NatAlgReport

Name: Kush Kapur User-ID: ccbd24

AB -> Bee colony algorithm

Algorithm Implemented: *Bee colony algorithm*
Best Result: Minimum Value = *0.0****, Minimum Point*** *= [-3.281580677104401e-09, 8.08915138459822e-09, -1.4676840570899771e-08],* ***time elapsed =*** *15.4 seconds*
Best Parameters: Number of employee bees: *40,* ***Number of onlooker bees:*** *40,* ***Lambda:*** *80,*
Fitness: *1/computing function as the aim was to minimise the function and maximise fitness.*
Hyperparameter Tuning: Number of employee bees and onlooker bees (lambda: 10, Running time: 50s): *Initial started with 20 employee, and 20 onlooker bees, achieving minimum of 0.005 +- 0.0005 for 10 runs, therefore I thought that it was converging at a local minimum, so I increased the number of onlooker bees from 20 to 40, as they perform near neighbour and search around the best solutions. Increasing onlooker to 40 with 10 runs made the result worse to 0.01 +- 0.005, so it's as necessary to have the same number of food sources as onlookers, to have variety of solutions as greater number of food sources allows greater coverage of solution space. Increased onlooker and onlookers did not show further signs of improvements, so I focus on lambda next.*
Lambda (Employee bees:40, Onlooker bees:40, Running time: 50s): *Lambda holds the key to abandoning solutions in bee colony, so I increased lambda from 10 to 40, showing significant improvement with a solution of 6.23e-07, suggesting that a lambda too low means that good solutions are abandoned too quickly and can prevent an onlooker from exploring a point close to a global minimum. Increasing lambda to a 100 guaranteed a 0.0 minimum value. Although setting lambda to 500, I found that it took a 120s for convergence to 0.0 because the bad solutions were not being abandoned frequently enough. Found 80 to converge fastest (11s for 0).*
Experimentation: Near Neighbour *exploring sphere is controlled by a float* ***change*** *lying between -1 and 1, one shortcoming could be because of this constant range it could take longer to converge to a minimum. So, I replaced it with a epoch-based learning schedule similar to one used in SVD. Through this at higher epochs a lower change is used, under the assumption that we are closer to a global minimum. Although when utilising this I found it to converge to 0 at 20s in comparison to 11s before. This could be potentially because instead of learning a representation of a function, here we are randomly exploring the solution space. Hence, decreasing change would prevent proper exploration hence longer convergence time, and not getting to the global.*
Fitness Function: *In lecture it was recommended to select food source to explore through a fitness function which is compute f(i)/total fitness, although I realise this has a shortcoming in minimisation, because even though at a better solution compute f(i) decreases but so does total fitness, therefore the change isn't as significant, whereas I chose the fitness to be 1/compute f(i) as that presents a greater difference between different food sources. This is evident because my new fitness function converges to 0.0 in 11s whereas using the previous fitness function takes 21s, in my code it is possible to use the first fitness function by replacing function* ***calc_fitness_population*** *with* ***calc_fitness_population_lectures.*** *This is also because onlookers select the solution based on probability, and by differentiating the probabilities more it makes it more likely for them to explore better solutions, hence producing better near neighbours.*

(continued over)

Name: Kush Kapur User-ID: ccbd24

AB -> Bee colony algorithm and **GC** -> Graph coloring **KEY -> Color for same name used in code**

Graph A: number of conflicts** = 4**, time taken** = 55.4 seconds **(**5 **by hybrid greedy)
Graph B: number of conflicts** = 11**, time taken** = 60.6 seconds **(**14 **by hybrid greedy)
Graph C: number of conflicts** = 55**, time taken** = 102.4 seconds **(**57 **by hybrid greedy)
***Optimal Parameters: Employee bees:**200**, Onlooker bees:**500**, lambda:**500**, fitness:**1/conflicts*
***Discretization methodology:** I chose the graph colouring problem to discretise, with the use of a hybrid greedy colouring algorithm and bee colony to optimise the results from the hybrid greedy approach. My hybrid greedy algorithm takes an input of a sequence of vertices, and creates a dictionary with vertex, colour as key, values. Another dictionary is used with vertex number, values as list of vertices with shared edges. Hybrid greedy assigns lowest colour to first vertex in sequence, and then refer to the vertex in adjacency dictionary through which any vertex sharing an edge with the initial vertex would have colour 1, removed from their set of potential colours. Then population of sequences, and colourings are generated. This population is a set of food sources used by the bee colony, where employee bees and onlooker bees use explore these food sources, using nearest neighbour. Fitness defined as inverse of number of conflicts in a colouring.*
***Experimentation:** **near_neighbourv1:** Choose 2 random food sources, and a random vertex number. Change colour for vertex in food source 1 to colour of same vertex in food source 2. **near_neighbourv2:** Rearrange sequence in canonical form (group vertices by colours), then choose a random vertex for each colour and switch its colour. Results in change of colours for max number of allowed colours vertices (resulting in very different solution).*
***Fitness:** Lecture suggested fitness 1 / **|V|^3 + (bad_edges ×|V|) + unique_colors.** When using this there are no improvements because the |V|^3 overshadows conflicts, hence onlookers don't choose the best solutions to explore resulting in this fitness never improving greedy solution. This can be tested, by comments in my code to comment existing fitness calculation with **calc_fitness_pop_lecture**. There is commented code with instruction regarding changes to test.*
***Overall Hybrid Nearest Neighbour:** Initially the hybrid greedy generated best result, not being improved further by the bee colony, so introduced another variable **Threshold_for_convergence**, incremented each time a better solution not found by nearest neighbour. When threshold is greater than 400, **near_neighbourv1** is applied by choosing the current best global solution and a random solution, to generate a near neighbour. Also, when threshold > 600, **near_neighbourv2** is used as it changes colours of a vertex belonging to each colour class, so generates a very far away solution. Hence used when threshold > 600, as that may represent a convergence to a local minimum so near_neighbourv3 generates a very different food source to escape that local minima and explore a different region of the solution space.*
***Performance:** Before the hybrid near neighbour implementation the best solution was generated by the initial hybrid greedy approach, now the solution from the greedy is improved upon using for example for Graph B greedy provides 15 conflicts which is then improved to 11 (60.6s) by my hybrid nearest neighbour approach as it takes the best possible solution at convergence and generates a better local minimum, also **near_neighbourv2** is the prevents convergence at a local minima. Although for Graph C, the algorithm needs to be ran for more than 100s in order to get some results, within 60s no improvements are shown using the bee colony.*
***Hyperparameter Tuning: Employee and Onlooker bees:** large number of onlooker bees used as near neighbour algorithm essential to generate a better solution than greedy, therefore by having more food sources it makes the algorithm more inefficient, as most results generated by the hybrid greedy approach are quite consistent. Increasing onlookers' results in greater exploration of the solution space. By increasing onlooker bee from 200 to 500, Graph A and B improved from their results from 5, 14 to 4, 11.*