Constrained Optimisation

• Minimise \( f(x) \) subject to constraints \( c_i^T x \leq d_i \)
  – Linear Programming problem
  – Quadratic Programming problem

Linear Programming

• Aim to maximise \( f(x) = g^T x \)
  subject to primary constraints: \( x_i \geq 0 \)
  the equality constraints: \( a_i^T x = b_i \)
  and inequality constraints: \( c_i^T x \leq d_i \)

Solution lies on constraint line
Use ‘Simplex’ algorithm

Quadratic Programming

• Aim to minimise \( f(x) = 0.5x^T Hx + g^T x \)
  subject to linear equality constraints:
  \( a_i^T x = b_i \quad i = 1..t \)
  and inequality constraints:
  \( a_i^T x \leq b_i \quad i = t+1..t+d \)

Unconstrained QP

Minimise \( f(x) = 0.5x^T Hx + g^T x \)
\[ \frac{df(x)}{dx} = Hx + g \]
Solution is \( x = -H^{-1}g \)

QP with Equality Constraints

Minimise \( f(x) = 0.5x^T Hx + g^T x \) subject to \( Ax = b \)

Lagrange function: \( f_\lambda(x) = 0.5x^T Hx + g^T x + \lambda^T (Ax - b) \)
\[ \frac{df_\lambda(x)}{dx} = Hx + g + A^T \lambda = 0 \quad Hx + A^T \lambda = -g \]
\[ \begin{pmatrix} H & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} -g \\ b \end{pmatrix} \]
Solution: \( x = -H^{-1}(g - A\lambda) \)
\( \lambda = -(AH^{-1}A)^{-1}(b + AH^{-1}g) \)

QP with Inequality Constraints

• Solve with equality constraints
• Test each inequality
• If any broken, move as far as possible and add broken inequality as an equality
• Repeat until all satisfied
QP with Inequality Constraints

• Assume we have an initial feasible $x$
  – One that satisfies all the constraints

Minimise $f_d(x) = f(x + d) = 0.5(x + d)'H(x + d) + g'(x + d) \quad \text{subject to } A(x + d) = b$

Minimise $f_d(x) = 0.5d'HD + (Hx + g)'d \quad \text{subject to } Ad = (b - Ax)$

$x \to x + \alpha d, \quad \alpha \text{ chosen so inequalities satisfied}$

Global Search Methods

• Dynamic Programming
• Simulated Annealing
• Genetic Algorithms
• Estimation of Distribution Algorithms

Dynamic Programming

• Powerful, fast technique for solving a particular class of problems

$$f(x) = \sum_{i=1}^{n} g_i(x_i) + \sum_{i=1}^{n} h_i(x_i, x_{i-1})$$

$x_i, i = 1..n \text{ are discrete variables}$
$x_i \text{ can have } n_i \text{ different values (states)}$

$g_i(x_i)$ is the cost of $x_i$ being state $x$
$h_i(x_i, y)$ is the cost of $x_i$ taking state $x$ and $x_{i-1}$ taking state $y$

Typical Problem

• Find cheapest smooth path through a grid

Cost associated with each square, $c_{ij} = g_{ij}'(x_i - x_j)^2$

To encourage smoothness $h_i(x_i, x_{i-1}) = \alpha(x_i - x_{i-1})^2$

Dynamic Programming

$$f(x) = \sum_{i=1}^{n} g_i(x_i) + \sum_{i=1}^{n} h_i(x_i, x_{i-1})$$

$x_i$ can have $n_i$ different values (states)

Total number of possible states of system $= \prod_{i=1}^{n} n_i$

DP can find global solution in approx. $\sum_{i=1}^{n} n_i n_{i-1}$ steps

If $n_1 = 10$ and $n = 20$, then that’s $= 2000$ vs $10^{20}$

Incremental solution

• DP takes advantage of the fact that the problem can be broken down into parts

Suppose for some $k \leq n$ we have the total cost of the cheapest path ending at each value of $x_k, T_k(x_k)$

$$T_k(x_k) = g_k(x_k)$$
Incremental solution

\[ T_{k+1}(s_{k+1}) = g_k(s_{k+1}) + \min_{j=1..n_k} [T_j(j) + h_k(j, s_{k+1})] \]

\[ T_f(s_f) = g_f(s_f) \]

Finding solution

- Backtrace to find the best solution

\[ s_{k_{opt}} = \arg \min_{j=1..n_k} [T_j(j)] \]

for \( k = n-1 \ldots 1 \)

\[ s_{k_{opt}} = p_{k-1}(s_{k_{opt}}) \]

DP Algorithm

Temporary storage:

Vectors: \( T_i(j) \quad k = 1..n, i = 1..n_k \)

Int vectors: \( p_i(j) \quad k = 2..n, i = 1..n_k \)

\[ T_f(i) = g_f(i) \quad i = 1..n_k \]

for \( k = 1..n-1 \)

\[ j_{min} = \arg \min_{j=1..n_k} [T_j(j) + h_k(j, i)] \]

\[ T_{k+1}(i) = g_k(i) + T_j(j_{min}) + h_k(j_{min}, i) \] (Record cost)

\[ p_{k+1}(i) = j_{min} \] (Record path to best previous state)

Simulated Annealing

- General purpose optimisation method
  - Minimise \( f(x) \)
  - Stochastic approach
  - Try random perturbations
    - Always accept better solution
    - Sometimes accept worse solution
  - Can jump out of local minima

Simulated Annealing

- Initialise \( s_i(t) = 0 \)
- Repeat
  - Generate random displacement \( dx \)
  - If \( f(x) + dx < f(x) \) then \( s_{i+1} = x + dx \)
  - else
    - If \( \text{rand}(0,1) \leq \frac{e^{(f(x) - f(x + dx))}}{T(t)} \) then \( s_{i+1} = x + dx \)
    - else \( s_{i+1} = x \)
  - Until happy
Population Based Methods

• Previous methods only consider one point at a time
  – Can’t deal well with multiple minima
• Alternative approach:
  – Retain many possible solutions
  – Can investigate many local minima
  – More likely to locate true global minima

Genetic Algorithms

• Originally developed as an analogy to evolution – survival of the fittest
• General approach
  – Initialise large population of points
  – Repeat
    • Combine (‘breed’) good points
    • Discard poor points
  – Until convergence
• Population should evolve toward good solutions

Genetic Algorithms

Key issues

• Method of encoding parameters
• Method of generating offspring
• Size of population

Encoding and ‘crossover’

Parents

\[
x_1 = \begin{barray}{c} \text{0} \text{0} \text{1} \text{0} \text{1} \text{0} \text{1} \text{1} \text{1} \end{barray} \quad x_2 = \begin{barray}{c} \text{1} \text{1} \text{0} \text{0} \text{1} \text{1} \text{0} \text{0} \text{0} \end{barray}
\]

Cut point, Randomly chosen

Offspring

\[
x_1' = \begin{barray}{c} \text{1} \text{1} \text{1} \text{0} \text{0} \text{1} \text{1} \text{0} \text{0} \text{0} \end{barray} \quad x_2' = \begin{barray}{c} \text{0} \text{0} \text{1} \text{0} \text{1} \text{0} \text{1} \text{1} \text{1} \text{1} \end{barray}
\]

GA

• Initial random population \( \{x_i^{(s)}\} i = 1..m \)
• \( j = 0 \)
• Repeat
  – Evaluate \( f(x) \) and sort so that \( f(x_i^{(s)}) \leq f(x_j^{(s)}) \)
  – Generate new population \( \{x_i^{(s+1)}\} i = 1..m \)
    • Randomly select pairs and apply crossover
    • Randomly select examples and \textit{mutate} them
• Until happy

When selecting example, give preference to lower fits

GAs

• Rich area of research
  – Method of encoding
  – Crossover methods
  – Mutation methods
  – Population size
  – ‘Speciation’ to encourage multiple solutions
Estimation of Distribution algorithms (EDAs)

- Initial estimate of distribution \( p_0(x) \) \( j = 0 \)
- Repeat
  - Generate \( m \) samples from \( p_j(x) \) : \( \{x^{(i)}\} \ i = 1..m \)
  - Evaluate \( f(x) \) for each
  - Discard worst 50%
  - Estimate pdf of remainder: \( p_{j+1}(x) \) \( j = j + 1 \)
- Until happy

Modes of distribution should converge toward local minima of \( f(x) \)

EDAs

- Performance is dominated by
  - form of PDF
  - accuracy of parameter estimation
- If expensive to evaluate \( f(x) \), it is worth using quite complex PDFs

Example on Quadratic Function

Use 2D Normal PDF, 20 samples

After 1 iteration
After 2 iterations
After 3 iterations
After 4 iterations
After 5 iterations
After 9 iterations

EDA on wiggly function

Use 2D Normal PDF, 20 samples

After 1 iteration
After 3 iterations
After 5 iterations
After 7 iterations
After 9 iterations
After 11 iterations