these constraints
- *CAP* is to be maximised

A Single Objective Algorithm

- Attempt to combine different objectives into one fitness function
- But which is more important?
- The likely range of *CAP* values is approximately 20 times the likely range of *P80* values
- Use the fitness function:

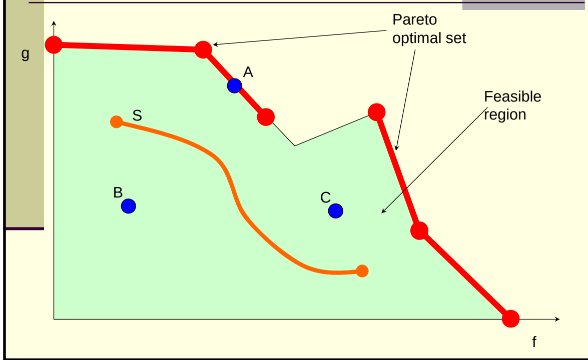
  $fitness = 0.05 \times CAP + 0.95 \times P80$

  as the basis for selection in the EA

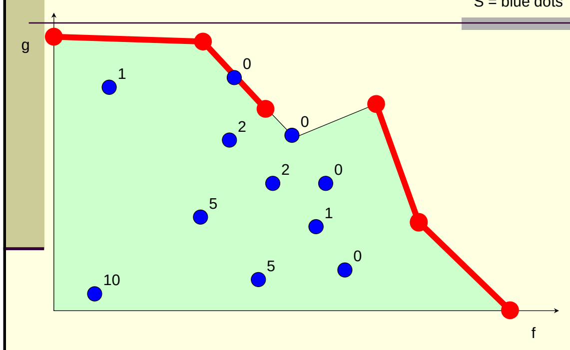
The Multi Objective Algorithm

- Define Pareto dominance between designs using CAP and P80
  - *x dominates y if x has higher CAP and higher P80 than y*
- Use Pareto ranking instead of fitness
  - *Pareto rank of x = number of designs in the population that dominate x*

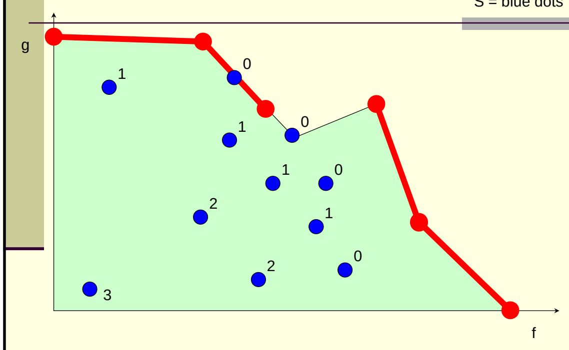
The Multi Objective Algorithm



Pareto ranking – Fonseca and Fleming



Pareto ranking – Goldberg



The Multi Objective Algorithm

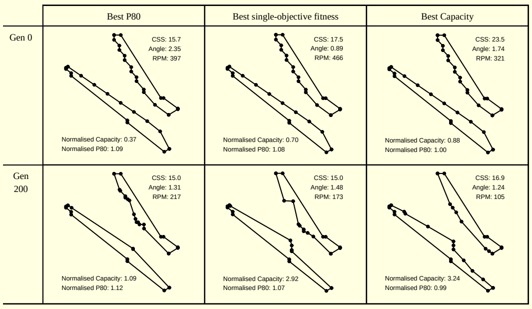
- Selection in MOEAs is not based directly on a solution's fitness, but on its **rank** in the population
- The rank of a solution *A* is a measure of how 'dominated' *A* is in the population
- Two ranking schemes are in common use
- Fonseca and Fleming:
  - rank(*A*) = the number of solutions in *S* that dominate *A*
- Goldberg:
  - rank(*A*) = *0*, if *A* is non-dominated in *S*, otherwise
  - rank(*A*) = the highest rank among the solutions in *S* that dominate *A*, plus *1*

The Multi Objective Algorithm

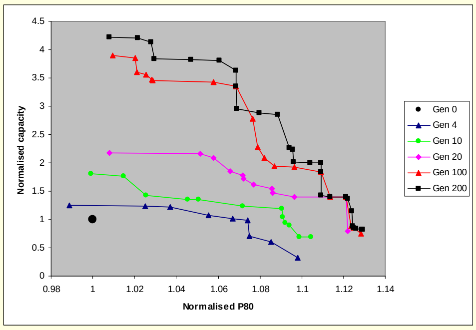
So the steps of the algorithm are:

1. Create an initial population of *n* designs.
2. Evaluate CAP and P80 of these designs.
3. Create a population of *n* children by mutating the members of the current population.
4. Evaluate the CAP and P80 of these children.
5. Select the *n* lowest Pareto rank designs from the parents and children together.
6. Repeat steps 3 to 5 until done.

Typical evolved designs



Progressive Pareto Fronts

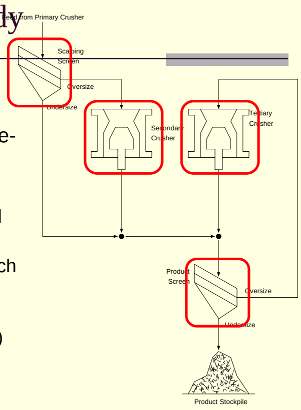


Conclusions

- The EA has produced crusher designs that have shown an improvement of
  - >10% in *P80* (over existing designs); or
  - >200% in CAP; or
  - significant improvement in both simultaneously
- For the "real" problem, we estimate an improvement in profit of ~US$20m per year
- Still needs
  - greater realism (e.g. wear effects, different feeds)
  - validation in field trials

Further Case Study

- More realistic circuit
- Product ore particles must be less than a pre-determined size
- Variables:
  - machine variant used at each component
  - control settings of each component
  - number of parallel machines (unit count) used at each component



Objectives

- Circuit performance is measured by two (potentially conflicting) objectives:
  1. minimisation of the size of the product
  2. minimisation of the overall cost of the circuit
- Define *cost* as a measure of the cost of the circuit:

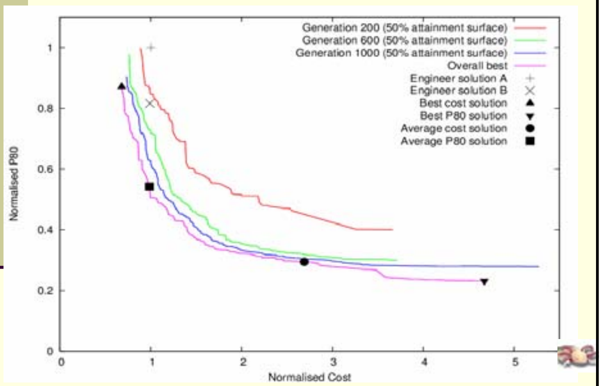
  $cost = \sum component\ cost_c$
- Component cost is non-linear with respect to unit count:

  $component\ cost_c = n_c \times machine\ cost_c \times (0.9 + 0.1 n_c)$

Infeasibility

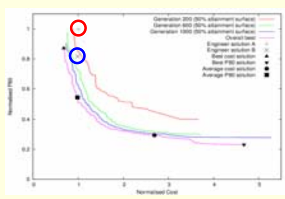
- Not all candidate designs are feasible:
  - crushers may completely fill up with ore particles such that additional ore particles overflow out of the machine
  - ore particles may be too large to enter a crusher
  - product ore particles may be larger than the pre-defined maximum allowed size
- Not all infeasible solutions are equally bad
- Need some way of rewarding “less bad” solutions
- Add a third error objective to measure how infeasible a design is
- Use the error objective to choose between equally ranked solutions during selection

Experimental Results



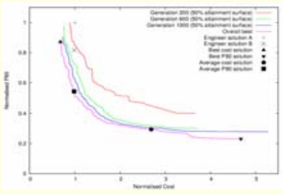
Baseline Engineer Designs

- Engineer solution A is a standard “by-the-book” design
- Engineer solution B is an optimised version of A, based on intuition and experience
- Engineer solution B Pareto dominates engineer solution A



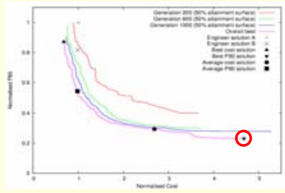
Observations

- Takes ~20 generations before finding first valid (zero error) solution
- Extent of the Pareto front increases over time
- Generates a wide range of different designs
- Produces circuit designs superior in performance to existing designs
- Difficult to improve *P80* beyond a certain value



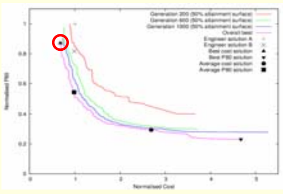
Evolved Designs – Best *P80* Solution

- Trades-off cost for product size reduction
- Uses small openings for the tertiary crusher and product screen
- Contains a large amount of recirculating ore
- Uses high unit counts
- Similar designs are used in stone quarrying where very fine crushing is required



Evolved Designs – Best Cost Solution

- Minimises cost of circuit
- Uses large openings for both screens, reducing “sieving” area and cost
- Employs a smaller (and cheaper), yet coarser secondary crusher
- Uses low unit counts
- A similar design was trialed in practice, - performance dependent on ore composition



Summary

- EA generated a wide range of different designs
- EA produced circuit designs superior in performance to existing designs
- Challenges remained in incorporating more realism:
  - varieties in input ore stream
  - interactions with other processing stages
  - better economic cost models
  - operational practicality considerations (e.g. risk of failure)