Constraint Handling in GAs

- Most problems introduced in this course are constrained only by bound constraints defining the decision variable ranges → eg $x_i \in [0, 100]$
- There are many common optimization problems that are constrained optimization problems
- These can have linear and non-linear equality and inequality constraints eg $1 < x_i < 10$ or $1 < x_i^2 < 10$ or $x_i = 2$
- GAs are generic optimizers that only consider the fitness value and are 'blind' to constraints
- As a result, a wide variety of heuristic methods are used to apply GAs to constrained problems

Handout 10-7-2011

Constraint Handling in GAs

- The bulk of the remaining slides are based on an excellent paper by Deb (2000)
- Constraint handling techniques are classed into 5 methods:
  1. Based on preserving feasibility of solutions
  2. Based on penalty functions
  3. Based on distinguishing feasible and infeasible solutions
  4. Based on decoders
  5. Based on hybrid approach

↳ satisfies all constraint

↳ at least one constraint is violated.

Problem Statement and Notation for Constrained Nonlinear Optimization

cost function

$$\begin{aligned}
\text{Minimize} \quad & f_1(\vec{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \\
\text{Subject to} \quad & g_1(\vec{x}) \equiv 4.84 - (x_1 - 0.05)^2 - (x_2 - 2.5)^2 \geqslant 0, \\
& g_2(\vec{x}) \equiv x_1^2 + (x_2 - 2.5)^2 - 4.84 \geqslant 0, \\
& 0 \leqslant x_1 \leqslant 6, \;\; 0 \leqslant x_2 \leqslant 6.
\end{aligned}$$

constraints

input bounds

We will *also* parallel the above notation as follows:

nomenclature

$f(\boldsymbol{x})$ → objective (cost) function

$g_k(\boldsymbol{x})$ → the k[th] inequality constraint

$h_j(\boldsymbol{x})$ → the j[th] equality constraint

Constraint-handling methods presented assuming inequality constraints

$\boldsymbol{x} = [x_1\ x_2\ \ldots\ x_n]$ → vector of decision variables

$F(\boldsymbol{x})$ → the GA fitness function (i.e. we may modify the cost function $f(x)$ to deal with constraints)

Penalty Function Method

- Considers constraint violations in the fitness function (maximizing fitness)
- Constraint violations decrease the fitness usually by using the following form:
- where

$$F(\boldsymbol{x}) = f(\boldsymbol{x}) - \sum_{j=1}^{J} R_j \left\langle g_j(\boldsymbol{x}) \right\rangle^2$$

use in place of cost function; original cost function; sum penalty for all constraints; scaling factor; constraint; function not important but we do want absolute value; sign implies maximization

R_j is the penalty coefficient (weight) for inequality constraint j (of J total)

$\langle g_j(\boldsymbol{x}) \rangle = |g_j(\boldsymbol{x})|$ for $g_j(\boldsymbol{x}) < 0$ (definition)

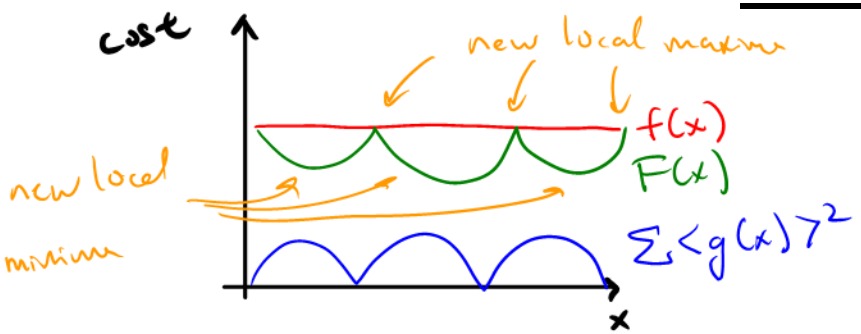
$= 0$ for $g_j(\boldsymbol{x}) \geq 0$

- R_j serves to make the penalty function the same order of magnitude as the objective function
- Often, constraints are normalized and a single R value used

Problems with Penalty Function Method

more parameters → more knobs → more tuning → more work → more time

- Introduces at least one but sometimes more algorithm parameters (the R_j) to tune
- Possible experiments needed to determine penalty coefficient values R_j that guide the algorithm towards feasible region
- Distorts the objective function and can even introduce <u>additional</u> local optima



Alternate Method for GA Constraint Handling

- Method proposed by Deb (2000)
- **Parameter free** penalty function method that considers feasibility of solution
- Takes advantage of 1:1 comparison of solutions in Tournament Selection
- Very simple!

Efficient GA Constraint Handling

Deb (2000) Method

In tournament selection, when comparing two solutions together, the following criteria are always enforced:

1. Any feasible solution is preferred to any infeasible solution. → choose feasible over infeasible
2. For two feasible solutions, the one with the higher fitness (better objective) is preferred. → original algorithm
3. For two infeasible solutions, the one having a smaller constraint violation is preferred. → pick the one that violates the constraint the least

New to tournament

If the GA initial population only contains feasible solutions, can the GA generate infeasible solutions in future generations?

Deb, Kalyanmoy, *Compt. Methods Appl. Mech Engrg* 186 (200) 311-338

Deb's GA Constraint Handling

- Fitness function defined as:

$$F(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & \text{if } g(\boldsymbol{x}) \not\leq 0 \text{ for all j=1, 2, ... } J, \\ f_{\min} - \sum_{j=1}^{J} \langle g_j(\boldsymbol{x}) \rangle & \text{otherwise} \end{cases}$$

where

  - $f_{\min}$ is the fitness value of the worst feasible solution found so far
  - and all constraints, $g_j(\boldsymbol{x})$, are normalized as follows:

    for $g_1(\boldsymbol{x}) \equiv 13 - x_1 x_2 \geq 0$

    normalize to: $g_1(\boldsymbol{x}) \equiv 1 - x_1 x_2 / 13 \geq 0$

    Make all constraints: $***** \geq \mathbf{0}$

  Note there is no weighting coefficient R_j

Efficient GA Constraint Handling: Simply Computed Constraints

- There are constraints that can be quickly computed without computing the cost function (e.g. $x_1+x_2 \leq 10$)
- Check these constraints first and DO NOT evaluate objective function if they are violated (differs from penalty function method) ⇒ save on computation!
- Makes practical sense
- Particularly beneficial for computationally demanding cost functions
- However, for many cases it is necessary to do expensive calculations to calculate the constraints *g(x)* along with the cost function *f(x).* We use the next method for that situation.

Constraint Handling for Other Algorithms

yes. all you're doing is changing the cost function

- Does the penalty function approach work for other heuristics like greedy search, simulated annealing?
  - Yes but method has same pitfalls as described earlier → proper selection of weighting R_j of penalty functions required

- Could the approach proposed by Deb (2000) be applied to Simulated Annealing (SA)?
  - Force SA algorithm never to accept a move to infeasible region assuming it started in feasible region.

Equality Constraints

If direct substitution is possible:

The best way handle equality constraints is to get rid of them if possible by substitution. For example if $x_1=x_2+5$, then just replace the variable x_1 in the cost function with x_2+5 wherever x_1 appears.

If direct substitution is not possible:

With Deb's method you can also use the evaluation of the equality directly to determine if a solution is feasible or infeasible even if direct substitution is not possible.

However to use the penalty function approach or to use Deb's solution for selecting between two infeasible solutions, you can use the following approach:

One way to handle these is to put them in this form:

$h(\boldsymbol{x}) \equiv \sin x_1 * x_2 / (x_1^4 + \text{Max}(x_1 * x_2, x_1 + x_2 + 5)) = 0$ so let g(x) be absolute value:

•:

$$g(\boldsymbol{x}) \equiv | \sin x_1 * x_2 / (x_1^4 + \text{Max}(x_1 * x_2, x_1 + x_2 + 5)) | \geq 0$$

•

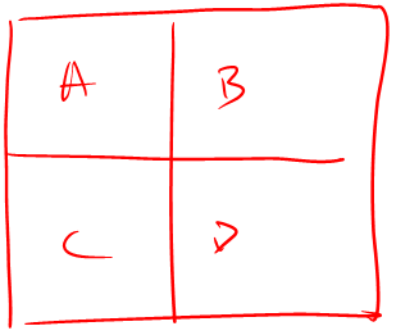
- In general:

$$g(\boldsymbol{x}) \equiv | h(\boldsymbol{x}) | \leq 0$$

•

Partitioning A Method for Maintaining Diversity in the GA

- ~~Process~~ Purpose of a population to continuously and simultaneously search multiple promising regions of the search space
- ~~Helps~~ Want to combat premature convergence of the population
- The idea is to allow selection to occur simultaneously in subpopulations and then periodically mix the populations in some way.



search space

1. start 4 runs of GA, each starting in one of the quadrants
2. Run the GA's for k generations
3. Mix best individuals for every k generations

this method is sometimes called "islands" to preserve the evolution analogy

Parallel/Distributed Computing and Heuristic Methods

- For years parallel computing was just done on supercomputers largely at US government research laboratories and universities.
- Now parallel computing is everywhere, including even in your cell phone.
- We will discuss how computing in parallel affects efficiency and how this improves the relative advantages of heuristics over many other methods.

- ***What is Parallel (Distributed) Processing?***
- Example: Suppose you wanted to order a deck of cards in order (Clubs 2,3,4,…., King, Ace; diamonds 2,3,…,Ace: Hearts 2,…,Ace; Spades 2,3,…Ace).
- How would you do this if you had two people each at a separate table? Would it be faster than if one person did it? → probably not. There's the merge sort afterwards
- What if a large number of people (each at separate table) did the sorting (with a "runner" to do the communication between tables). Is it faster? How much faster?

- Another concept in parallel processing is “Speed UP”, which is a measure of how much faster is the computation when you use more processors. (In the deck example, each person is a computer processor.)
- What is the speed up for the card example?

- What’s an equation for speed-up?

$$\frac{t_{parallel}}{t_{serial}}$$

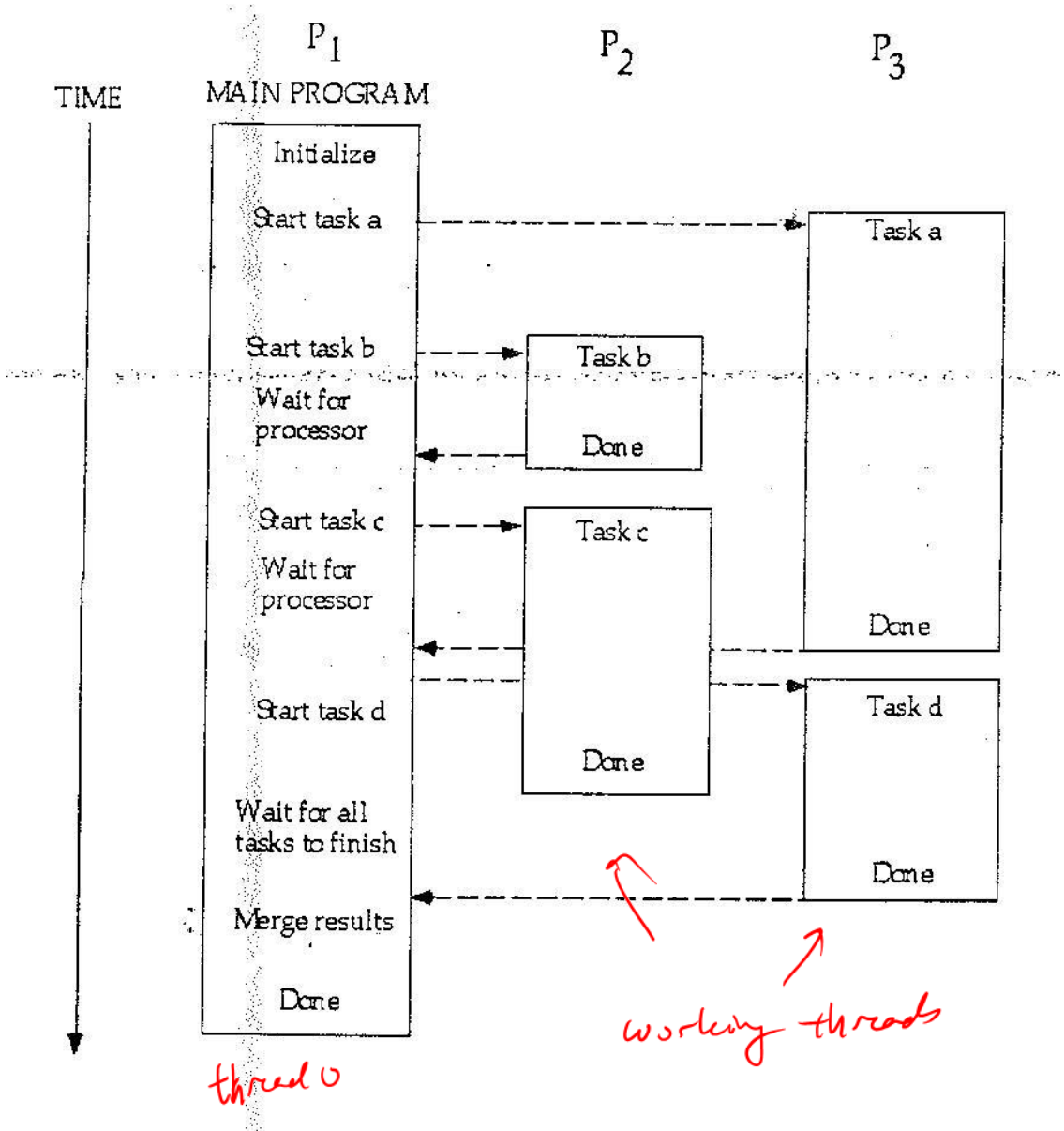
- What is the difference between Serial and Parallel Processing?

- What are the issues determining the efficiency for this problem of doing the work with “parallel processors” (people in this case)?:

$\Rightarrow t_{computation} >> t_{communication}$ to avoid comm. overhead

- Let us generalize these concepts to parallel (distributed) computing, where there are a number of individual processor doing calculations and communicating with each other.
- **Walk Clock Time** How long does it take to compute the solution using N processors.
- **Speed Up** How much faster is the solution with N processors than with one processor.
- (Speed Up = (Serial Time)/(Walk Clock time with N Processors)
- *In a perfect parallel algorithm the speed up for N processors would be = _____?*

Example of Work under Parallel Conditions



Issues in Selecting Serial Versus Parallel Processing:

- Do we expect the use of Parallel Processing to increase or decrease in coming years? Why or why not? increase: many core trend
- Which kinds of problems can be effectively computed with parallel processing?

Can you efficiently compute the answer to the following questions using parallel processing?:

→ embarrassingly parallel

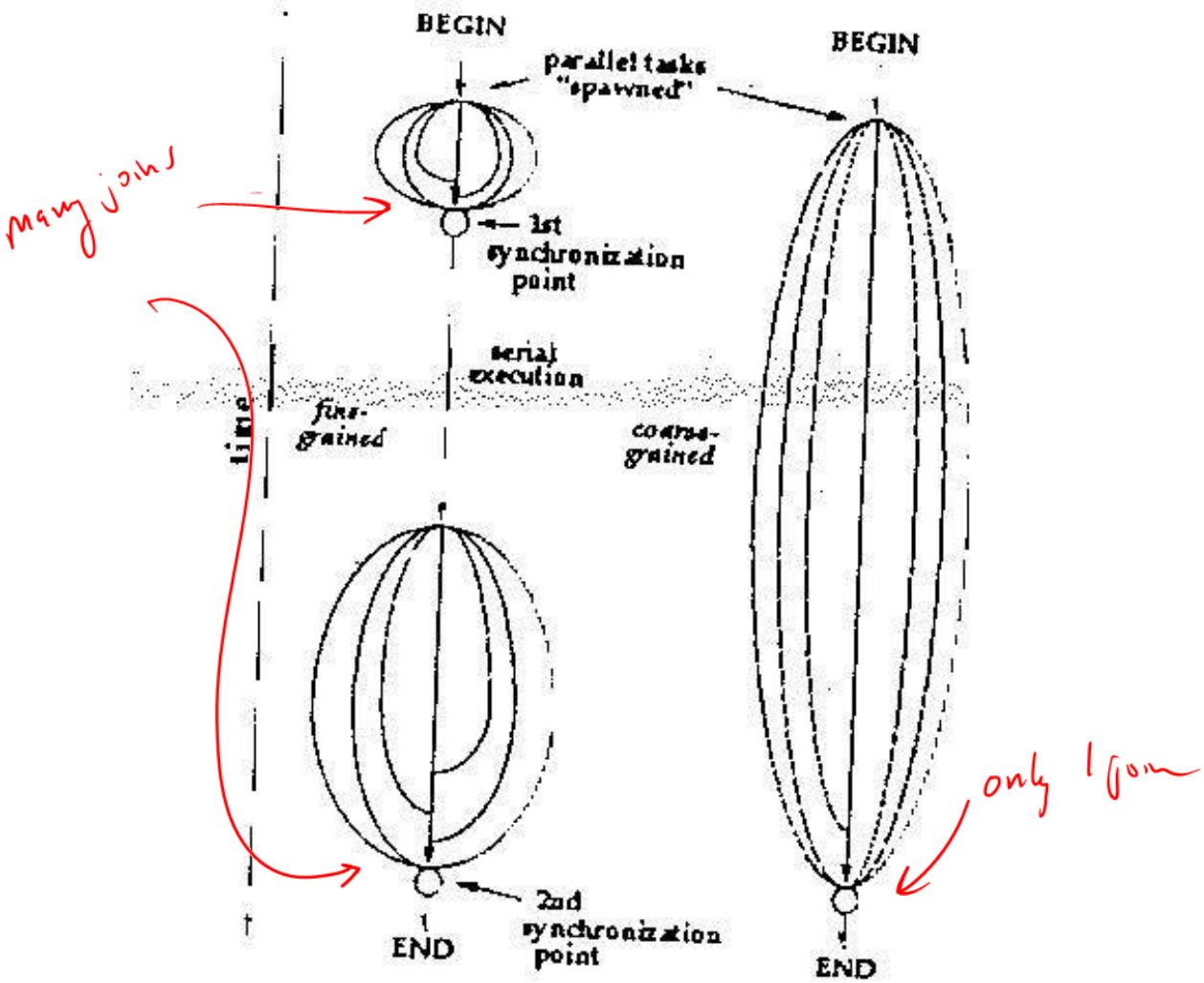
- Problem 1: Calculate the potential energy for each of several thousand independent conformations of a molecule. When you are done, find the minimum energy conformation.
- Problem 2: Calculate the first million terms of the Fibonacci series (1,1,3,3,5,8,13, 21) from the formula:
- $F(k+2) = F(k+1) + F(k)$ → difficult to parallelize due to synchronization

Why is one problem more parallelizable than another?

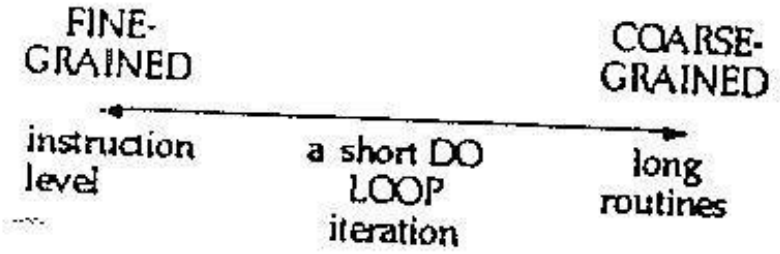
-

- ***Granularity***
- Granularity is a measure of the ratio of the amount of computation done in a parallel task to the amount of communication.
- "Fine Grained" means there is lots of communication and
- "Coarse Grained" means there is lots of parallel computation in comparison the amount of communication.

▷ We want coarse-grained to reduce communication


BEGIN
parallel tasks "spawned"
BEGIN
Many joins
1st synchronization point
serial execution
time
fine-grained
coarse-grained
only 1 join
2nd synchronization point
END
END

Continuum


FINE-GRAINED
COARSE-GRAINED
instruction level
a short DO LOOP iteration
long routines

How would you structure a solution of a heuristic in parallel?

- Assume for example we do a GA
  - that it takes F= 1 minute (60 seconds) to evaluate fitness Cost(s) once and
  - 10 seconds to do the heuristic steps for crossover and mutation and select best solution to be used in the next iteration and
  - one second to communicate between processors.
- Assume in each iteration, you want to have a population of 20