High Performance Computing Systems

- Metaheuristic Optimization

Doug Shook via Roger Chamberlain

Optimization

- We have some design we want to optimize
  - Set of parameters
  - Cost function
- Can be done by brute force
  - Especially if the problem is "nice"
    - Linear
    - Convex
    - Continuous
- Can usually guarantee optimization
  - Or at least put a bound on it

- Not always so straightforward...

Metaheuristic Optimization

- What if the list of parameters is large?
- What if the parameters / cost don't behave well?

- Can still attempt to solve these using brute force...

- Metaheuristic Optimization good for these cases
  - Can't guarantee optimization
  - Works well in practice, however
  - Metaheuristic – not problem dependent
- Sometimes "optimal" isn't necessarily the goal

Example: Place and Route

General Approach

start at some random configuration
repeatedly consider perturbations ("moves")
accept some and reject some
until you decide that you are done

- Algorithms themselves are pretty simple
- Important / interesting parts
  - Design of moves
  - How much to accept / reject

Hill Climbing

```
X <- init configuration
while not done
  E <- eval(x)
     N <- movesetsize(x)
     for i = 1 to N
          E_i <- eval(move(x, i))
     if all E_i > E terminate
     else i <- argmin(E_i )
     x <- move(x, i)
end while
```

Simulated Annealing

X <- init configuration
E <- eval(x)
while not done
    i <- random neighbor of x
    E_i <- eval(i)
    if E_i < E accept
    else accept with some probability
    else reject
end while

Accept/Reject

- Accept with probability exp(-ΔE / T)
  - Where T is "temperature"

- Initially, T is high
  - Almost all moves accepted
- Finally, T is low
  - Only improvements accepted

- T_{new} <- αT_{old}, where 0 <= α <= 1

Genetic Algorithms

- Set of configurations (chromosomes)
  - Referred to as a generation
- Assess fitness (cost) of each configuration
- Create a new generation
  - Selection: choose two parents
  - Crossover: create children from parents
  - Mutation: induce random perturbation
  - Conditionally accept / reject

- Simulation of evolution – later generations should have higher fitness
- Finding the correct cost function is true goal

Particle Swarm

- Cost function:
  - N-dimensional space -> single real result
    - Solution expressed as a vector

Notation:

x_i: position of particle i

v_i: velocity of particle i

Maintain:

p_i: best position of individual particles

g: global best position

Particle Swarm

Update v_i for all i:

$$v_i \leftarrow \omega v_i + \phi_p r_p (p_i - x_i) + \phi_g r_g (g - x_i)$$

Where:

r is a uniformly generated RN

omega and phi are weights

Update x_i for all i:

$$x_i \leftarrow x_i + v_i$$

Update p_i, g:

p_i: best position of individual particles

g: global best position

Recap

- Provides an alternate approach to optimization
  - Sacrifice the guarantee of finding optimum
    - All the time algorithms
- Important design / tuning considerations:
  - Cost function
  - Moves
  - Weights
- No free lunch theorem

Parallelization

- Might think: choose different starting points
  - Problem: revisiting the same "state" is a waste
- Speculative parallelization (SA)
  - Compute ahead when acceptance is high
  - Middle ground becomes less parallel
  - Predict rejections when acceptance is low