Virtual machine placement using Gray wolf optimization, Genetic algorithm, Memetic and Simulated annealing

Kasra Darvishi

The Institute for Research in Fundamental Sciences (IPM)

Abstract. This report reviews the result of research and works that have been done during the internship program of IPM. The purpose of this program was to study different optimization algorithms and apply them to the VM placement problem.

Due to recent increase of energy consumption in data centers, improving their energy efficiency is now necessary. This goal can be achieved by minimizing the number of active physical machines in data centers. Various algorithms performing VM placement and their effectiveness is discussed in this report.

1 Gray wolf optimization

GWO is inspired by gray wolf groups where all the wolves in a group are guided by the dominant wolves. In this algorithm each possible solution is assumed as a wolf and solutions with highest score are the dominant wolves.
In an abstract search space we have no idea about the location of the optimum solution(prey). In order to mathematically simulate the hunting behavior of grey wolves, we suppose that the dominant wolves have better knowledge about the potential location of prey.
Therefore, we save the first three best solutions obtained so far and oblige the other search agents to update their positions according to the position of the best search agents:

$$D = |C.X_p(t) - X(t)| \tag{1}$$

$$X(t+1) = X_p(t) - A.D \tag{2}$$

The vectors 'A' and 'C' are calculated as follows ('r' is random number):

$$A = 2A.r_1 - a \tag{3}$$

$$C = 2.r_1 \tag{4}$$

'A' is a random value in the interval [2a, 2a] where 'a' is decreased from 2 to 0 over the course of iterations, in order to mathematically model approaching the prey.
When the value of 'A' is more than 1 or less then -1, the wolves diverge from pray to search and avoid getting stuck in local optimum (exploration). Another component of GWO that favors exploration is 'C' which causes exploration not only during initial iterations but also final iterations. This component is very helpful in case of local optima stagnation, especially in the final iterations.
When random values of 'A' are in [-1,1], the wolves attack towards the prey. In other words they will converge to the best optimum found so far (exploitation).

GWO for VM palcement

Each allocation is a wolf and we need to define what is the distance between the wolves in order to use GWO for finding a good allocation. Allocations are sequences of numbers like "223" which show number of the server to which each VM is allocated.
Now if we assume that each allocation is a point in space, the distance between the allocations "223" and "224" would be 1. But if we measure the distance of the allocations "223" and "226" we see that the distance is 2 units more than the two previous allocations. This increase in distance is clearly meaningless so we need some thing else in order to set distance between allocations.
We can assume that the distance between allocations is the percentage of the difference between them. In this way the distance of the allocations "223" and "224" is the same as the distance between "223" and "226" and both are 33 percent.
Using the definition of the distance between wolves, the position updating procedure for each non-dominant wolf would be as follows:

- Choose one dominant wolf randomly from the three wolves that have highest scores among the others and find the percentage of difference

- Calculate the final percentage of different that the wolf should have with the dominant wolf by multiplying A with D

- Get a random number for each of the VMs

- If random number is more than final difference, make the allocation of that VM same as the dominant wolf

- If random number is less than final difference, allocate that VM to a random server

'A' is a random value in the interval [0, a] where 'a' is decreased from 2 to 0 over the course of iterations, in order to mathematically model approaching the prey. When the value of 'A' is more than 1, the wolves diverge from pray, by increasing the percentage of difference, inorder to search and when 'A' is less than 1, they will converge to the best optimum found so far.

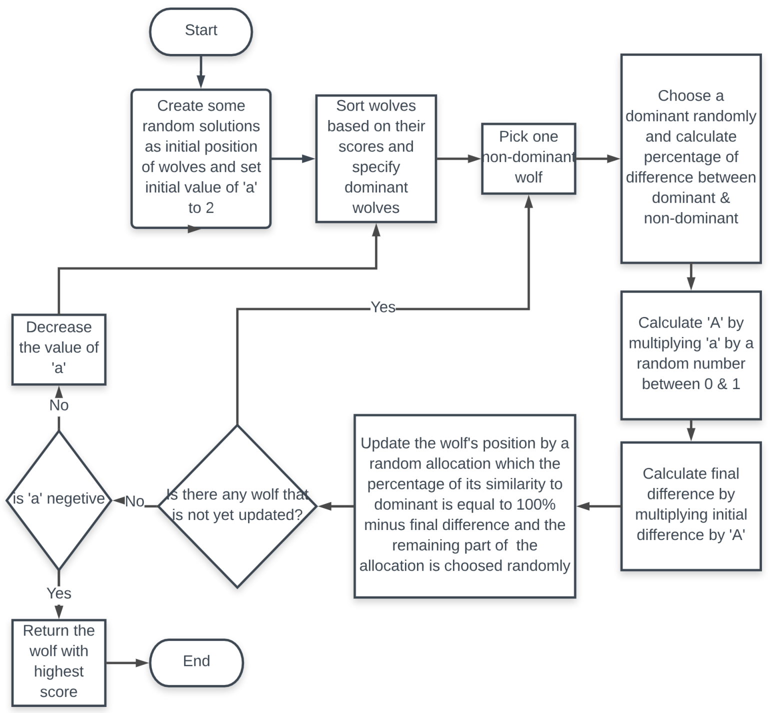

Start
Create some random solutions as initial position of wolves and set initial value of 'a' to 2
Sort wolves based on their scores and specify dominant wolves
Pick one non-dominant wolf
Choose a dominant randomly and calculate percentage of difference between dominant & non-dominant
Calculate 'A' by multiplying 'a' by a random number between 0 & 1
Calculate final difference by multiplying initial difference by 'A'
Update the wolf's position by a random allocation which the percentage of its similarity to dominant is equal to 100% minus final difference and the remaining part of the allocation is choosed randomly
Is there any wolf that is not yet updated?
Yes
No
is 'a' negetive
No
Decrease the value of 'a'
Yes
Return the wolf with highest score
End

Fig. 1: Flowchart of GWO

2 Genetic Algorithm

Genetic algorithm is inspired by the process of natural selection. Individuals that have the highest fitness among the population have higher chance to breed and transfer their characteristics to next generations.
First population is a set of random solutions to our allocation problem. The GA then improves the structures in this population by performing selection, followed by crossover and mutation. After several generations, sufficiently good solutions will be formed in the population.

Evaluation function

The goal is to maximize utilization and having the most possible number of empty servers while all the tasks are allocated to some server. Evaluation function which is currently used is as follows, but it should be modified in order to consider the cost of moving a task from one server to another.

```
for server in servers:
    if server.isEmpty():
        value += 1
    elif server.hasExceeded:
        value -= server.get_utilization()
    else:
        value += server.get_utilization() ** 2
```

Fig. 2: Evaluation function

Selection and Crossover

Selection is performed based on the fitness function. Solutions with highest fitness are chosen to perform crossover operation. A few number of the randomly chosen solutions contribute in this operation too. This way the population avoids getting stuck in local optimum. Crossover operation is described in fig 2

```
   Input  : two parent chromosomes, $C^i = x_1^i x_2^i \cdots x_n^i$ and $C^j = x_1^j x_2^j \cdots x_n^j$
   Output: one child chromosome, $C^k = x_1^k x_2^k \cdots x_n^k$
1  $f^i \leftarrow fitness(C^i)$;
2  $f^j \leftarrow fitness(C^j)$;
3  for $q$ = 1 to $n$ do
4      randomly generate a real value between 0 and 1, $r$;
5      if $r < f^i/(f^i + f^j)$ then
6          $x_q^k \leftarrow x_q^i$;
7      end
8      else
9          $x_q^k \leftarrow x_q^j$;
10     end
11 end
12 output $C^k$.
```

Fig. 3: Crossover function

Mutation

The goal of this operation is to prevent the algorithm to be blocked in a local minimum. By mutation we decrease the risk of losing a good solution which does not exist in the first population. Another way to meet this goal is to increase the population size which is not computationally effective.

The mutation operator simply randomly picks up a gene in the chromosome and inverts the value of the chosen gene (each solution is called a chromosome and server numbers in the solution are its genes).

Hybrid GA

Hybrid GA (also called Memetic) has a simple difference from the algorithm that was just described. It performs some operations on the solutions in order to improve them locally. In particular, our task allocation algorithm tries to repair the solutions which are not valid (for example because of allocating VMs to a server more than its capacity) using a method which accelerates getting to the optimum solution.
The basic idea behind the infeasible solution repairing procedure is to re-allocate the tasks currently allocated to the server, one by one to other servers until the violation in the resource constraints on the server are resolved.
The procedure first attempts to re-allocate the tasks to the servers that has already been used in order to form solutions with high utilization.
Hybrid GA finds better solutions than GA in a less time consuming manner. The main procedure of GA and Hybrid GA is shown in fig3 and fig4:

```
generate a population of $PopSize$ individuals, $Pop$;
find the best individual in $Pop$;
while the termination condition is not true do
    for each individual $x$ in $Pop$ do
        calculate its fitness value $f(x)$;
    end
    for each individual in $Pop$ do
        use the roulette selection to select another individual to pair up;
    end
    for each pair of parents do
        probabilistically use the biased uniform crossover operator to produce
        an offspring;
    end
    for each individual in $P$ do
        probabilistically apply the mutation operator the individual;
    end
    find the best individual in $Pop$;
    if the best individual in Pop is better than the current best individual then
        replace the current best individual with the new best individual;
    end
end
```

Fig. 4: main procedure fo GA

```
generate a population of $PopSize$ individuals, $Pop$;
find the best individual in $Pop$;
while the termination condition is not true do
    for each individual $x$ in $Pop$ do
        if $x$ is not feasible then
            use the repairing procedure to convert $x$ into a feasible one;
        end
        use the local optimization procedure to optimize $x$;
        calculate its fitness value $f(x)$;
    end
    for each individual in $Pop$ do
        use the roulette selection strategy to choose another individual in the population and pair them
        up;
    end
    for each pair of selected individuals do
        probabilistically use the biased uniform crossover operator to produce one offspring;
    end
    for each individual in $P$ do
        probabilistically apply the mutation operator to it;
    end
    find the best individual in $Pop$;
    if the best individual in $Pop$ is better than the current best individual then
        replace the current best individual with the new best individual;
    end
end
decode the best individual and output it.
```

Fig. 5: main procedure fo Hybrid GA

3 Simulated annealing

The simulated annealing was inspired from the process of annealing in metal work. Annealing involves heating and cooling a material to alter its physical properties.
Algorithm starts with one random solution and then it tries to find better solutions by exploring the neighborhood states. The Temperature has an fundamental role in the function that decides whether to accept the newly found solution or not.
We initially set the temperature high then allow it slowly to cool as the algorithm runs. While this temperature variable is high the algorithm will be allowed, with more frequency, to accept solutions that are worse than our current solution. This gives the algorithm the ability to jump out of any local optimums it finds itself in early on in execution. As the temperature is reduced so is the chance of accepting worse solutions, therefore allowing the algorithm to gradually focus in on a area of the search space in which hopefully, a close to optimum solution can be found.

Neighbor solutions

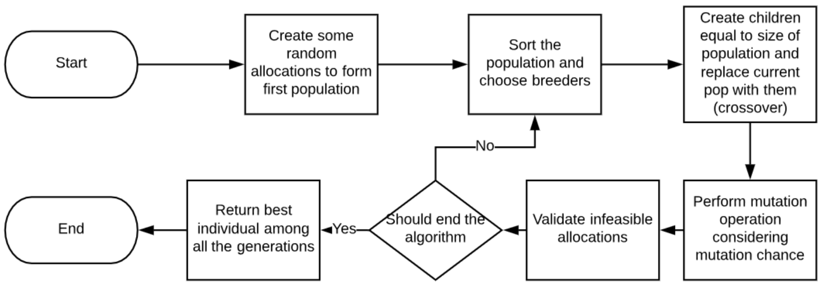

Start
Create some random allocations to form first population
Sort the population and choose breeders
Create children equal to size of population and replace current pop with them (crossover)
Perform mutation operation considering mutation chance
Validate infeasible allocations
Should end the algorithm
No
Yes
Return best individual among all the generations
End

Fig. 6: Flowchart of Hybrid GA

At each iteration one of three operations named Inversion, Translation and Switching are randomly used to get a neighboring solution. These operations are shown in figures 6 to 8.

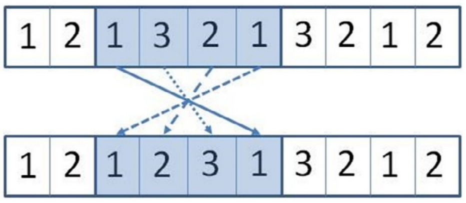


Fig. 7: Inversion operation

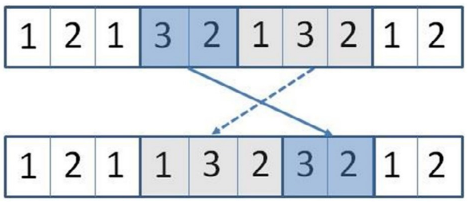


Fig. 8: Translation operation

Acceptance criteria of new solutions

Acceptance criteria decides which neighboring solution becomes the next new state. If a new solution is a valid allocation and it has higher score based on the evaluation function, it will be accepted as the new state. But when the new valid solution has lower score than the current state, it is accepted only if the difference between their scores is lower than a certain limit. That limit is calculated as follows:

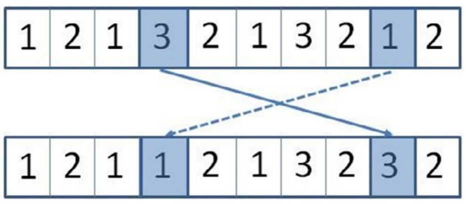


Fig. 9: Switching operation

$$ScoreDecreaseLimit = 1/2 * maxScore * t_c/t_0 \tag{5}$$

Where tc is the current temperature and t0 is the initial temperature and the "maxScore" is the maximum score that one sever can possibly have based on Evaluation function . This way in the initial iterations large decrease in the score is allowed for the next state in order to avoid local optimums (exploring) and in the end of the cooling process only neighbors with higher scores are accepted (exploiting).

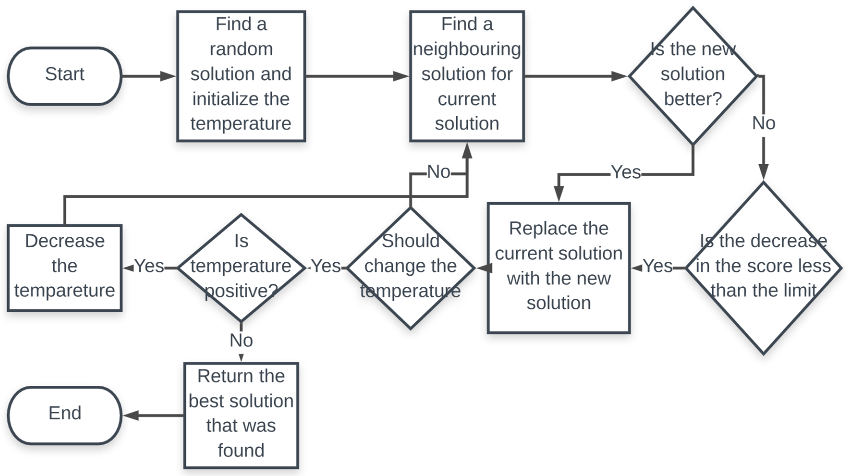
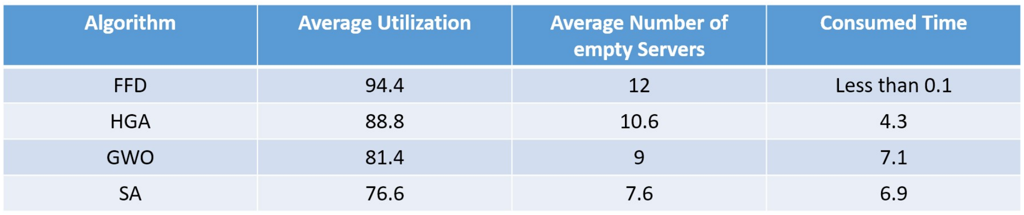

Start
Find a random solution and initialize the temperature
Find a neighbouring solution for current solution
Is the new solution better?
No
Yes
Is the decrease in the score less than the limit
Yes
Replace the current solution with the new solution
Should change the temperature
No
Yes
Is temperature positive?
Yes
Decrease the tempareture
No
Return the best solution that was found
End

Fig. 10: Flowchart of SA

4 Results

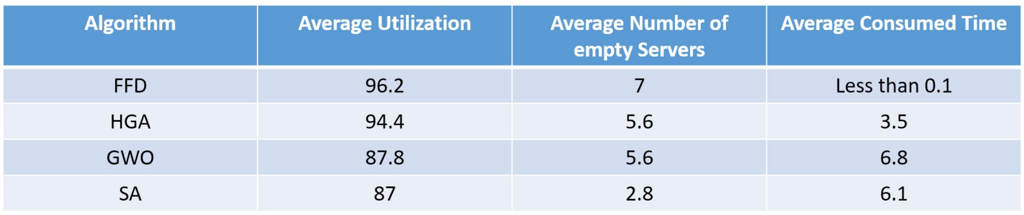
In order to test the implementations of the mentioned algorithms, it was needed to generate some VM requests and Server specifications. For the first test specifications are similar to published data set of Microsoft Azure. In the second test

all the VMs and Servers are generated randomly.
When the Goal is maximizing the utilization, First fit decreasing algorithm(FFD) performs very well. But if the Evaluation function is some thing more complicated, FFD can not be used in order to decrease the power consumption and the described algorithms will be more useful.
Each of the tests are repeated five times.



Algorithm	Average Utilization	Average Number of empty Servers	Consumed Time
FFD	94.4	12	Less than 0.1
HGA	88.8	10.6	4.3
GWO	81.4	9	7.1
SA	76.6	7.6	6.9

Fig. 11: Test using Microsoft Azure data set



Algorithm	Average Utilization	Average Number of empty Servers	Average Consumed Time
FFD	96.2	7	Less than 0.1
HGA	94.4	5.6	3.5
GWO	87.8	5.6	6.8
SA	87	2.8	6.1

Fig. 12: Test using random specifications