Optimization of File Compilation by Meta-heuristic and Mathematical Techniques

Suvayan Nath
B.Tech Chemical Engineering
Indian Institute of Technology Guwahati
n.suvayan@iitg.ac.in

Eshu Patel
B.Tech Chemical Engineering
Indian Institute of Technology Guwahati
p.eshu@iitg.ac.in

***Abstract*—The optimization problem for file compilation on servers is framed, and the constraints and objective function are discussed. A meta-heuristic approach using genetic algorithm (GA)is proposed to obtain a feasible solution.Both single-objective GA and multi-objective GA (MOGA) are explored. A mathematical formulation (MINLP) is further shown to obtain the optimal solution to the problem.**

I. Introduction

The problem of file compilation is similar to the classical job shop scheduling problem **(JSSP)** in optimization that arises in manufacturing settings. In this problem, a set of jobs needs to be processed on a set of machines. Each job is composed of a sequence of operations that must be processed in a specific order, and each operation has a specific processing time and requires a specific machine. The objective is to find a schedule that minimizes the total time required to complete all the jobs. This problem is known to be NP-hard, and finding an optimal solution is generally infeasible for large instances. Therefore, efficient heuristics and metaheuristics have been developed to approximate the optimal solution. The job shop scheduling problem has important practical applications in industries such as automotive, aerospace, and electronics manufacturing.

The problem of file compilation is similar, where multiple files need to compiled on multiple servers. However, a constraint of file dependencies is added. Certain files need their dependencies i.e a set of files they are dependant on, to be compiled and available on the same server before compilation can begin. A dependency file compiled on a different server has a replication period, after which it becomes **global** and every server has access to it. Thus, a file with dependency needs to wait until all its dependencies are satisfied, before it can begin compilation. This gives rise to a problem known in computer science as **deadlock**. Deadlock is a situation that can occur in computer systems where two or more files are blocked, unable to proceed because they are waiting for resources that are held by the other files in a circular dependency. In other words, each file is waiting for a file (dependency) that is waiting for another one, causing a cycle of waiting that cannot be resolved. This can result in a system that appears to be hung or frozen, as no progress can be made. Deadlocks can be particularly problematic in multi-process systems, such as operating systems or distributed systems, where resources such as memory, CPU time, or network connections are shared among multiple processes. Deadlocks can lead to system crashes, reduced performance, and can be difficult to detect and resolve.

Thus the problem of file compilation is to avoid deadlocks, and find a feasible sequence. Since we have the constraint of replication time, this means feasible sequences can often have servers undergoing large idle periods, waiting for access to a dependency. This leads to deadline violations. Thus, to obtain feasible solutions, we employ both single-objective and multi-objective genetic algorithm (GA), and show the problems it faces in converging to the optimal sequence, mainly due to presence of multiple feasible sequences. Then, we formulate the problem as a mixed integer non-linear programming (MINLP), and obtain an optimal solution.

II. Meta-Heuristic Approach

A. Genetic Algorithm

Genetic Algorithm (GA) is a popular meta heuristic optimization technique inspired by the process of natural selection in biology. It is widely used to solve complex optimization problems, especially those that involve non-linearity, discontinuity, and non-convexity. The following demonstrates how the classic Genetic Algorithm works:

- ***Initialization***:The algorithm starts by creating a population of potential solutions randomly. Each solution in the population, also called **chromosome** represents a candidate solution to the problem.
- ***Fitness Evaluation***:Each solution in the population is evaluated for its fitness by applying an objective function. The objective function determines the quality of the solution in terms of the optimization criteria.
- ***Selection***:A subset of solutions is selected from the population based on their fitness values. The selection process is typically done using a probabilistic method such as roulette wheel selection or tournament selection.
- ***Crossover***:The selected solutions are paired up and combined to create new solutions. The process of combining solutions is called crossover, and it is done by exchanging a portion of the solution between two parents.

- ***Mutation***:The new solutions produced by crossover are then subjected to random mutation. Mutation introduces small random changes to the solution to create further diversity in the population.
- ***Repeat***:Steps 2 to 5 are repeated for a fixed number of generations or until a satisfactory solution is found. Each new generation of solutions is evaluated for fitness, selected, crossed over, and mutated to create a new population.

Through the iterative process of selection, crossover, and mutation, the genetic algorithm converges towards an optimal solution to the optimization problem. The algorithm can handle large and complex search spaces and can find near-optimal solutions in a reasonable amount of time. However, the performance of the genetic algorithm is highly dependent on the selection, crossover, and mutation operators used, and the appropriate parameter settings for the problem at hand.

B. Modified Genetic Algorithm

We extensively alter the original GA, and formulate a set of decision variables or chromosome for the problem at hand. A step-by-step approach is described for our version of GA, suited for permutations.

- **Encoding Scheme**: A solution or chromosome can be defined as a vector, representing a point in the search space for the problem. To represent our encoding scheme, we take up a case of compilation of 6 files on 2 servers. Let F_i define the i^{th} file, for $i \in \{1..6\}$. We express a solution S as

  $$S = [F_1\ F_3\ F_6 \parallel F_2\ F_4\ F_5]$$

  where the first set of files are compiled in the 1^{st} server in that sequence, and so on, with "$\parallel$" serving as a partition between compilation lists of every server. In code, S is represented as follows:

  $$S = [1\ 3\ 6\ 0\ 2\ 4\ 5]$$

  The numbers are synonymous with file numbers, and "0" is our partition symbol. This encoding is sufficient to express the complete process at hand, and the sequence in which it occurs. If a file with dependencies is lacking a required file, it can wait till it becomes available. However, if a time occurs when all the servers are in waiting state with non-empty queues, and no replications are ongoing, we flag it as a deadlock or non-feasible sequence.

- **Crossover:** The aim of cross over is to combine the characteristics of a pair of parent chromosomes to obtain two offsprings, each of which has a part of the parent chromosomes. We use a modified Linear Order Crossover *LOX* as described in [2]. A example is shown, and reasoning for selection of this variant.

  Assume the parents are: **123456789** and **769432158**
  Two crossing sites are chosen at random, say at positions 3 and 6.
  We get: **123**$\|$**456**$\|$**789** and **769**$\|$**432**$\|$**158**
  However, we cannot directly exchange the crossing sections yet, since the chromosomes need to be permutations, and the contents of the sections are not the same. So *egalization* is performed, as described in [2]. First, we take the digits in the 1^{st} parent and replace the corresponding digits of the second parent with *hole* "$-$ ". So we obtain the 2^{nd} parent as:

  $$7 - 9\| - 32\|1 - 8.$$

  Now,we slide the chromosome to the left, starting from the right end of the crossing section and then to the right, to get all the holes in the crossing section, thus obtaining:

  $$793\| - - - \|218.$$

  Now,we can insert the crossing section of the 1^{st} parent into that of the second, obtaining:

  $$793\|456\|218.$$

  This is the entire process for creation of the 1^{st} offspring. We repeat it again for the second offspring by replacing the order of the parents. We add a modification to this process. Since our partition indicators "0" are same across the chromosome, but they are different in the sense of the server lists it is dividing. So, if a "0" is to fall in the crossing section, it would only be matched with a "0" of same position. The reason for LOX variant over standard crossovers like Order Crossover (OX) is as follows. OX and similar operators treat chromosomes as circular, but this causes destruction of the priority structure for our chromosome. A feasible solution could become a infeasible one very easily, which we try to minimize.In other words, we preserve the high and low priorities in our chromosome parents, and pass it on to the offspring. A more detailed explanation can be found in [2].

- **Mutation:** Mutation is the process of diversifying offspring. Again, random changes cannot be done to our chromosome, so we opt for a simple binary inversion. We choose two positions and swap the digits present therein. We only perform this once, as multiple inversions can lead to worsening of the solution.

C. Objective Function

– **Single Objective:** We combine objectives for both file completion within deadlines, and tardiness measure into a single objective. Let there be I files, each with deadlines d_i, start times ts_i,compilation time $comp_i$ and goals g_i.The objective F, initialized with 0 can be defined as the following pseudo-code:

Algorithm 1 Single Objective Algorithm

```
for i ϵ I do
    if d_i − (ts_i + comp_i) >= 0 then
        F = F + g_i + d_i − (ts_i + comp_i)
    else
        continue
    end
end
```

– **Multi Objective:** As shown above, the goal points and points for finishing before the deadline can be split into two separate objective to maximize. One is the sum of goals, and the other the sum of speed points. In our experiments we find cases where the goals achieved may be the same, but the speed points may differ. So, we try to find pareto fronts, for the best sum of goals as well as the minimum makespan i.e time of operation. The dual objectives **goalpoints** and **speedpoints** can be defined as follows.

Algorithm 2 Dual Objective Algorithm

```
for i ϵ I do
    if d_i − (ts_i + comp_i) >= 0 then
        goalpoints = goalpoints + g_i
        speedpoints = speedpoints + d_i − (ts_i + comp_i)
    else
        continue
    end
end
```

D. Starting time calculation and deadlock detection

We use an iterative approach to figure out starting times or un-feasibility from the chromosome. We iterate through every server to see if they are able to compile the next file in their sequence. Every server maintains a local time. If in a epoch, we find no server is able to compile and the replication periods are over across other servers, we detect a deadlock and apply penalty.

III. Mathematical Formulation

The mathematical formulation of this file compilation problem is inspired by mixed integer linear programming(MILP) formulation of the classical Job shop scheduling problem (JSSP) [1].. In the JSSP we have the number of jobs and m number of machines and there is a cost associated with every job in every machine. Our objective was to get all the jobs done before the deadline at the total minimum cost.

In our problem we have n number of files and m servers. For every file we have deadline , compile time , replication time. Replication time is the time for compiled file to gets replicated in the other servers. Any file can be compiled on any server and it takes same time on all the servers. A file may also have some dependencies a file can not be compiled before all of its dependencies gets compiled. If a file is compiling on the same server in which its all of its dependencies gets compiled then it can start at any time, but if it is compiling on some other server it can only start after all of its dependencies gets replicated to current server. We have to compile all the files before their deadlines and maximize the total points.

The formulation we are going to discuss is for single objective mixed integer non-linear problem (MINLP). In our problem the main decisions involved are assignment of files on servers, sequencing of files on each server, and start time of all the files. I set the set of files we have and M is that of the servers. x_{im} is a assignment variable which is binary which is one if file i is to assigned on server m and zero otherwise. Another binary variable $y_{ii'}$ is a sequencing variable which takes the value one when both files i and $i^{'}$ are assigned to the same server and file $i^{'}$ is to be compiled after file i, otherwise it will be zero. Start time of a file i is indicated by the continuous variable ts_i. $comp_i$ is the compilation time, rep_i is the replication time of file i. $dep_{ii'}$ is the binary variable which is one if $i_{'}$ is the dependency of i, zero otherwise. t_i is a binary variable if it is one it means file i is one of our target file. The MINLP model using all these variables can be written as

$$max \sum_{i \epsilon I} \sum_{m \epsilon M} t_i(g_i + (d_i - (ts_i + c_i))) \tag{1}$$

s.t.

$$ts_i \leq d_i - \sum_{m \epsilon M} c_i x_{im} \quad \forall i \; \epsilon \; I \tag{2}$$

$$\sum_{m \epsilon M} x_{im} = 1 \quad \forall i \; \epsilon \; I \tag{3}$$

$$\sum_{i \epsilon I} x_{im} p_i = \max_i d_i \tag{4}$$

$$y_i + y_{i'} \geq x_i + x_{i'} - 1 \;\; \forall i, i' \; \epsilon \; I, i' \; > \; i, m \; \epsilon \; M \tag{5}$$

$$ts'_i \geq ts_i + \sum_{m \epsilon M} comp_i x_{im} - U(1 - y_{ii'}) \quad \forall i, i' \; \epsilon \; I, i \neq i^{'} \tag{6}$$

$$y_{ii'} + y_{i'i} \leq 1 \quad \forall i, i' \; \epsilon \; I, i' \; > \; i \tag{7}$$

$$y_{ii'} + y_{i'i} + x_{im} + x_{i'm'} \leq 2$$
$$\forall i, i' \epsilon\ I, i > i', m, m' \epsilon M, m \neq m' \quad (8)$$

$$ts_i \geq dep_{ii'}(ts_{i'} + comp_{i'} + x_{im}(1 - x_{i'm})rep_{i'}$$
$$\forall i, i' \epsilon\ I, i > i', m \epsilon M \quad (9)$$

$$ts_i \geq 0 \quad (10)$$

$$x_{im} \epsilon \{0, 1\}\ \ \forall i \epsilon\ \ I, m \epsilon\ M \quad (11)$$

$$y_{ii'} \epsilon \{0, 1\}\ \ \forall i, i' \epsilon\ \ I, i\ \neq\ i' \quad (12)$$

Objective(1) is to maximize the total points obtained by compiling all the target files before deadline. Constraint(2) ensures that start time of a file must be atleast $comp_i$ times ahead of it's deadline, if this contraint gets violated this will mean that file i not compiled before its deadline. Any file can be compiled on only one server constraint(3) takes care of that, for a given i x_{im} will be one for any one m and zero for others. Constraint(4) ensures that the total time for which any server is compiling the files must be less than the latest deadline, if this constraint gets violated this will mean that there is at least one file which is not compiled before its deadline, this constraint tightens the LP relaxation of the problem. Constraint(5) is based on the fact that if the files i and $i^{'}$ are assigned to the same server then they must be compiled one after another. In constraint(6) $U = \sum_{i \epsilon I} comp_i$, this is a sequencing constraint which makes sure that if binary variable $dep_{ii'}$ is one then $i_{\prime}$ must be compiled after i. Constraint(7) is logical cut, it is based on the fact that if i and $i_{\prime}$ are assigned to the same server then either of $y_{ii'}$ and $y_{i'_i}$ woul d be one, but if they are assigned on different machine then both $y_{ii'}$ and $y_{i'_i}$ would be zero. Constraint(8) is also a logical relationship which makes sure that if the files i and $i_{\prime}$ are assigned to different server then the sequencing variables in this equation will be zero. Constraint(9) ensures that if $i_{\prime}$ is the dependency of i and i is compiling on the same server on which $i_{\prime}$ has been compiled then start time of i must be finished time of $i_{\prime}$, but if i is compiling on some other server then start time of i must be after (finished time + replication time) of $i_{i'}$. Constraint(10) is a obvious constraint which makes sure that start time can not be negative. Constraints(11) and (12) makes sure that x_{im} and $y_{ii'}$ are binary variable.

IV. Results

In this section, we show examples of the file compilation problem that are solved using our proposed meta-heuristic and mathematical formulations. For mathematical formulations, each example is implemented on NEOS server's Mixed Integer Nonlinearly Constrained Optimization using DICOPT[GAMS]. For meta-heuristic method, we use MATLAB. Note that we make the goals and speed points negative to convert the problem to a minimization task.

Example 1 In our first example, we have six files and two servers, all other necessary data is given, we have to compile all the files before deadline and maximize the total points. The data for this example is given in Table 1.

TABLE I
DATASET FOR EXAMPLE 1

$File$	$compi$	rep_i	g_i	d_i	$dependencies$	t_i
1	15	5				0
2	10	18				0
3	15	35			1	0
4	13	32	8	40	2	1
5	20	52	15	100	2,3	1
6	15	21	35	53	3,4	1

TABLE II
OPTIMAL ASSIGNMENT OF FILES TO SERVERS FOR EXAMPLE 1, OBJ=102

$File$	$Server1$	$Server2$	ts_i
1	1		0
2		1	0
3		1	23
4		1	10
5		1	53
6		1	38

Fig. 1. **Convergence Plot for Single-Objective GA Example 1**

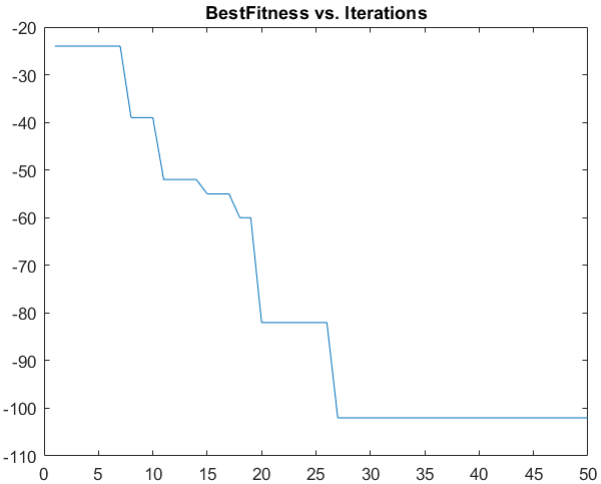
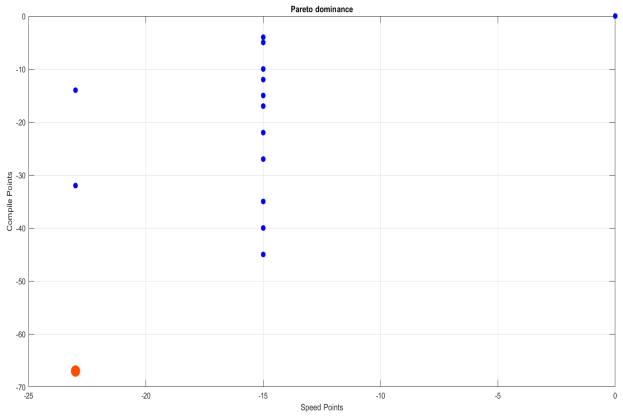


Fig. 2. **Pareto Plot for MOGA.** Red points indicate the first front.



The optimal solution via mathematical formulation is given in the table 2. If File i is compiled on server m then the corresponding cell entry 1, like file 1 is compiled on server 1 so in the column server 1 , the entry is 1 for file 1. ts_i is the start time of file i. The maximum points that can be obtained by compiling all the target files before deadline are 102.

We display the convergence plot for GA in figure 1. Since we have a relatively simpler example, we get convergence to optimal solution. In figure 2, we show the pareto plot from MOGA. For the same value of speed points, we get various different goal points in each front.

Example 2. In our second example, we have ten files and three servers, all other necessary data is given, we have to compile all the files before deadline and maximize the total points. The data for this example is given in Table 3. As shown in the convergence plot Fig 3, we obtain optimal solution by GA here as well.

TABLE III
DATASET FOR EXAMPLE 2

File	*compi*	repi	gi	di	dependencies	t_i
1	15	5				0
2	10	8				0
3	15	5			1	0
4	13	5	8	40	2	1
5	20	5	15	100	2,3	1
6	15	7	35	53	1,2,3,4	1
7	8	8	30	75	2,3,4,5	1
8	10	10	40	90	3,4,5	1
9	11	10	45	110	3,5,6,7	1
10	15	12	50	125	3,4,5,7	1

Example 3. In this example, we display the shortcoming of meta-heuristic approach. We have 20 files to be compiled on 3 servers. *For this harder example, GA is unable to converge or even find a feasible solution after 1000 iterations.*

V. CONCLUSIONS

In this paper, we strive to give a complete description of the problem and extensively try out optimization methods to achieve a solution. Here, we list out our observations.

TABLE IV
OPTIMAL ASSIGNMENT OF FILES TO SERVERS FOR EXAMPLE 2, OBJ=420

File	*Server*1	*Server*2	ts_i
1	1		0
2		1	0
3	1		15
4		1	10
5	1		30
6		1	35
7	1		50
8		1	55
9		1	66
10	1		58

Fig. 3. **Convergence Plot for Single-Objective GA Example 2**

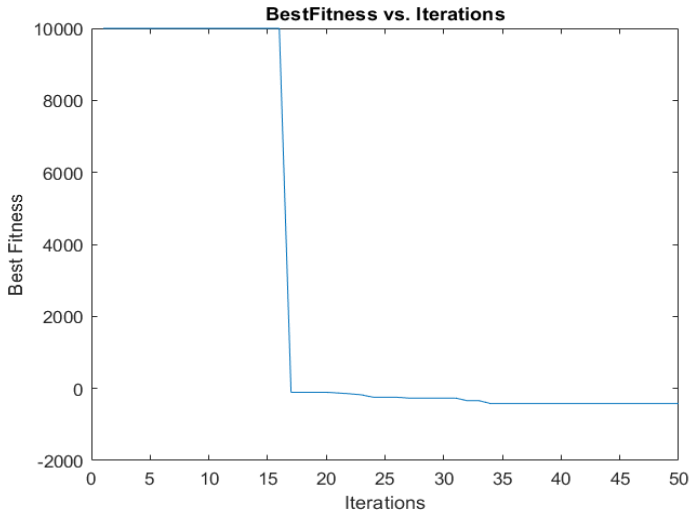


TABLE V
DATASET FOR EXAMPLE 3

File	$comp_i$	rep_i	g_i	d_i	*dependencies*	t_i
1	5	5	10	20		1
2	10	4	15	20		1
3	15	9	12	40	1	1
4	12	8	16	45	2	1
5	15	6	8	50		1
6	17	3	6	60	3,4	1
7	11	4	20	40	5	1
8	7	5	22	80	3,7	1
9	10	7	11	60	3,5	1
10	12	5	15	80	8,9	1
11	14	6	5	80	4,5,8	1
12	10	8	6	85	6,8,9	1
13	16	5	7	90	9,7,11	1
14	13	4	15	100	8,10,12	1
15	18	6	20	100	6,8,11	1
16	9	4	12	120	9,13,14,15	1
17	11	6	11	120	8,12,14,15	1
18	10	5	10	130	6,13,14,15	1
19	15	8	21	125	1,12,15,16,17	1
20	18	9	25	130	8,10,14,15,16	1

- The example (3) establishes the supremacy of mathematical formulation over the meta-heuristic techniques like GA. But both meta heuristic techniques and mathematical formulations has their own advantages and disadvantages. We can solve any black box

TABLE VI
OPTIMAL ASSIGNMENT OF FILES TO SERVERS FOR EXAMPLE 3, OBJ=714

$File$	$Server1$	$Server2$	$Server3$	ts_i
1	1			0
2		1		0
3	1			5
4		1		10
5			1	0
6	1			30
7			1	15
8			1	29
9		1		29
10	1			47
11			1	36
12		1		50
13		1		60
14	1			68
15			1	50
16		1		85
17			1	85
18	1			81
19			1	98
20	1			102

problem using metaheuristic techniques without even having its mathematical model while we can use mathematical formulations only if we have mathematical model for the problem. Mathematical formulations are computationally intensive but they guarantees the best solution(if formulated properly) while metaheuristic techniques does not give us any such guarantee so we need to test the problem for various initial populations but still we may not get optimal solution.

- Another major shortcoming of our meta-heuristic approach is the variance with respect to the random seed and hyper parameters. The optimal solution only arrives for certain seeds in a fixed number of iterations. Certain seeds may entirely fail to provide a feasible solution.

ACKNOWLEDGMENT

The authors would like to acknowledge the advice of Professor Prakash Kotecha, course instructor of CL643 (Computer Aided Applied Optimization) under whose supervision we undertook this project. We also appreciate the help provided by Mr. Rajani Kant Baro, Mr. Nikhil Anant Purao and Mr. Vidvendu Gupta in figuring out the GAMS software. We also acknowledge Google's Hashcode Competition 2019 Final Round's Problem statement and datasets, which gave us the idea for this project.

REFERENCES

[1] Vipul Jain and Ignacio E. Grossmann, "Algorithms for Hybrid MILP/CP Models for a Class of Optimization Problems" in INFORMS Journal on Computing/Vol. 13, No. 4, Fall 2001.

[2] E. Falkenauer and S. Bouffouix, "A genetic algorithm for job shop," Proceedings. 1991 IEEE International Conference on Robotics and Automation, Sacramento, CA, USA, 1991, pp. 824-829 vol.1, doi: 10.1109/ROBOT.1991.131689.

[3] Chaudhry, Imran Ali. (2012). A Genetic Algorithm Approach for Process Planning and Scheduling in Job Shop Environment.