

Lesson 12: Searching and optimization

G Stefanescu — University of Bucharest

Parallel & Concurrent Programming
Fall, 2014

Combinatorial search and optimization techniques:

- find a solution for a problem with many potential solutions — often exhaustive search is infeasible
- examples: traveling salesperson problem, 0/1 knapsack problem, n -queens problem, 15-tiles puzzle, etc.
- many “serious” applications in commerce, banking, industry; e.g., financial forecasting, airline fleet/crew assignment, VLSI chip layout, etc.
- basic techniques: *branch-and-bound search, dynamic programming, hill climbing, simulated annealing, genetic algorithms*
- parallel methods may be used for all these basic techniques



Branch-and-bound search

Branch-and-bound search:

- the state space of this problem is a *(static/dynamic) tree*:
 - a divide-and-conquer approach is used
 - at each level one choice between a finite number of choices C_0, \dots, C_{n-1} is selected
 - the leaves represents possible solutions to the problem
- the space tree may be explored in various ways:
 - depth-first* - first downward, then left-to-right
 - breadth-first* - first left-to-right, then downward
 - best-first* - select the branch most likely leading to the best solution
 - selective* (*pruning* some parts of the tree using *bounding* or *cut-off* functions)



..Branch-and-bound search

A few details regarding a sequential implementation:

- a node
 - is *live* if it was already reached but not all its children have been explored
 - is *E-node* if its children are currently explored
 - is *dead* if all its children have been explored
- in branch-and-bound search all children of an E-node becomes live nodes
- the live nodes are usually stored in a queue (this is a *priority queue* for best-first search strategy)

Backtracking is usually considered a distinct approach: when a search cannot proceed downward the search continues with a next alternative on the upper level; an appropriate data structure for this method is a stack (lifo queue).



Parallel branch-and-bound search

Parallel branch-and-bound

- a partitioning strategy may be used: separate processes may independently search different parts of the space tree
- even if each process uses a “pure” (e.g., depth-first search) on its own part, the resulting *parallel search* may be a sort of mixture as the order of parallel processing is difficult to predict
- the method is by far *less efficient* as one initially may *expect*:
 - the current “best solution” may be known by all processes in order to act properly on their part, for instance to prune certain irrelevant parts; hence, a lot of communication is to be used, decreasing the efficiency of the parallel approach
 - good load balancing is difficult to achieve



..Parallel branch-and-bound

- various strategies have been developed
- in a *shared-memory* model one may use a shared queue, but the serialization of the queue access limits the speedup to

$$S(n) \leq \frac{t_{queue} + t_{comp}}{t_{queue}}$$

where: t_{queue} is the mean time to access the queue and t_{comp} the mean time of process computation (this comes from Amdahl's law)

- for best-first strategies priority queues have to be used; they may be efficiently implemented using a *heap* data structure; furthermore, concurrent versions to handle the heap have been suggested (Rao and Kumar '88)



..Parallel branch-and-bound

Anomalies: with n processes one normally expect to reach the solution n times faster; but:

- *acceleration anomaly* - parallel speedup may be greater than n or “super-linear”:
 - in the parallel version a good solution may be found very early cutting down the time more than expected (n times)
- *deceleration anomaly* - parallel speedup is less than n , but more than 1:
 - this appears when the position of a feasible solution in the tree cannot be reached n times faster than with 1 process
- *detrimental anomaly* - parallel speedup less than 1:
 - total time with n processes is worse than using a single process



Genetic algorithms: generalities

Natural evolution:

- the basic information of living beings is contained into their *chromosomes*
- natural evolution works at this chromosome level:
 - when individuals reproduce, portions of parents' genetic informations are combined to generate the offspring' chromosomes
 - the combination is based on a *crossing over* mechanism
 - in addition, sometimes (random) *mutations* may appear
 - the environment selects the “most fit” individuals
 - most mutations *degrade* the individuals, but sometimes *favorables changes* may appear



..Genetic algorithms: generalities

Genetic algorithms: a *genetic algorithm* describes a computational method to solve a problem using ideas abstracted from this biological process of evolution:

- the algorithm starts with an *initial population* of solutions (individuals); *next generations* are iteratively created
- the individuals are evaluated using a set of *fitness* criteria
- a *subset* of population is selected, tending to favor the “most fit” individuals; this subset is used to produce a new generation of *offspring* (the “crossing over” mechanism is used here)
- finally, a small number of individuals of a new generation are subject to random *mutations*



Sequential genetic algorithms

Sequential genetic algorithms (sketch):

```
generation_no = 0;
initialize Population(generation_no);
evaluate Population(generation_no);
termination_condition = False
while (!termination_condition) {
    generation_no++;
    select Parents(generation_no) from
        Population(generation_no-1);
    apply crossing_over to Parents(generation_no) to get
        Offspring(generation_no);
    apply mutation to Offspring(generation_no) to get
        Population(generation_no);
    evaluate Population(generation_no);
}
```



..Sequential genetic algorithms

An example: a numerical computation problem

find the maximum of a function

Initial population:

- suppose the goal is to find the maximum of

$$f(x, y, z) = -x^2 + 10^6x - y^2 - 4 \cdot 10^4y - z^2$$

for integer x, y, z between -10^6 and 10^6 (actually, maximum value is for $x = 5 \cdot 10^5, y = -2 \cdot 10^4, z = 0$)

- initial population: $(2 \cdot 10^6 + 1)^3$ individuals (all combinations of values for x, y, z in the given range)



..Sequential genetic algorithms

Data representation:

- we need “strings” (as in the case of chromosomes)
- there are $2 \cdot 10^6 + 1$ values in the given range, hence we may use 21 bits to represent them (sign-plus-magnitude representation:
 $\overline{36}_{10} \rightarrow 0000000000000000100100;$
 $-\overline{7}_{10} \rightarrow 10000000000000000000111;$
 $-\overline{1024}_{10} \rightarrow 1000000000100000000000;$)
- concatenating all these bits, we get the representation of a potential solution by a string with 63 bits
 $(36, -7, -1024) \rightarrow$
00000000000000000100100100000000000000000011110000000000100000000000



..Sequential genetic algorithms

Fitness function:

- for each (x, y, z) computes $f(x, y, z)$ (larger value, better fit)

Constraints:

- not all 63 bit strings represent valid combinations: the values outside $[-10^6, 10^6]$ should be deleted (or “surgically repaired”)

Number of individuals:

- small number \Rightarrow slow convergence;
large number \Rightarrow heavy computation
- usually 20 to 1000 pseudo-random generated strings



..Sequential genetic algorithms

Parents' election:

- more fit individuals have greater chance to be selected:
 - best fit individuals may be selected to produce offspring,
 - but choosing only the “best fit” individuals may lead to a fast convergence to a local optimum, totally missing the global one
- *tournament selection* - is a solution to avoid the stick into a local optimum:
 - a set of k individuals are randomly selected to enter into a tournament; the most fit individual is selected as a parent for the next generation;
 - the tournament is repeated n times (if n parents have to be selected)



..Sequential genetic algorithms

Offspring production:

- *single-point crossing over:*
 - given two parents $A = a_1 \dots a_n$ and $B = b_1 \dots b_n$, randomly choose a cut point - say, this is between p -th and $p + 1$ -th bits
 - create 2 children:
child 1 = $a_1 \dots a_p b_{p+1} \dots b_n$ and child 2 = $b_1 \dots b_p a_{p+1} \dots a_n$
- other alternatives may be used (multi-point crossing over, uniform crossing over, gene sharing, etc.)

Mutation:

- randomly select and modify one or more bits of an individual (sometimes this may have huge effect - e.g., in our example, the 1st bit represents the sign; the last bit is less important)
- mutation should be kept at a small rate (for convergence)



..Sequential genetic algorithms

Variations:

- carry over a few well fit individuals to the next generation
- randomly create a few new individuals
- let the population size to vary from a generation to the next

Termination:

- either apply the basic step to generate an a-priori given number of generations
- or use other criteria (like the difference between the value of the optimum at a generation and at the next one; or the degree of similarity of the individuals within a generation, etc.)



Parallel genetic algorithms (PGA)

Parallel genetic algorithms: two quite different possible use of parallel processing are

1. *isolated subpopulations*: each processor operates independently on an isolated subpopulation; their communication is through *migration* of individuals
2. *common population*: in this approach there is a common population; each processor does a portion of the selection-crossover-mutation work



..(PGA) Isolated subpopulations

Isolated subpopulations:

- each processor handles an isolated subpopulation of individuals (it does the evaluation of fitness, selection, crossing over, mutation)
- periodically, say after k generations, the processors exchange certain individuals - a so called “migration” process takes place



..(PGA) Isolated subpopulations

Migration operator:

- migration requires a few activities: *selecting the emigrants; sending the emigrants; receiving the immigrants; integrating the immigrants*
- a selection criterion may be to send the best individuals to a particular processor; the convergence is then faster, but there is a risk of sticking into a local optimum
- sending/receiving the emigrants/immigrants may be easily done in a message-passing model
- migration introduces a communication overhead, hence both the frequency and the volume of communication should be carefully considered



..(PGA) Isolated subpopulations

Migration models:

- *the island model:* individuals are allowed to be sent to any other subpopulation
- *the stepping stone model:* individuals are allowed to be sent only to the neighboring subpopulations
- the island model allows more freedom, but also more communication overhead is introduced



..(PGA) Isolated subpopulations

Parallel genetic algorithms (sketch of slave code):

```
generation_no = 0;
initialize Population(generation_no);
evaluate Population(generation_no);
termination_condition = False
while (!termination_condition) {
    generation_no++;
    select Parents(generation_no) from
        Population(generation_no-1);
    apply crossing_over to Parents(generation_no) to get
        Offspring(generation_no);
    apply mutation to Offspring(generation_no) to get
        Population(generation_no);
    apply migration to Population(generation_no);
    evaluate Population(generation_no);
}
```



..(PGA) Parallelizing a common population

Parallelizing a common population:

- in this approach general techniques to parallelize the sequential algorithm are used
- for instance, if processors have access to the required information, then parallel selection may be used (e.g., each processor does separate tournaments),
- also crossover and mutation for different individuals may be done by different processors
- as with many other parallel programs, the approach is useful when the computational effort is large enough to hide the time spent on communication



Successive refinement

Successive refinement:

- this is another iterative approach on searching algorithms based on a series of successive refinements of the solution space
- start with a coarse “cube” of solution space; e.g., for the given example (slide 12.11) test every ten-thousandth point
- the best k points are retained and new finer, but smaller, cubes centered in these points are evaluated
- finer and finer grids are introduced and evaluated till a good solution is obtained
- this method may be easily parallelized



Hill climbing

Hill climbing:

- hill climbing is another common approach to searching and optimization
- based on the strategy of improving the result at any step (keep moving toward the pick - in the “hill climbing” metaphor)
- the problem with this approach is that one may reach a local pick and never go down to a little valley to climb up a higher pick on the other side
- a solution is to use more starting points - then the chance of reaching a global optimum is increasing



..Hill climbing

Parallel versions may be easy to implement:

- the starting points may be chosen in a random way (Monte-Carlo method)
- each processor may independently handle different starting points
- depending on the problem, either a static or a dynamic work pool may be used to allocate points to processes



..Hill climbing

A banking application:

- Problem:
 - how to distribute “lock-boxes” across a country to optimize the receiving of payments for a given company?
- the optimization function is in terms of *float_days* (a “*float*” is the delay between customer’s payment mailing date and the firm’s collection date, multiplied by the amount of payment)
- —the firm has to pay for the lock-boxes, hence not too many are to be used;
 - their optimal distribution depends on the mailing time, but also on the distribution of the money the firm is expecting to collect from various places



..Hill climbing

Sequential code (sketch):

```
set Float_days to 1000;
for (i=1; i < MaxLockBoxes; i++) {
    for (all possible placements of i lockboxes) {
        compute CurrentFloat_days;
        if (Float_days > CurrentFloat_days) {
            Float_days = CurrentFloat_days;
            save i and lockbox placement pattern;
        }
    }
}
```

Computational effort: to select k lock-box places among n places

requires to check $\frac{n!}{k!(n-k)!}$ combinations;

[for $(n, k) = (200, 6)$, approx. $8 \cdot 10^{10}$ combinations]



..Hill climbing

Parallel version:

- use hill climbing to avoid searching into such a huge space of possible placements
- incrementally change a lock-box location to another one that produces the greatest reduction of float_days
- eventually a location realizing a local minimum of float_days is reached
- one may randomly generate a number of initial starting locations to increase the chance of reaching a global minimum
- parallelization techniques (described when hill climbing method was introduced) may be used