Vehicle Routing Using Artificial Bee Colony Algorithm

Presentation 3

EMBARRASSINGLY PARALLEL
Ajinkya Dhaigude
Sameer Raghuram
Contents

- Review of the Vehicle Routing Problem
- Breakdown of the ABC algorithm implementation
  - Solution generation phase
  - Employed Phase
  - Onlooker phase
- Sequential Approach
- Parallel Approach
- Demo
- Observations
- References
Vehicle Routing Problem: A Review

- Falls in category of NP-hard
- Can be thought of as an m-TSP problem.
  - So, we would have m number of vehicles based out of a depot.
- Can have several constraints applied to it
  - Capacity
  - Time Windows
  - Customer Demand
Figure 1 - Solution for VRP with 4 vehicles [2]
Artificial Bee Colony Algorithm explored in 3 steps
Step 1: Generate Candidate Solutions (Scout Phase)

- Generate candidate solutions by randomly permuting the order of nodes in a route.
- We see the depot node appear multiple times - this signifies the start (and end) of the route of one vehicle.
- Number of solutions is equal to the number of employed bees.

<table>
<thead>
<tr>
<th>Customers</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution</td>
<td>( X = {0,1,2,0,4,5,3,0,6,7,0,0} )</td>
<td>Route 1: {0,1,2,0}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Route 2: {0,4,5,3,0}</td>
<td>Route 3: {0,6,7,0}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Route 4: {0,0}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 1 - Pseudocode (Generating a random solution)

- For each Solution:
  - For each node in solution route:
    - Compute random index ‘x’ (where $x < \text{len(route)}$)
    - Swap current node and node at position $x$. 
Step 2 - Explore Solutions (Employed Phase)

- All the SN solutions are explored in this phase.
- The exploration involves randomly swapping two nodes in a route to see if the fitness of a solution has improved.
- If a solution is improved in an exploration, it’s trial number is reset, if it doesn’t, the trial count is incremented. (Each solution has a maximum number of trials before it is discarded)

<table>
<thead>
<tr>
<th>Before swap</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th>7</th>
<th>0</th>
<th>6</th>
<th>1</th>
<th>0</th>
<th>4</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>After swap</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Pseudocode - Step 2

- For all Solutions:
- Keep track of total_fitness value
- For each Node in Solution:
  - Randomly choose indices x and y (where x\&y < length(route array))
  - Swap nodes at position x and y
  - Compute new fitness
  - Compare old and new fitness
  - Keep new solution if new fitness is greater. Reset Trial number.
  - Revert swap at x \& y of no improvement in fitness. Increment trial number.
  - Increment value of total_fitness with the fitness of the current solution.
Step 3 - Choose solutions based on fitness (Onlooker Phase)

- Once the solutions have been explored, and have been assigned a fitness value, they are ready to be explored again.
- These explorations are done for solutions chosen using a ‘roulette-wheel’ selection method, where solutions with a better fitness value are more likely to get explored further.
- After this step, solutions that have been depleted get discarded and new ones are randomly generated to replace them.
- The process repeats again from step 1.
Pseudocode - Step 3

- For each onlooker solution, do the following -
- Calculate probability as -
  - Select a random double ‘probab’ between 0 and 1
  - Multiply ‘probab’ with ‘total_fitness’ calculated from previous step’ to obtain ‘selection_probability’
- Select a solution from previous step
  - For each solution, subtract it’s fitness from ‘selection_probability’
  - Continue until ‘selection_probability’ is negative or zero.
  - Choose the corresponding solution
- Try and optimize chosen solution.
Sequential Approach

- ABC Algorithm is suitable for sequential programs, and is pretty straightforward.
- Each round of the ABC algorithm depends on the solutions generated by the previous round.
- Within each round, we have sequential dependencies since the onlooker phase depends upon all the solution generated by the employed phase.
Sequential Approach

- FOR N ROUNDS, DO
- Generate $SN$ number of initial solutions
- Employed Phase - Explore each solution
  - Try improving it by swapping two nodes chosen at random
  - Replace solution with new one of fitness improves during exploration
- Onlooker Phase - Choose solution based on fitness
  - Use roulette wheel selection method.
  - Solutions with better fitness have a higher probability of being selected.
  - Try and improve chosen solutions
- Discard stale solutions, replace with newly generated solutions
- Output best solution found after N rounds
Start

1. Generate the initial solutions
2. Evaluate the solutions and memorize the best solution
3. Generate new neighborhood solutions by employed bees
   - Evaluate the solutions and apply greedy selection
   - Calculate selective probabilities of food sources for onlooker bees
   - Generate new neighborhood solutions for onlooker bees
   - Evaluate the solutions and apply greedy selection
   - Memorize the best solution
4. Abandon any solutions?
   - Yes
   - Replace abandoned solutions with new solutions by scout bees
   - Meet stopping criterion?

End
Parallel Approach

- The ABC algorithm is tricky to parallelize.
- There are far too many sequential dependencies between rounds, and within each round as well.
- Tried several approaches (none of them really worked out)
  - Tried parallelizing each phase within a round
  - Tried to parallelize only parts of the algorithm that were embarrassingly parallel.
- Ultimately went with the approach suggested by one of the research papers covered in our literature survey - H Narasimhan, “Parallel artificial bee colony (PABC) algorithm”, World Congress on Nature & Biologically Inspired Computing, 2009, NaBIC 2009, pp 306 - 311.
Parallel Approach

- Divide $N$ solutions into equal sections of size $N_p$
- Divide $N_p$ solutions among processors
- Perform ABC algorithm as described earlier on each processor
- Generate new solution in each cycle at each processor
- Keep track of local best solution for each processor
- Perform parallel reduction on processor local best solutions to obtain global best solution.
DEMO
Observations

- Parallel implementation scaling properly.
- Certain constraints put in place to improve solution quality
  - Example - We try and reject solutions where not all vehicles are given valid routes.
  - Constraints reduce the number of valid solutions however.
  - Sometimes the algorithm fails to obtain valid solutions
- Since, the ABC algorithm is a heuristics based algorithm, we to perform the algorithm multiple times and take the best answer among these trials to get a valid answer.
- Hence, it is key that we make sure that our solutions are of a good quality, regardless of whether they are completely optimized or not. (It is highly unpractical to aim for 100% optimization for such problems)
References


Questions?