IE 607 Heuristic Optimization

Constrained Design

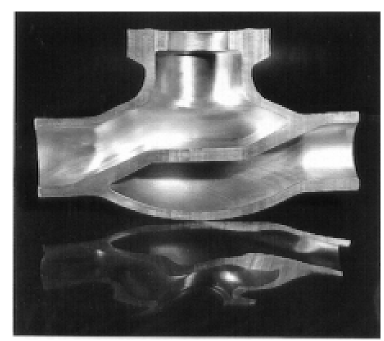
(Resource: Dr. Alice E. Smith)

Constraints Arise In:

- Product design
- Process design
- Process planning
- Plant design
- Plant management
  - scheduling
  - lot sizing
  - sequencing

Examples from Real World

- Eljer *Patriot* toilet - water use, flushing performance, manufacturability, aesthetics
- Ford *SVT Contour* manifold - air flow quantity per passage, interior smoothness
- Superalloy steels - grain size, uniformity, hardness, purity



Why Handle Constraints During Optimization?

- ⇩ Optimal **solutions** (designs, process settings, operational plans, facility floorplans) must be **feasible** to be implemented.
- ⇩ It is often not easy or intuitive to transform an infeasible solution to a feasible solution. And even if this can be done, the feasible solution is often no longer optimal.

Handling Constraints Can be Difficult

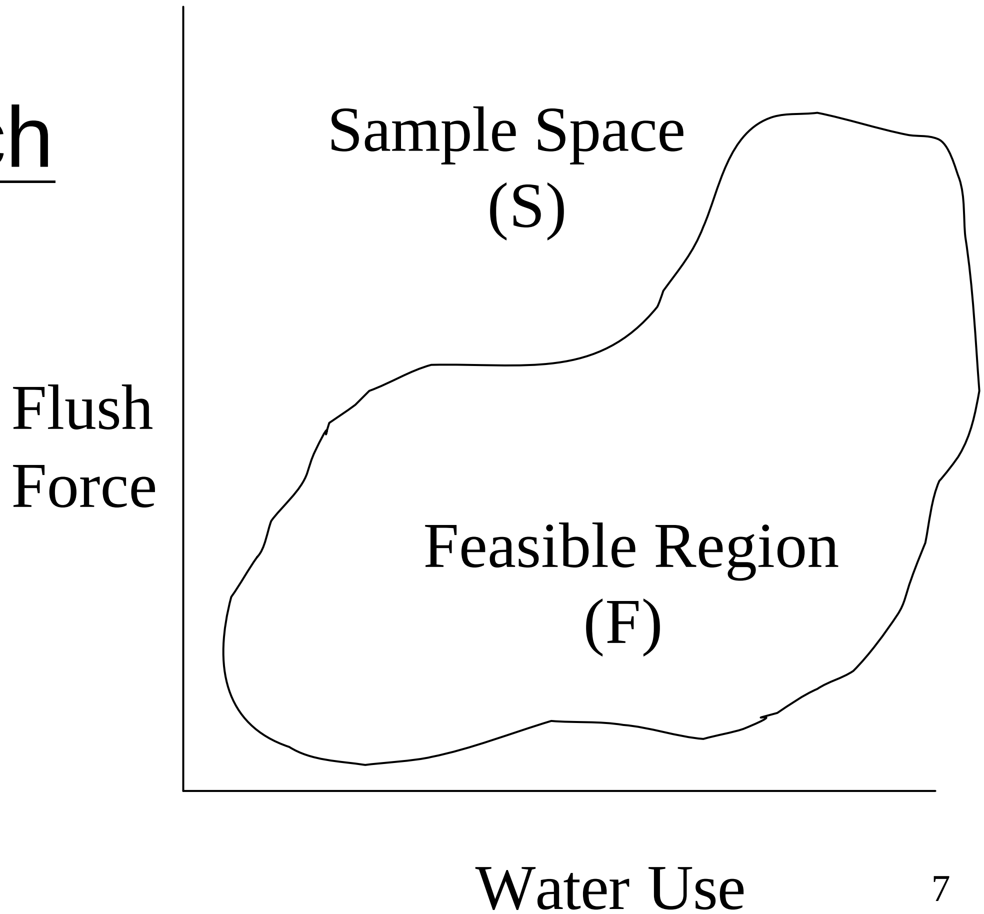
- Multiple (and often *conflicting*) constraints - not obvious which will be active
- Discontinuous feasible regions
- *Hard* and *soft* constraints
- Combinatorial constraints
- Severe constraints (S >> F)
- Constraints of greatly differing magnitudes

Difficulties are Compounded in Adaptive Search

- ☹ Some encodings engender infeasibilities
- ☹ Initial solutions are often random (and infeasible)
- ☹ Recombination (e.g. *crossover*) of feasible solutions often results in infeasible solutions
- ☹ Perturbation (e.g. *mutation*) of feasible solutions often results in infeasible solutions
- ☹ Fitness of feasible versus infeasible solutions is critical to search direction

General Constraint Handling Methods

- ☆**Disallow**
  - limit search
  - discard
- 🕐**Repair**
- 🕑**Penalize**
  - exterior
  - interior



Limit Search to Feasibles

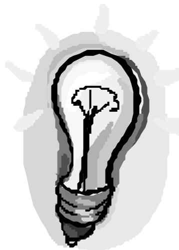
Through encodings, move operators, etc.

Scheduling by permutation encoding

B	F	A	C	E	D	parent 1
E	F	C	A	B	D	parent 2
B	F	C	C	B	D	uniform crossover

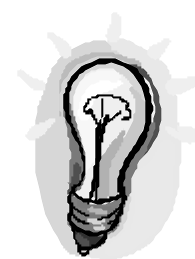
Alter to random keys encoding

A	B	C	D	E	F	
.31	.02	.46	.69	.57	.29	B F A C E D
.48	.51	.32	.62	.17	.24	E F C A B D
.31	.51	.32	.69	.17	.29	E F A C B D



Discard Infeasibles

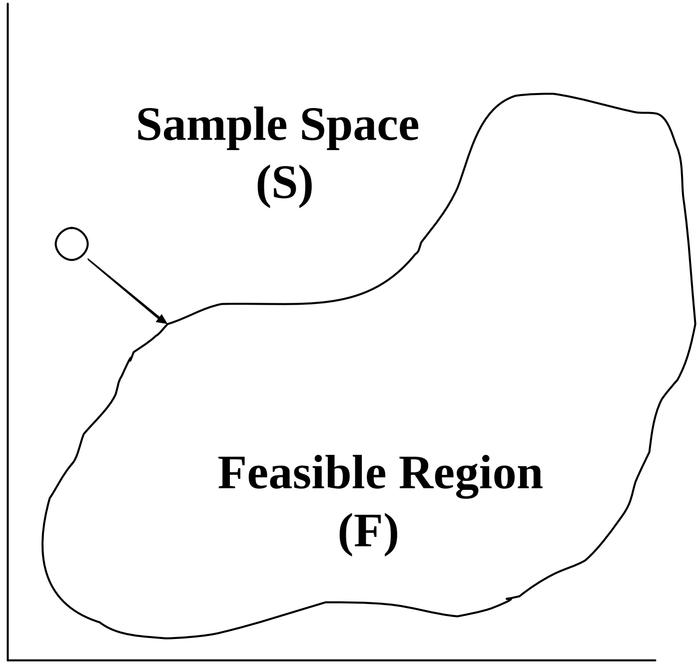
- ☺ Also called the *Death Penalty*
- ☺ Simple and easy to implement
- ☺ Guarantees feasible final solution
- ☹ Effort of generating and evaluating (at least for feasibility) of infeasibles wasted
- ☹ Can lead the search away from the F border and into the F interior (feasible, but suboptimal)
- 😐 Effective if S > F but not S >> F



Repair

For effective repair:

- ✰ repair must be computationally simple
- repair must not disturb original solution too much
- question of whether to replace repaired solution or just fitness
- it must occur relatively infrequently

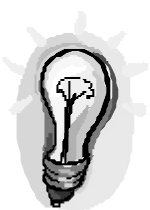


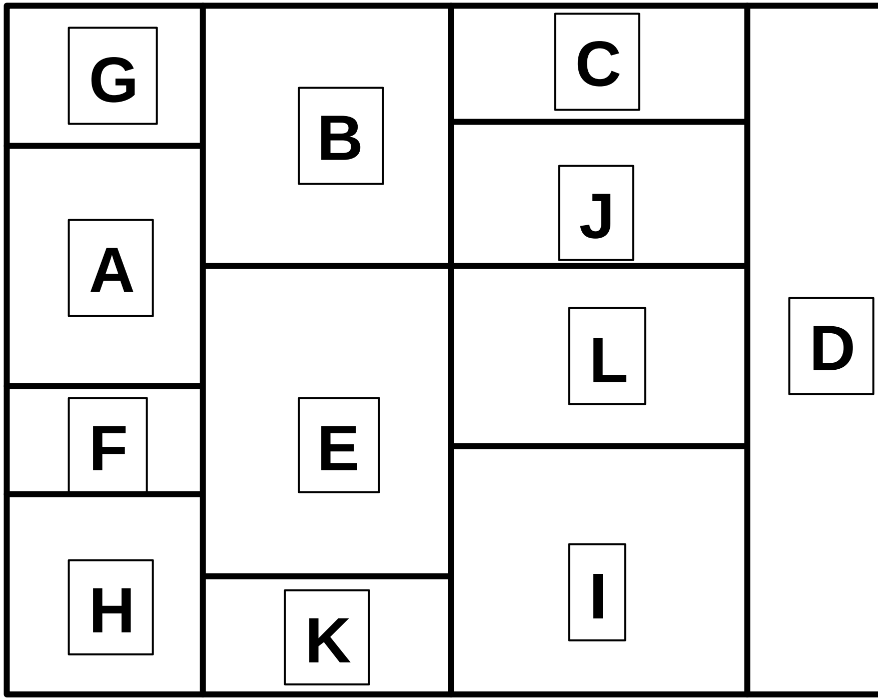
B	F	A	C	E	D	parent 1
E	F	C	A	B	D	parent 2
B	F	C	C	B	D	child
B	F	A	C	B	D	repair

Repair (cont.)

Repair is ineffective when:

- it is not obvious how to repair a solution to make it feasible
- it is very disruptive to the original solution
- it is computationally expensive
- most solutions have to be repaired

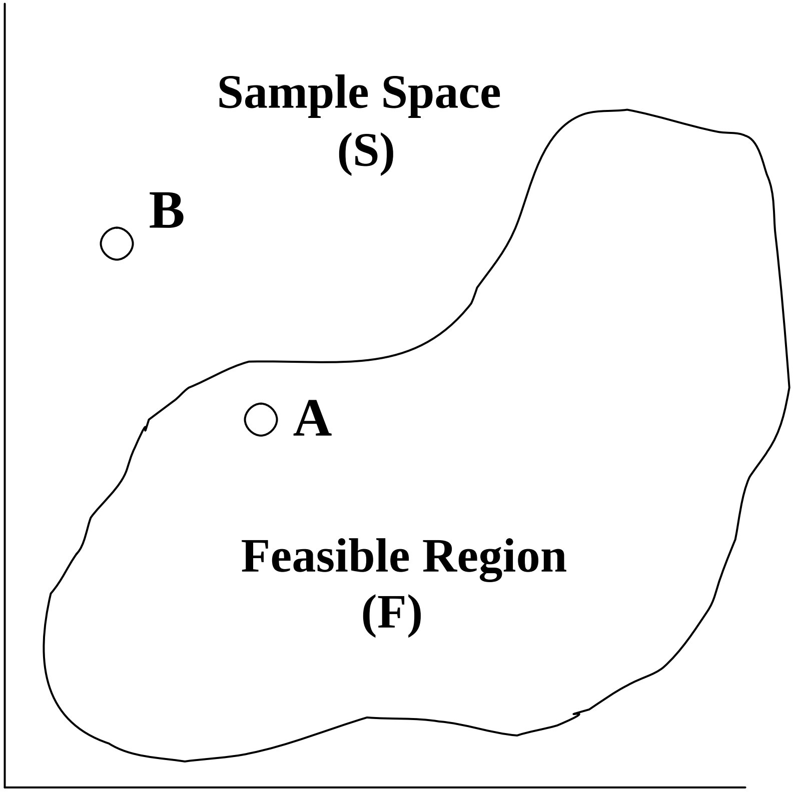




Department D is too long and narrow - how can this repaired to meet a maximum aspect rati constraint?

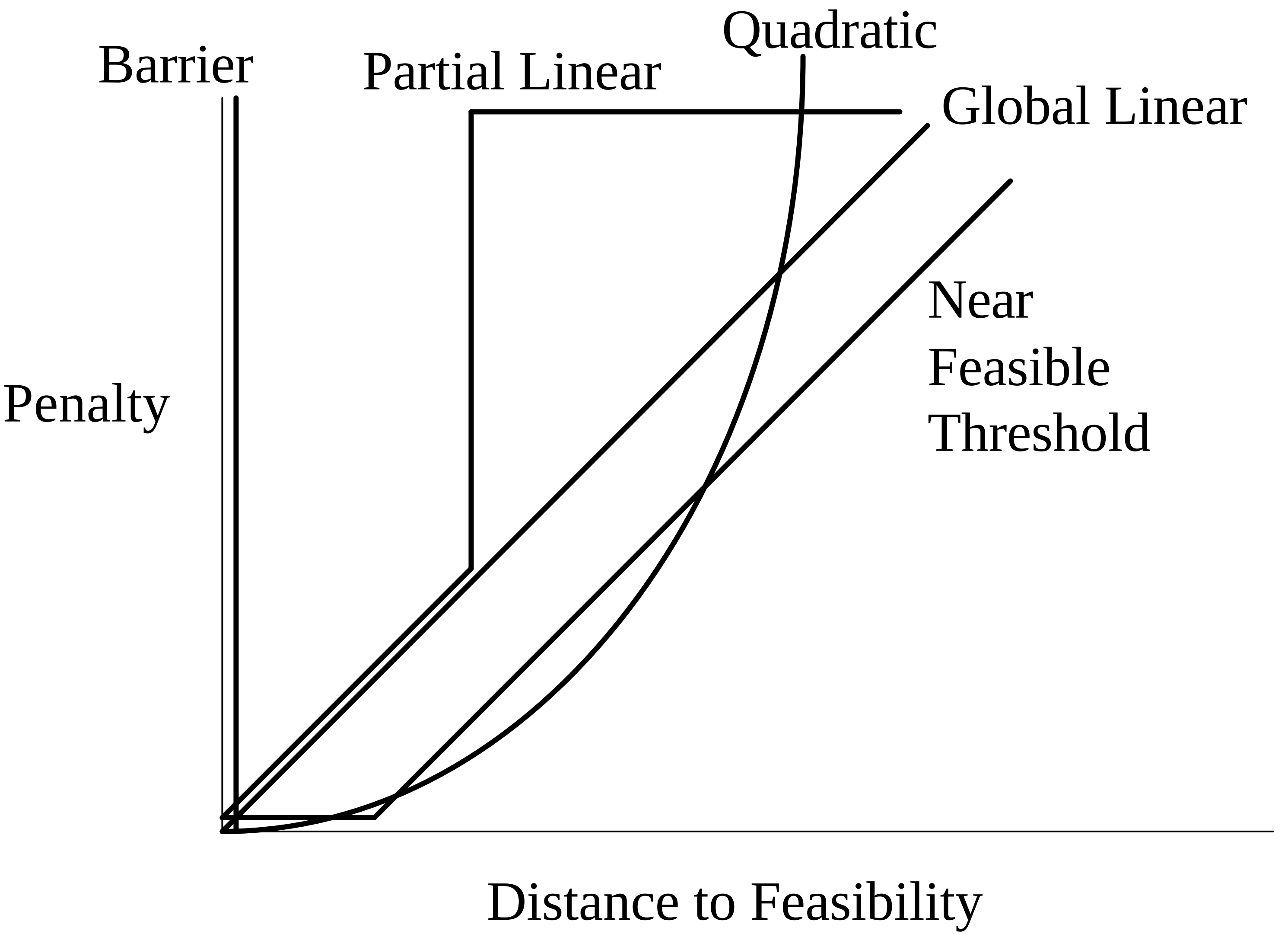
Penalizing Infeasibles

- Interior - *optimality*
- Exterior - *feasibility*
- Metrics:
  - number of constraints violated
  - weighted violations
  - distance to feasibility
    - linear
    - non linear



How do I compare A and B?

Some Distance Strategies

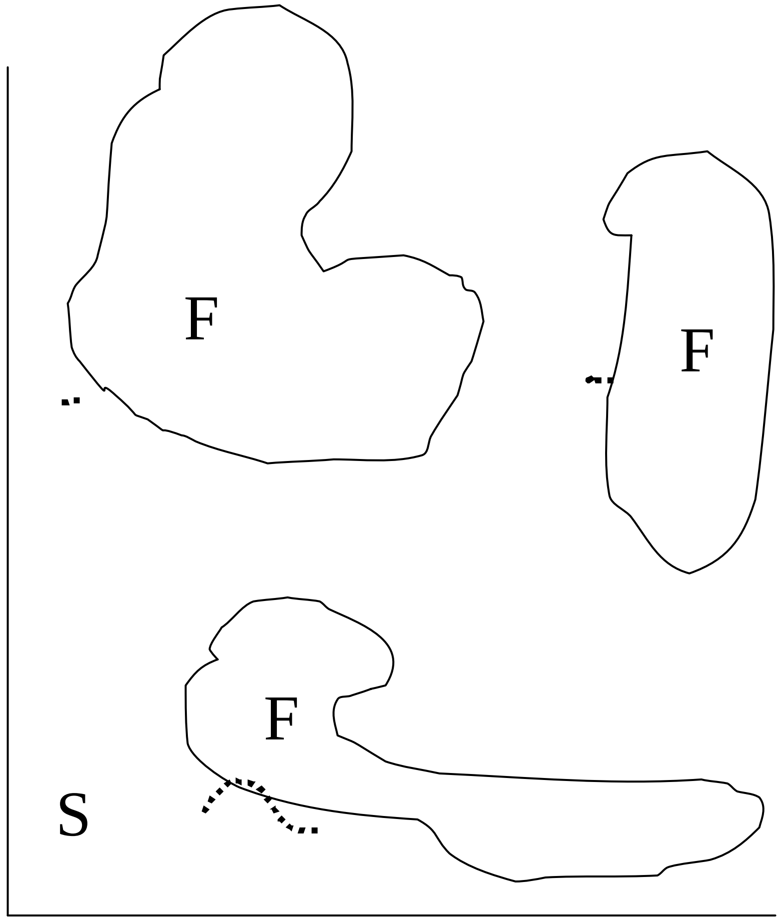


Desirable Properties of a Penalty Function

- ☺Thorough search of promising feasible and infeasible regions
- ☺Results in final feasible optimal solution
- ☺Scales multiple constraints
- ☺Works for all constraint levels - loose to tight
- ☺Is easy to calculate
- ☺Has intuitive interpretation

A Good Penalty Function Can:

- Concentrate search on the border between feasibility and infeasibility
- Identify disparate regions of superior feasible solutions
- Provides insight to relative difficulty of multiple constraints

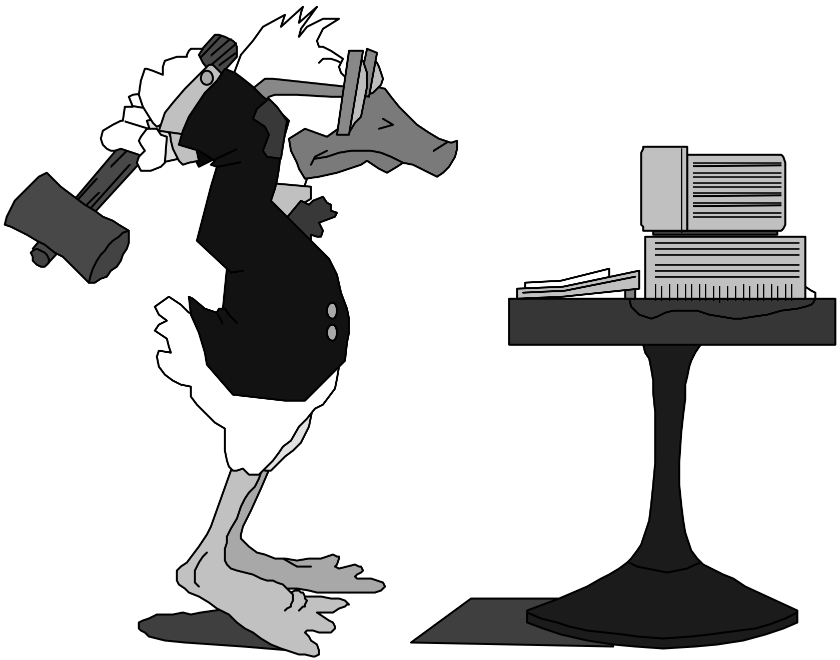


Good Penalty Methods for Adaptive Optimization

- ☺ Adding a dynamic aspect - generally increasing the penalty as the search progresses
- ☺ Adding an adaptive aspect - incorporate information about solutions already found or current regions of search into the penalty
- ☺ Evolving multiple populations for multiple constraints or constraint levels

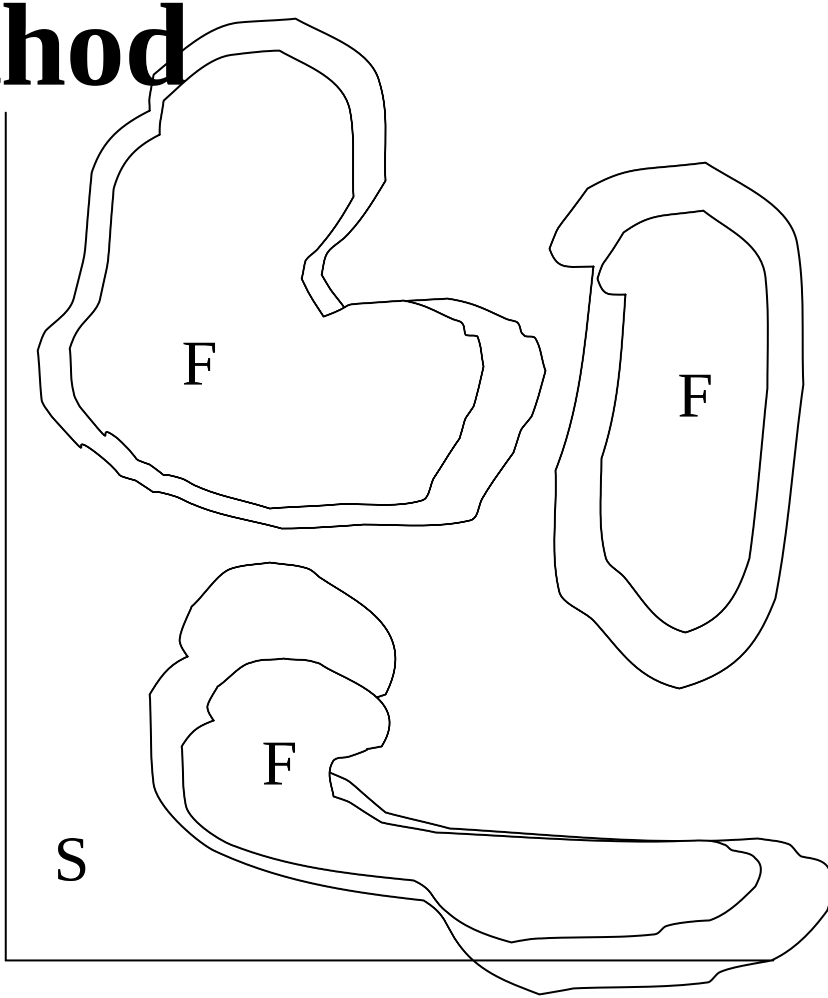
Ineffective Penalty Methods

- ☹ Many tuneable parameters and highly sensitive to these parameters
- ☹ Stalls in the interior of feasibility or too far from the feasible region
- ☹ Cannot handle multiple constraints
- ☹ Provides poor discrimination among infeasible solutions



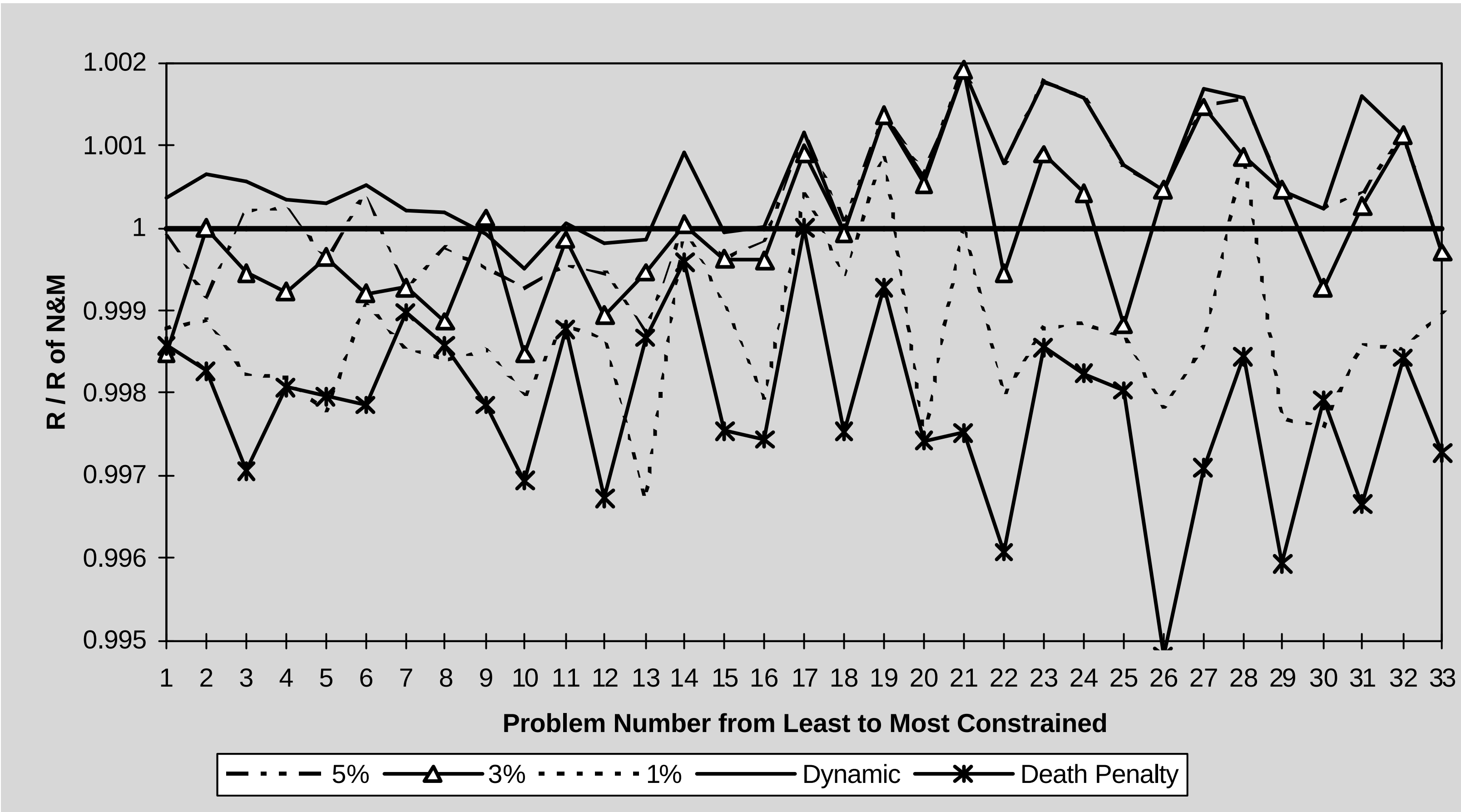
NFT Method

- Encourages search of the infeasible region within the Near Feasibility Threshold
- Adapts to search history and self scales constraints
- NFT can be static, dynamic or adaptive



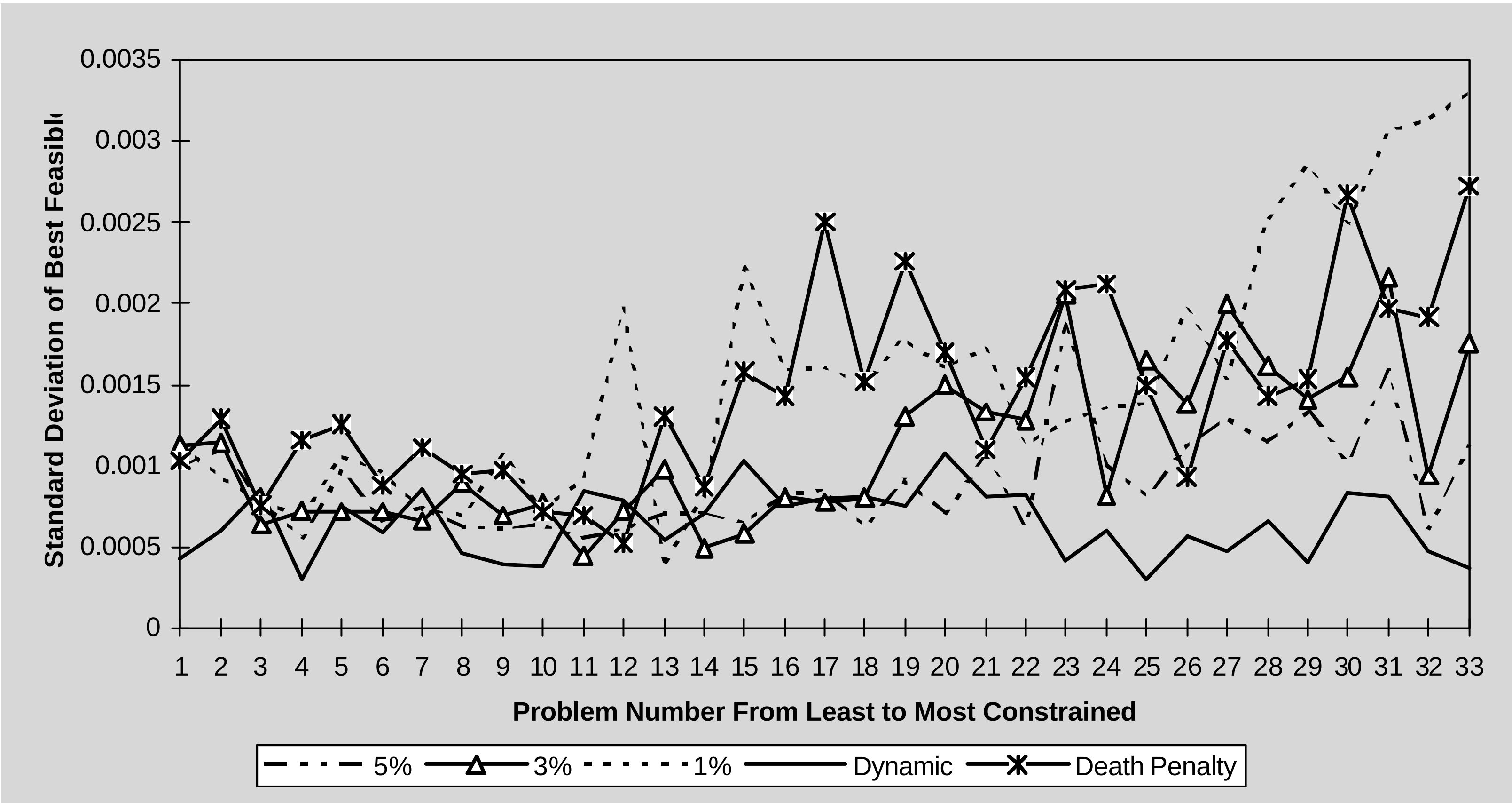
$$F_p(\mathbf{x}) = F(\mathbf{x}) + (F_{feas} - F_{all}) \times \sum_{i=1}^{N} \left(\frac{d_i(\mathbf{x}, \mathrm{B})}{\mathrm{NFT}_i} \right)^{K_i}$$

Results Comparing Death Penalty, Static Penalty, Dynamic Penalty



From *Computers & IE*, 1996, reliability design problem.

Results Comparing Death Penalty, Static Penalty, Dynamic Penalty

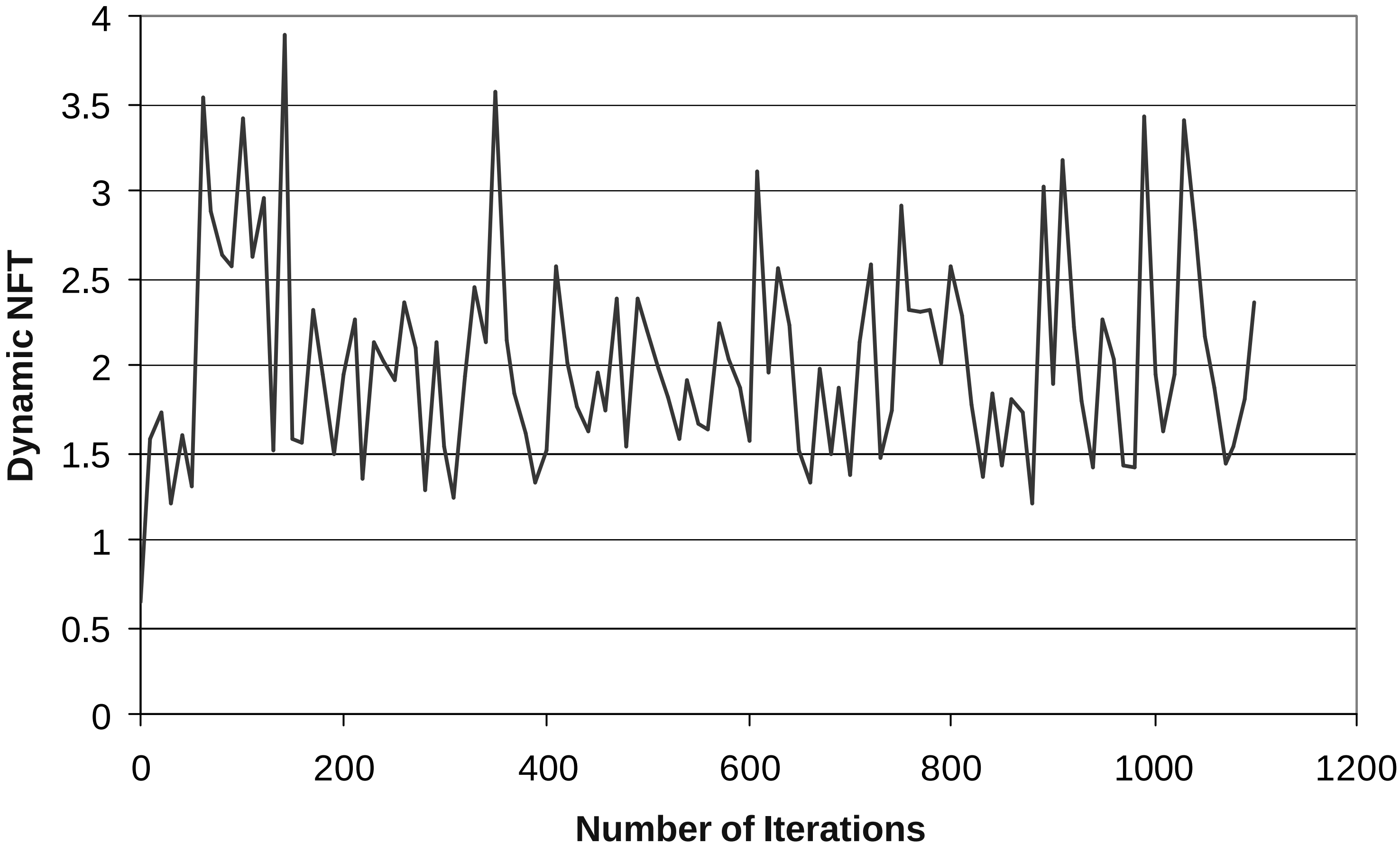


From *Computers & IE*, 1996, reliability design problem.

Adaptive NFT for Tabu Search

- General form $$F_p(\mathbf{x}) = F(\mathbf{x}) + (F_{feas} - F_{all}) \times \sum_{i=1}^{N} \left(\frac{d_i(\mathbf{x}, \mathrm{B})}{\mathrm{NFT}_i} \right)^{K_i}$$
- Plant layout design with a constraint on department aspect ratio $$F_p(\mathbf{x}) = F(\mathbf{x}) + (F_{feas} - F_{all}) \left(\frac{n}{\mathrm{NFT}} \right)^{K}$$
- where NFT changes according to both current move and tabu list:
  - if most moves are feasible, increase NFT
  - if most moves are infeasible, decrease NFT

NFT Over Search



Original GA used a static NFT of 1 or 2.

Other Effective Approaches

- With multiple constraints, alternate the constraint in the objective function or apply different constraint levels to different groups within the population

- For hard and soft constraints, severely penalize the hard constraints and lightly penalize the soft constraints. Consider both feasible and slightly infeasible solutions at the end.

Concluding Comments

- Handling constraints requires special care with adaptive optimization
- It is often better to consider infeasible solutions during search
- Population based search methods can be used advantageously for multiple constraints and for hard/soft constraints

Check out the following:

"Constraint Handling Techniques" (Chapter C5) in *Handbook of Evolutionary Computation*, 1997, IOP and Oxford University Press.