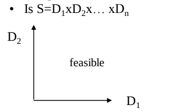
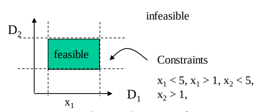
EAs and Constraints

- Constrained problem definition
  - Suppose a problem has n variables $x_1, x_2, \ldots x_n$
  - A feasible solution is any choice of the x_i which satisfies a set of conditions (or constraints)
  - x_i drawn from a set D_i
    - Discrete values (finite set)
    - Continuous data
  - Search space (S) of solution
    - Is $S=D_1 x D_2 x \ldots x D_n$



Free Search space for n=2

- Many practical problems are constrained
- Many constrained problems are NP-hard or NP-complete
- Handling constraints not straightforward with an EA
  - EA operators are 'blind'
  - Generate infeasible as well as feasible solutions



Constrained Search space for n=2

Types of problem

- Free search space
  - Each point can be tested for membership independently
  - Feasible search space is conjunction of points satisfying the condition
- Objective function and constraints
  - Can be used to define a search problem
  - FOPs are naturally solved by EAs since the objective function is also a fitness function
- Solve COPs and CSP by
  - Mapping to a FOP
  - Can use direct or indirect constraint handling

	Objective function	
Constraints	Yes	No
Yes	Constrained Optimisation problem (COP)	Constraint Satisfaction problem (CSP)
No	Free Optimisation Problem (FOP)	?

<u>Examples:</u>

FOP: find the max of some function y=f(x)

COP: find the max of f(x1, x2) in the interval 1≤x1≤n, 1≤x2≤n

CSP: find a colouring of a graph G(V,E) such that adjacent nodes have different colours and only three colours are used in total

Handling Constraints (1)

- Direct handling
  - Generate a phenotype from the genotype
  - Check all constraints satisfied (i.e. solution is feasible)
  - Use optimization function to choose best feasible solution
- Indirect handling
  - Convert each constraint to an objective
  - Combine objectives using penalty functions
  - Problem is now a FOP and can ignore explicit constraint handling
  - Conversion is done before solving the problem with an EA

<u>Example:</u>

<u>COP:</u> find the max of f(x1, x2) in the interval 1≤x1≤n, 1≤x2≤n (assume non-negative for simplicity)

Convert by defining:

f1(x1) = if (1≤x1≤n) then 1 else 0

f2(x2) = if (1≤x2≤n) then 1 else 0

<u>FOP :</u> fnew(x1, x2)= f(x1,x2)-w1*f1(x1)-w2*f2(x2)

Where w1 and w2 are weights chosen to give infeasible point in solution space poor fitness

<u>Exercise:</u> formulate the Sudoku grid problem as a CSP and convert it into a FOP

Handling Constraints (2)

- Ideally only want to generate feasible solutions
  - Avoid time spent in unprofitable areas of search space
  - Focus on 'good' solutions
  - Can use domain specific knowledge to help eliminate infeasible possibilities

Usually the feasible set (F) is much smaller than the search space (S)

An interior point is on inside F and exterior point outside F

- Four basic mechanisms
  - Use of penalty functions
    - Fitness reduced in proportion to number of constraints violated
  - Repair an infeasible soln
    - Convert it into a feasible soln
  - use an alphabet and operators to reduce chances of infeasible soln
    - Create an unambiguous mapping from genotype to phenotype
  - Always generate feasible soln
    - Decode each genotype so that phenotype is always feasible
    - Many to one mapping of genotype to phenotype

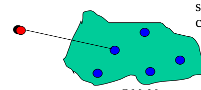
Handling Constraints (2)

- More on penalty functions
  - Popular and easy to use
  - Can work on disjoint Feasible regions
  - Useful when global optimal is close to boundary of feasible region
- Success depends on
  - Balance of exploring infeasible region and not wasting time
  - Severity of penalties for violating constraints
- Can use a distance metric
  - Indicates how far infeasible point is from boundary of F
- Severe penalties
  - Infeasible point near boundary of F discarded
  - Can slow down the search
- Weak penalties
  - Infeasible points can dominate feasible solution
  - Algorithm can stagnate
- Penalty function types
  - Static
    - Choose weights by experimentation
    - Distance metric (Boolean or Euclidean distance)
  - Dynamic
    - Allow weight penalties to vary with time
  - Adaptive
    - Learn the weights as algorithm runs

Handling Constraints (3)

- Repair functions
  - Takes an infeasible point and produces a feasible point
  - For example using a local search to reduce constraint violations
- Repair type
  - Baldwinian
    - Fitness of repaired point is allocated to infeasible point
  - Larmarkian
    - Infeasible point is overwritten by the feasible point
- Michalewicz's repair method
  - Maintain two populations P_s and P_r
  - P_s are search points
  - P_r are reference points (i.e. inside F)
  - when an infeasible point is created the nearest reference point is located
  - A 'line' is drawn between the infeasible point and the reference point
  - A search (or averaging) scheme is used to find a point close to the boundary of F

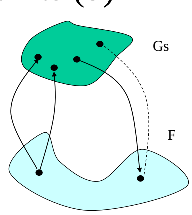
Blue in P_r
Black in P_s
Red repair



Feasible region F

Handling Constraints (3)

- Decoding functions
  - Map the genotype space (G_s) to feasible set
  - Thus every phenotype is guaranteed to produce a feasible solution
  - Not always possible depends on problem being solved
- Decoder function properties
  - Every g in G_s must map to a point s in F
  - Every solution s in F must have at least one representation g in G_s
  - Every s in F must have the same number of representations in G_s (not necessarily one)



Example : knapsack problem: Let genotype be binary string of objects to be taken. Scan left to right including an item if gene allele is one. Ignore rest of string as soon as a constraint is violated

Example : three colouring a graph

- Representation
  - Let the graph be G(N,E) where
    - N finite set of n>0 nodes
    - E set of edges
  - Genotype
    - Let G be a vector of length n
    - each element takes the values 1, 2, or 3
  - Adjacency matrix
    - Let C be an n by n matrix
    - C[i,j]=1 if an edge between nodes i and j in graph
    - C[i,j]=0 if no edge between nodes i and j
- FOP formulation
  - Define the penalty function as blow
  - Maximise 1/eval
  - Optimal answer is 1

```
Function eval(G:genotype): fitness;
{ penalty := 1; // to avoid divide by zero
  for i := 1 to n do
    for j := 1 to n do
      if C[i,j] <> 0 then
        if G[i] = G[j] then
          penalty := penalty +1 ;
  return eval = penalty;
}
```

Example : Sudoku grids

- Representation
  - Genotype
    - G is an n by n array
    - G[i,j] has takes a value 1..9
  - Suppose the existence of functions
    - Row(G, i) sums the values in row i of G
    - Col(G,j) sums the value in col j of G
    - Block(G,i,j) recovers and sums the block (G, i,j) on the grid
  - Assume a standard grid
    - n=9 and there are nine 3 by 3 blocks
    - Problem specific knowledge – a valid row, col, or block sums to 45

- Conversion to FOP
  - Penalty function (see below)
  - Maximise 1/eval
  - Optimal answer is 1;

```
Function eval(G:genotype): fitness;
{ penalty := 1; // to avoid divide by zero
  for i := 1 to n do
    if Row(G, i) <> 45 then penalty := penalty +1;
  for j := 1 to ndo
    if Col(G, j) <> 45 then penalty := penalty+1;
  for i:= 1 to 3 do
    for j := 1 to 3 do
      if Block(G, i, j) then penalty := penalty +1 ;
  return eval = penalty;
}
```

Question: does this eval function always work?
Can you suggest and alternative?