CA2 Team-L Report: Dynamic Stacking Optimization in Uncertain Environments

ECMM409 Natural Inspired Computation

I INTRODUCTION

The dynamic stacking problem has a wide range of applications in many industries. For example, in the container shipping industry millions of containers need to be stacked for storage and transport each year. The efficient movement of each container to its location is an issue that needs to be studied. This includes the handling of containers in the arrival area so that there is always free space in the arrival area to accept the next arriving container, and that the containers are stored appropriately and delivered in time for the scheduled time of delivery

Regarding the container problem we have simplified it to a dynamic stacking scenario. The blocks with due time represent each container, a crane is responsible for moving each block, and the three areas are the arrival area, the buffer area and the delivery area. If the arrival area is full, no additional blocks can be reached. When each block is converted to a ready state after expiry, the crane moves it from the buffer stack to the delivery area for delivery. Each area has a fixed amount of space in the stack and if the space is full no more blocks can be stored.

There are three objectives to be achieved in this scenario: (1) Avoid delays in the arrival area as much as possible. When the arrival area is full, the next upcoming block stops arriving and therefore creates delays. (2)Deliver as many blocks as possible on time. When a block's arrival is due, deliver this block as quickly as possible. (3) Minimize crane operations so that the crane can operate more efficiently.

II MODELING AND TASK DIVIDE

The model is divided into two parts: the simulation and the evolutionary algorithm. With regard to building the simulation environment, it contains an arrival stack, three buffer stacks, a handover stack and a crane. set the capacity of the arrival stack to 5, the capacity of each buffer stack to 7 and the capacity of the delivery stack to only 1. Each operation of the crane consists of four parts: (1) moving to the stack where the block must be picked up, (2) grabbing the topmost block from the stack, (3) moving to the stack where the block should be dropped, ((4) release the block to the top of the stack. By default the crane takes 1 for each step of the operation and the crane can only move one block at a time. In the arrival stack, each new block is generated at random intervals at the bottom of the arrival stack (i.e. following the FIFO principle) and if the arrival stack is full, new block generation stops, at which point the arrival stack state is blocked and the blocking time is recorded. If the due time for this block is reached, the crane needs to move the block to the handover stack If the block in the arrival stack does not reach the due time, it will be placed on top of the buffer stack, which follows the first-in, last-out principle. When the block in the buffer stack reaches the due time, it changes to the ready state and waits for the crane to move it into the handover stack.

For the evolutionary algorithm, a certain number of blocks enter the whole environment from the arrival stack, the crane randomly selects a block (containing the blocks in the arrival stack and the blocks in the buffer stack) to operate on as a solution and determines whether it is a valid operation: whether this block is at the top of the stack. If so, the current crane operation is a valid operation and a valid solution, if not, it is chosen at random from a new selection. The above operation is then repeated until all block operations are valid and all solutions produced form an initial population. This is followed by a discussion of the best solution to be achieved using a suitable evolutionary algorithm. The optimal solution needs to achieve three objectives: (1) minimize blocking time, (2) arrive at the total number of blocks on time, and (3) minimize the number of crane operations.

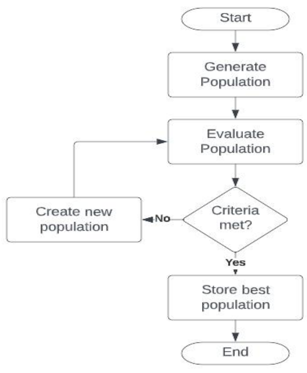
For the evolutionary algorithm, the task can be broken down, into manageable chunks that group members can work on in parallel, into the following: research and analysis of potential solutions; research into fitness functions, selection, mutation operators, etc.; experimentation; research into how we may improve our algorithm; developing a simulation (design, development and testing); developing an evolutionary algorithm (design, development and testing).

III ANALYSIS AND DISCUSS

As the scenario requires the optimization of three objectives, i.e. Minimize Blocked Arrival Time, Maximize Total Blocks on Time and Minimize Crane Manipulations, the easiest way to think about this is to use a multi-objective evolutionary algorithm. Multi-objective evolutionary algorithms are based on the idea of genetic algorithms, which simulate the process of biological evolution to find the approximate optimal solution. First, a random initial solution is found, then different solutions are generated by crossover and mutation, and the approximate optimum is found by using a 'Pareto front'. However, in this container model, generating a random initial solution is difficult because each solution needs to contain the complete lifting solution for the whole environment, which is exactly what we need to ask for, and therefore cannot be generated directly. So using a multi-objective evolutionary algorithm is not applicable to this container model.

Ant-Colony Optimization (ACO) has withstood many tests and has proved effective at solving static optimization problems (problems that do not change over time). However, traditional ACO would not suffice for dynamic problems, like the one we face, as the system state is constantly evolving over time. Dynamic Ant-Colony Optimization (DACO) has been used to find an optimal solution for a set of circumstances from a previous optimal solution from different circumstances (e.g. the change in circumstances after each crane move/second passed). DACO was investigated based on the travelling salesman problem and the results strongly suggested that a dynamic ACO algorithm can quickly find optimal solutions to future iterations given the parent optimal solution compared to finding each optimal solution from scratch[1]. This could apply to the Dynamic Stacking problem as each optimal move can be decided with help of the previous optimal move.

A final approach can follow a traditional evolutionary algorithm including selection, mutation, and evaluation. With the following general approach we can treat the problem as a static optimization problem by evaluating each state separately using a similar algorithm to the figure. The population will be made up of randomly generated solutions. A single solution is made up of a block, its source, and destination. Both block and destination can be randomly produced, source is dependent on the block. A fitness can be decided of each solution based off of the following: (1) Is the block ready? (2) Can it be put on the handover in time? (3) Time until the due date? (4) How many other blocks in the stack are due or near due? We can then



perform selection, apply nature inspired operators to generate new solutions, evaluate the new solutions and repeat until we converge at some value. Some solution is then passed back to the simulation, where the move will be processed and then the next state is passed to the EA, where it will process for the next best solution again[2]. We decided to use this approach due to our familiarity with this approach and also because we are unsure how we would go about setting the pheromone.

IV EXPERIMENT AND DATA ANALYSIS

We have executed combinations of different parameters, and each for 10 times (with different initial population) to get the average. I execute totally over 1000 times to get all the data. All the data I do not use is stored in Dynamic Stack Data.xlsx. Evaluations goals: Crane Moves minimize, Block Arrive Time minimize, Delivered Block maximize.

Notice: The number in the table indicate the average fitness of each try before and after EA. Where the Bold number means average of total try, the percentage means the improvement of the parameter using EA.

GRAPH 1 Crane Moves under Different Parameters

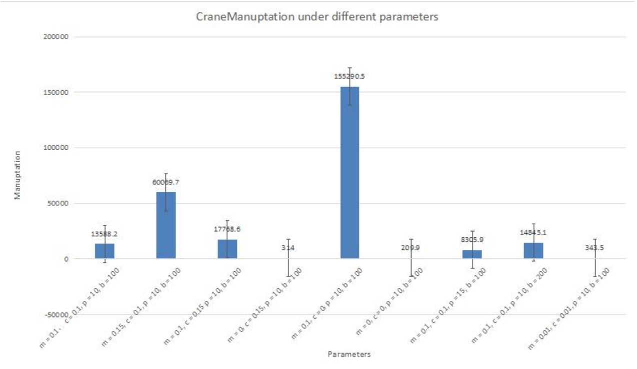


TABLE 1 Crane Moves under some Parameters

m = 0.1, c = 0.1, p = 10, b = 100	**m = 0, c = 0.15, p = 10, b = 100**
18462	337
15402	345
12249	289
12991	315
6200	287
12129	330
12031	329
8493	302
18523	320
19402	286
13588.2	**314**
m = 0.1, c = 0, p = 10, b = 100	**m = 0, c = 0, p = 10, b = 100**
300174	211
93770	210
46283	207
34735	215
105028	209
75888	215
161121	205
114065	207
385375	209
236466	211
155290.5	**209.9**

From table 1 and graph 1, we can see that the mutation does not act well for this problem. The way I set the mutation is to change the destination of the block under a certain rate. The cross over is act as switch the destination of two acts(described in section 2 of the report). In this chart, if we remove mutation, then the result turns to be better. But if we remove crossover, the result will be very unstable. The fact is when I use debug mode to find the reason, I figure out that the due date and ready time generation of some block is not appropriately. So if we just remove cross over, then it will cause the system is not good at identify "Move a block to another buffer" and "Hand over a block" because there fitness is really closed. Actually in this task we need to consider hand over more. If a block is ready, then hand over it is much more important than move another block from the arrive stack.

GRAPH 2 Block Arrive Time under Different Parameters

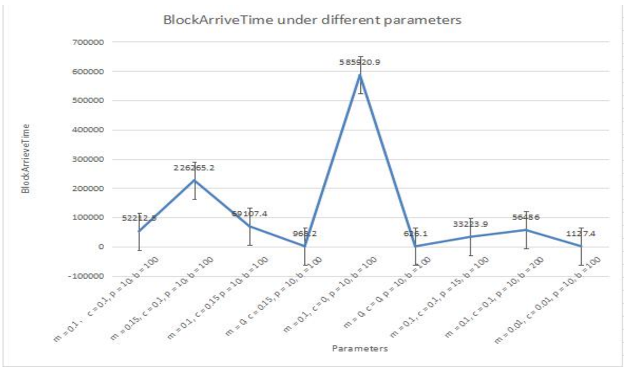


TABLE 2 Block Arrive Time under some Parameters

m = 0.1, c = 0.1, p = 10, b = 100	**m = 0.1, c = 0.1, p = 15, b = 100**
71816	33935
59948	39737
46147	58201
50416	32008
23674	36384
46538	9739
44921	16191
31814	59748
72819	26149
74032	20147
52212.5	**33223.9**
m = 0.1, c = 0.1, p = 10, b = 200	**m = 0.01, c = 0.01, p = 10, b = 100**
45205	1286
71453	999
57507	1300
59481	1087
59181	994
86888	1168
49401	1099
28162	1130
91948	1291
15634	920
56486	**1127.4**

In table 2 and graph 2, we can simply generate that when the scale of the task improved, the performance will not decrease too much. This indicate that our methods may act better when the total amount of the block have reached a large amount. And it is easy to generate that if we increase the population from 10 times of the block in the world to 15 times, the performance increase brilliantly. So this will give me the hint that evolutionary problems is better for large scale multi-objective optimization.

GRAPH 3 Delivered block under Different Parameters

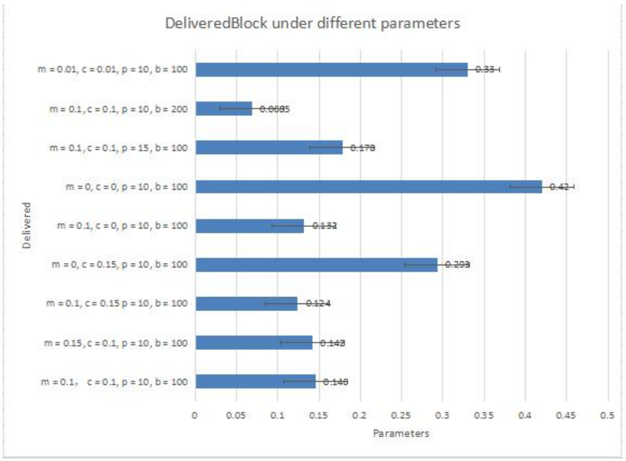


TABLE 3 Delivered Block under some Parameters

m = 0.1, c = 0.1, p = 10, b = 100	**m = 0.01, c = 0.01, p = 10, b = 100**
0.24	0.4
0.12	0.36
0.15	0.28
0.1	0.34
0.19	0.29
0.13	0.37
0.16	0.27
0.1	0.32
0.11	0.37
0.16	0.30
0.146	**0.33**
m = 0.15, c = 0.1, p = 10, b = 100	**m = 0.1, c = 0.15 p = 10, b = 100**
0.12	0.13
0.1	0.12
0.11	0.15
0.07	0.13
0.08	0.14
0.12	0.08
0.09	0.1
0.5	0.06
0.1	0.16
0.13	0.17
0.142	**0.124**

In table 3 and graph 3, there is another exploration of whether the rate is too high for the problem. After I have reduced the rate of mutation and cross over, the performance increase rapidly. For this problem, I think more rounds of evolution is better than higher mutations and cross over. Because all the block we can control is those on the top of the block(Arrive stack is a queue though), so we need to consider the influence that one move carries out to all the blocks in the system at that state. If we execute the mutation often, then the depth of search may decrease. This will lead to less valuable move.

V CONCLUSION

We chose to solve a dynamic stacking problem where, explained briefly, blocks with certain due dates arrive at an arrival area and need to be moved to a delivery area by a crane. The model for solving this problem is divided into a simulation environment and an evolutionary algorithm, which aims to find the most efficient move for the crane to make. The move aims to; minimize delays in the arrival area, delivering as many blocks as possible on time, and minimizing the number of crane operations. The effectiveness of this model is demonstrated through the experimentation of the evolutionary algorithm on the simulation environment, which allows for testing of various solutions under different sets of parameter settings.

There are still some ideas that we do not carry out in the experiment part. The problem is very close to traditional chess game problem as identifying the value of one move is more important. If we handle it using DFS and other Game algorithm to compare, they may act better on a limited amount. But EA will give better result when the object to be optimized increase and the scale of the problem increased. Unfortunately because of the low Computational power of personal computer, we cannot increase the block more.

In this experiment, we conclude that for dynamic stacking problems, more times of evolution and appropriate cross over will lead to better experimental results. Of course, mutations and cross should occur relatively infrequently. Also, for better results, the population per generation should be increased as memory allows. When the problem tends toward large-scale optimization, our solutions tend to get better.

VI REFERENCES

[1] Angus, D., Hendtlass, T. Dynamic Ant Colony Optimisation. Appl Intell 23, 33–38 (2005). https://doi.org/10.1007/s10489-005-2370-8

[2] Sureja, Nitesh & Dwivedi, Vedvyas. (2012). A nature inspired approach to solve dynamic travelling salesman problems. CiiT International journal artificial intelligent systems and machine learning. vol 4. 380-384.

[3] Sebastian Raggl, Andreas Beham, Stefan Wagner, and Michael Affenzeller. 2020. Solution approaches for the dynamic stacking problem. In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (GECCO '20). Association for Computing Machinery, New York, NY, USA,1652–1660.https://doi-org.uoelibrary.idm.oclc.org/10.1145/3377929.3398111

[4] Beham, A., Raggl, S., Karder, J., Werth, B., & Wagner, S. (2022). Dynamic Warehouse Environments for Crane Stacking and Scheduling. *Procedia Computer Science*, *200*, 1461-1470.

[5] Raggl, S., Beham, A., Wagner, S., & Affenzeller, M. (2020). Effects of Arrival Uncertainty on Solver Performance in Dynamic Stacking Problems.