Solving Scheduling Problems with Randomized and Parallelized Brute-Force Approach

Reggie Davidrajuh
University of Stavanger
Stavanger, Norway
e-mail: reggie.davidrajuh@uis.no

Chunming Rong
University of Stavanger
Stavanger, Norway
e-mail: chunming.rong@uis.no

ABSTRACT

Most of the scheduling problems are NP-hard problems. Thus, they do not have polynomial-time solutions. The literature review provides hundreds of methods and approaches to find polynomial-time near-optimal solutions. Most of these approaches are based on genetic algorithms. Genetic algorithms have the power of scanning most of the solution space, and they are not vulnerable to hill-climbing phenomena. However, as this paper shows, genetic algorithms cannot be used if the rate of production of healthy offspring is very low. Hence, this paper proposes a novel approach that is based on randomized brute-force and inspired by genetic algorithms. Also, the proposed approach uses parallel processing.

Keywords

Scheduling problems, genetic algorithms, the brute-force approach, parallel processes.

1. INTRODUCTION

This paper presents an approach to combine the powers of the brute-force approach, parallel processing, and genetic algorithms to solve scheduling problems in polynomial time.

Formally, the scheduling problems involve scheduling of jobs (series of tasks), while satisfying the constraints on the usage of resources and temporal precedence. As in this paper, a well-known class of scheduling problems is the job-shop scheduling in which resource constraints are the availability of processors (machines) for processing the tasks [1].

Scheduling problems are non-polynomial (NP) hard problems; thus, they do not have polynomial time solutions.

The literature review provides hundreds of methods and approaches based on genetic algorithms to find polynomial-time near-optimal solutions. Genetic algorithms are powerful as the newly made population (by the reproduction operations crossover and mutation) can be positioned in most of the solution space, and they are not vulnerable to hill-climbing phenomena. However, genetic algorithms cannot guarantee optimal solutions, as there is no assurance that any of the offspring eventually hit the optimal point. Also, genetic algorithms face another obstacle when applied to scheduling problems. In scheduling problems, the order of tasks has to be maintained all the time. When the reproduction operations (crossover and mutation) creates a new population in genetic algorithms, most of the population may become invalid as they do not satisfy the order of tasks. In this case, newer populations have to be created repeatedly just to make sure that a valid set of offspring is born. The computation time that is spent on repeated reproduction is usually ignored in the literature. This concealed time (the time that is spent on producing invalid specimens) can alone make the solution non-polynomial.

The brute-force is a simple approach that will result in optimal solution as every possible combination is checked. However, since every possibility is considered, the approach results in non-polynomial timings. This paper proposes an approach that combines the power of genetic algorithms with the simplicity of the brute-force approach. Also, parallel processing is applied to minimize the running time.

In this paper: Section-2 presents a concise introduction to genetic algorithms. Section-3 briefly introduces the brute-force approach. Section-4 proposes a new approach that combines the powers of genetic algorithm, parallel processing, and the brute-force approach.

2. GENETIC ALGORITHMS

A brief introduction to the genetic algorithm (GA) is given in this section. This brief introduction is only to make this paper self-contained. For a complete study of GA, the interested reader is referred to the following textbooks [2, 3].

GA is for solving optimization problems. GA follows the natural process of biological evolution. GA involves the following steps, as shown in Figure-1:

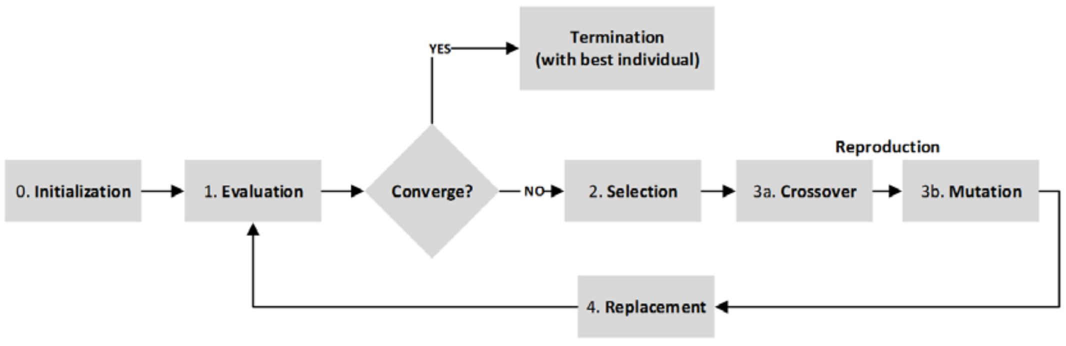

Figure-1: The steps involved in genetic algorithms

0. Initialization: This is the first step. A group of initial population is used to start the GA iterations. The iterative run of the algorithm involves steps 1-4.
1. Evaluation: Each individual is measured for its fitness.
2. Selection: A group of individuals are selected for reproduction; this is done randomly with a probability depending on the relative fitness of the individuals so that the best ones are often chosen for reproduction. Also, a fair chance is given to others that may include less-fit individuals too.
3. Reproduction: Reproduction of a new population from the selected individuals: new chromosomes are made by the reproduction operations such as crossover and mutation.
4. The newly created population replaces part of the older population.
5. Termination: The algorithm is stopped when the population converges towards the optimal solution.

3. BRUTE-FORCE APPROACH

The brute-force approach is an exhaustive search approach. In the brute-force approach, all the possible combinations are enumerated, and each of these combinations is checked one-by-one. Since all the possibilities are checked, the approach finds the optimal solution. Due to the exhaustive nature, the approach also results in non-polynomial timing for NP-hard problems. However, if the problem size is small, it is usual to apply the brute-force approach even for NP-hard problems, due to its simplicity. For a more detailed study on the brute-force approach, the interested reader is referred to [4, 5].

The brute-force approach is simple as it usually consists of an iteration involving the following steps:

0. Initialization: Let the number of iteration $i = 0$.
1. Increase i by one. Find a new combination c of the input parameters.
2. Use c to find the solution s. If s is a better solution than the previous solutions, then save (c, s).
3. If s is equal or better than the expected solution s_{OPT}, then stop. If $i = i_{max}$ where i_{max} is the maximum allowable iterations, then stop. Else, go to step-1 to generate the next unique combination c.

4. THE NEW APPROACH

The new approach is introduced through an example.

4.1 Example

A grid computing facility ('grid' for short) consists of three processors (P1, P2, and P3) that are capable of performing any jobs. The grid is to perform four jobs, and each consists of three sequential tasks ('*t*' for short). Temporal dependencies between the twelve tasks are shown below.

Job-A: tA_1 -> tA_2 -> tA_3
Job-B: tB_1 -> tB_2 -> tB_3
Job-C: tC_1 -> tC_2 -> tC_3
Job-D: tD_1 -> tD_2 -> tD_3

This means, tX_j cannot be started before the completion of tX_i, where X belongs to {A, B, C, D}, and $1 <= i < j <= 3$. However, since there is no dependency between tX_i and tY_j for different X and Y, these tasks can be performed independent of each other.

A scheduling problem is to find the optimal allocation of the processors so that all the four jobs are completed with the least possible time (minimal completion time).

Since there are twelve different tasks (*tA1* to *tD3*), the total number of combination is about 479 million (exactly, factorial(12) = 479,001,600). However, out of these 479 million sequences, only 396 k (exactly, 369600, resulting from 12C3 * 9C3 * 6C3) are valid ones. The valid combinations are the ones that obey the dependency constraints, e.g., the start timings of *tA1* < *tA2* < *tA3*, etc.

The total number of combination: c_T = 479,001,600
The number of valid combination: c_V = 369,600
The reproduction success ratio: $rep_{SR} = c_V / c_T$ = 0.08%

Since the ratio (***rep**$_{SR}$*) is extremely small (0.08%), we cannot use the genetic algorithm. This is because all of the new specimens that are made by reproduction will most probably (99.92%) be invalid ones. In other words, on average, for every 1296 specimens reproduced, only one will survive. This means a tremendous amount of time will be wasted on reproduction than on solving the problem.

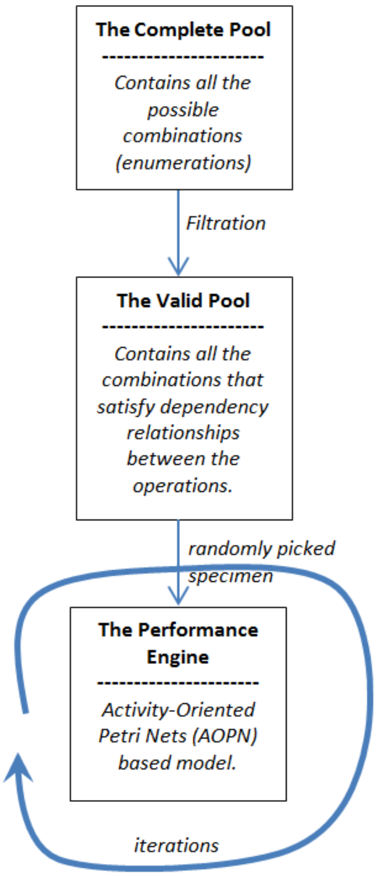
The Complete Pool
Contains all the possible combinations (enumerations)
Filtration
The Valid Pool
Contains all the combinations that satisfy dependency relationships between the operations.
randomly picked specimen
The Performance Engine
Activity-Oriented Petri Nets (AOPN) based model.
iterations

Figure-2: Towards the new approach: three-step process

4.2 Towards the New Approach

The new approach starts as the brute-force approach, see Figure-2. This approach consists of three steps:

- Step-1 "The complete pool": All the possible combinations are found. Though finding the unique combinations itself amounts to non-polynomial timing, an efficient "dynamic programming" based solution can be used [6].
- Step-2 "The valid pool": Each member of the pool is tested for validity. The valid ones are added to the valid pool.
- Step-3 "The Performance Engine": The valid pool is much smaller in size than the complete pool. Hence, the

valid pool can be used as the newly reproduced specimens. For example, iteratively, a random specimen can be taken from the valid pool and fed into the performance engine for fitness evaluation. As usual in the genetic algorithms, the iterations can be stopped once the maximum number of iterations are run, or a specimen with equal to or more than the expected value is found.

4.3 The Simulation
For simulations, the following timing is assumed (all in units of time TU):

Job-A:	$tA_1 = 3$	$tA_2 = 5$	$tA_3 = 6$
Job-B:	$tB_1 = 8$	$tB_2 = 4$	$tB_3 = 9$
Job-C:	$tC_1 = 2$	$tC_2 = 7$	$tC_3 = 5$
Job-D:	$tD_1 = 3$	$tD_2 = 2$	$tD_3 = 6$

The performance engine for fitness evaluation is an Activity-Oriented Petri Net (AOPN). The AOPN (shown in Figure-3) is designed following the guidelines given in [7]. The AOPN is realized with GPenSIM software [8].

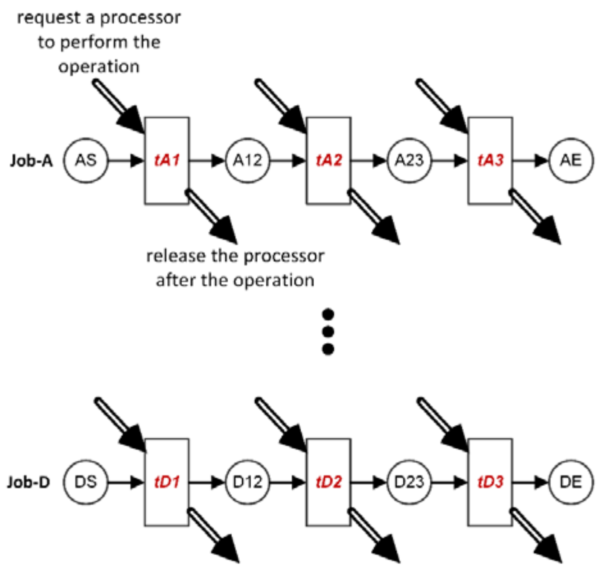
request a processor to perform the operation
Job-A
AS
tA1
A12
tA2
A23
tA3
AE
release the processor after the operation
Job-D
DS
tD1
D12
tD2
D23
tD3
DE

Figure-3: Activity-Oriented Petri Net (AOPN) as the performance engine for measuring the fitness.

The simulation reveals that the least completion time is 21 TU. The completion time of 21 TU is also theoretically the optimal value. The simulation captures all the best performing (in the completion time of 21 TU) combinations. Also, if necessary, the worst performing ones can be captured too. The complete code cannot be shown in this paper due to space limitation. However, the complete code is given in [9]. The interested reader is welcome to download the code and reproduce the results.

Finally, in the Petri Net, it is assumed at all the three processors can perform any of the tasks *tA1* to *tD3*. However, this may not be practical as some of the processors are dedicated, and can perform only specific tasks (e.g., P1 can only perform tA_1, tB_3, and tD_2). This kind of situation (dedicated processors) can be very easily accommodated in the Petri Net, by making some finite changes in the simulation code.

4.4 The New Approach
In the approach shown in Figure-2, the three stages are sequentially executed. This means Stage-2 for creating the valid pool can happen only after the generation of all the possible combinations are completed in Stage-1. As the generation of all the possible combinations will take a long time, the sequential execution of the three stages will certainly result in long delay. Thus, the new approach proposes a parallel version of the sequential approach shown in Figure-2. In the parallel version, shown in Figure-4, all the three stages are run as parallel processes.

In the approach shown in Figure-4, the *generator*, the *validator*, and the *performance engine* are run in parallel, as parallel processes. The generator is responsible for generation of all the unique combinations. The validator receives one combination at a time from the generator and validates it. The valid combinations are saved in the validator. The performance engine receives one valid combination at a time from the validator. The performance engine finds the completion time of the combination, and saves the result, if it is better than the previous results.

It must be re-iterated that the generator and the validator blocks are run separately and in parallel (not combined) is to minimize the time spent on generating the valid pool.

5. DISCUSSION
The generator dominates the running time of the new approach shown in Figure-4 is as it is the generator that is going to take a long time to run. The generator plays the “master” role here, as the other two are slaves seeking the combinations coming out of the generator.

If a polynomial-time execution is needed, then the generator (and the other two slaves) can be stopped if any of the following three conditions are met:
1. The performance engine has already found a satisfactory result.
2. The maximum allowable combinations are checked.
3. The maximum permissible execution time has passed.

If none of the conditions is met or simply ignored, then the iterations run forever until all the valid combinations that are produced by the generator are processed. In this case, the randomized brute-force approach becomes a pure brute-force approach.

In Figure-4 (as well as in Figure-2), it is clear that the reproduction stage is wholly omitted as it serves no purpose by producing invalid offspring. However, the randomness involved in the selection stage of genetic algorithms is missing in the new approach. The randomness in the new approach is completely dependent on the random generation of unique combination by the generator. Only the generator can exercise randomness in generating the combinations. As the other two slave processes work on the combinations they get from the generator, there is no randomness involved in these two processes. Thus, the power behind the success of the genetic algorithms - the randomness in selection and reproduction - is lost to a high degree in the proposed new approach.

Further Work: the simulations that are run for this paper are limited to the sequential approach shown in Figure-2. However, the more efficient approach shown in Figure-4 (that uses the three processes that are run in parallel) was not tested. Thus, making a prototypical system based on Figure-4 is proposed as further work.

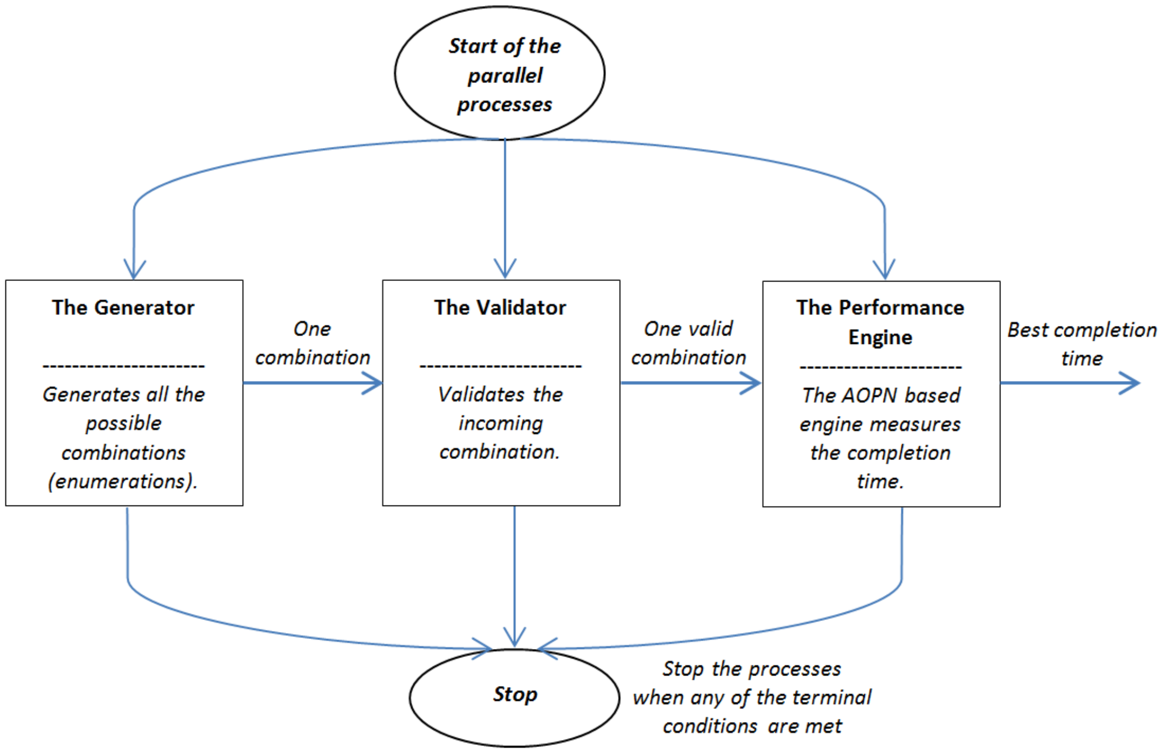
Start of the parallel processes
The Generator
Generates all the possible combinations (enumerations).
One combination
The Validator
Validates the incoming combination.
One valid combination
The Performance Engine
The AOPN based engine measures the completion time.
Best completion time
Stop
Stop the processes when any of the terminal conditions are met

Figure-4: The new approach based on the three parallel executed processes.

6. ACKNOWLEDGEMENT

The authors would like to thank professor Abdulrahman M. Al-Ahmari of College of Engineering, King Saud University, for giving a problem on scheduling. The authors made this paper while solving the problem, which closely resembles the one given in this paper.

REFERENCES

[1] F. Hutter, H. H. Hoos, & T. Stützle, T. Efficient stochastic local search for MPE solving. In *IJCAI*. pp. 169-174. 2005.

[2] Z. Michalewicz, Genetic algorithms+ data structures= evolution programs. Springer Science & Business Media, 2013.

[3] K. Deb, “Multi-objective optimization.” *Search methodologies*. Springer, Boston, MA, 2014.

[4] Johan Sannemo, Principles of Algorithmic Problem Solving. Draft version. 2018.

[5] B. N. Miller, and D. L. Ranum, Problem solving with algorithms and data structures using python Second Edition. Franklin, Beedle & Associates Inc., 2011.

[6] R. Davidrajuh, “Finding the enumeration of a sequence by dynamic Programming.” *Lecture Notes 07 on Algorithm Theory, University of Stavanger*, Norway. 2016.

[7] R. Davidrajuh, “Outperforming genetic algorithm with a brute force approach based on activity-oriented petri nets.” *International Joint Conference SOCO’16-CISIS’16-ICEUTE’16. Springer*, Cham, 2016.

[8] GPenSIM Website: http://www.davidrajuh.net/gpensim/

[9] Complete Simulation Code. Available from: http://www.davidrajuh.net/gpensim/Pub/2019-ICIT/