Dynamic Impact for Ant Colony Optimization algorithm

Jonas Skackauskas[1], Tatiana Kalganova[1], Ian Dear[1], Mani Janakiram [2]

Abstract

This paper proposes an extension method for Ant Colony Optimization (ACO) algorithm called Dynamic Impact. Dynamic Impact is designed to improve convergence and solution quality solving challenging optimization problems that have a non-linear relationship between resource consumption and fitness. This proposed method is tested against the real-world Microchip Manufacturing Plant Production Floor Optimization (MMPPFO) problem and the theoretical benchmark Multidimensional Knapsack problem (MKP). Using Dynamic Impact on single-objective optimization the fitness value is improved by 33.2% over the ACO algorithm without Dynamic Impact. Furthermore, MKP benchmark instances of low complexity have been solved to a 100% success rate even when a high degree of solution sparseness is observed. Large complexity instances have shown the average gap improved by 4.26 times.

Keywords: Ant Colony Optimization, Dynamic Impact, Sub-heuristics, Scheduling, Multidimensional Knapsack Problem, Sparse data

1. Introduction

Combinatorial optimization is a fundamentally difficult task to be computed. Most real-world optimization problems are NP-Hard, which means they are too large to check all combinations in a reasonable amount of time to find the optimal result. Instead of brute-forcing the optimization, often metaheuristic methods are used to reach a "good enough" solution quickly. We are using Ant Colony Optimization (ACO) metaheuristic algorithm to solve real-world Microchip Manufacturing Plant Production Floor Optimization (MMPPFO) problem and theoretical Multidimensional Knapsack problem (MKP). ACO algorithm is generally good at solving both of these problems. However, this work aims to improve the algorithm's performance, and solution quality to a level never reached before. This paper introduces a sub-heuristic search method for the Ant Colony Optimization algorithm called Dynamic Impact. The paper also provides insight into how Dynamic Impact can be used for any constrained optimization problem. This method is then used to solve real-word MMPPFO problem and solve a

[1] College of Engineering, Design and Physical Sciences, Brunel University London, United Kingdom

[2] Intel Corporation, Arizona, United States

theoretical MKP for further validation and compare ACO with Dynamic Impact solution results to peer published work and their results.

1.1 Ant Colony Optimization

Ant Colony Optimization (ACO) is a nature-inspired optimization algorithm that uses Ants as search agents navigating a search space. Navigation is mediated by pheromones that ants are naturally drawn towards. While an ant is navigating, it deposits pheromone on its path, therefore attracting even more ants. Originally Ant Colony Optimization algorithm has been designed for the travelling salesman problem (TSP) described in Dorigo [1] doctoral thesis in 1992. Since then, ACO has been applied to a much more comprehensive range of optimization problems like scheduling problems [2] [3] [4] [5], subset problems [6] [7], routing problems [8] [9], and more further reviewed by Dorigo et al. [10]

1.2 Microchip manufacturing

Microchip manufacturing is a complex process that utilizes expensive machinery. Tight manufacturing schedules are used to run operations at maximum efficiency and minimize machinery downtime while always maintaining products' optimum stock levels. Often predicted microchip demand does not meet observed real demand, and the microchip production schedule must be altered accordingly to meet the newly specified demand.

Microchip manufacturing scheduling problems have been researched from various points of view. Scheduling robotic arms of two-cluster tools in microchip manufacturing facilities [11], transport scheduling in automated material handling systems for wafer manufacturing plants [12], wafer production scheduling as a job shop scheduling problem [13]. This research on the MMPPFO problem approaches as a resource-constrained production scheduling problem.

Microchip manufacturing plant production floor scheduling is a difficult task. The nature of the problem does not allow to have precalculated static heuristic information on each edge that enables ants to navigate the search space efficiently.

1.3 Multidimensional Knapsack Problem

In addition to solving a real-world optimization problem, this research further proved the validity of proposed methods by solving a theoretical Multidimensional Knapsack Problem (MKP) and comparing the results with previously published research papers. The goal of MKP is to maximize the total profit of the items taken into knapsacks, where all items have multidimensional weights for each knapsack, and each knapsack has a capacity that must not be exceeded [7]. The nature of packing different size items in all knapsacks simultaneously makes the feasible region of the search very sparse

[14]. Such sparsity is a great challenge for optimization algorithms where good solutions are obtained by iterative convergence.

1.4 Sub-heuristics

Combinatorial search algorithms are designed to explore large search spaces efficiently and converge to a good solution quickly. The efficiency is achieved using metaheuristic methods that allow the search space to be explored more in areas of greater reward. In the case of Ant Colony Optimization, the primary metaheuristic method is a pheromone, which is learned iteratively when solving the optimization problem. However, using pheromone alone does not make a performant optimization engine and often requires many iterations to reach a good solution.

With the introduction of additional heuristics to ACO, much higher optimization performance can be achieved. The first attempt by Stützle and Hoos [15] is the introduction of heuristic information that significantly improved search convergence speed and solution quality. The heuristic information uses precalculated values for each edge that indicate the general quality of that edge. It guides the ant's search in addition to the pheromone which is learned.

As the case of all algorithms, ACO has tunable hyper-parameters that once found a good combination of them for a specific problem, it can give much better results than just using default values. It can be very time consuming or require expert knowledge of the problem domain to find a good set of hyper-parameters. The hyper-heuristic methods were introduced by Burke et al. [16] which fine-tunes hyperparameters of the search algorithm in an automatic way for genetic algorithm. Also, these methods were applied to the ACO algorithm too [17] [18] [19].

In pursuit of better algorithm performance, the ACO algorithm can be coupled with problem-specific local search for some optimization problems. The local search takes a completed candidate solution and applies a search operator that follows a deterministic set of rules to improve it before it is used to lay down the pheromones [20] [21].

Furthermore, for certain problems, purpose-made sub-heuristic enhancements can improve search efficiency, converge faster, and produce better final solutions. Sub-heuristics are the heuristic methods used in the core of search algorithms [22]. Sub-heuristics for ACO is not a well-researched area. Therefore in this paper, we clarify sub-heuristic as an additional core search method that acts upon the state of an incomplete partial solution. The existing introductions of Sub-heuristic enhancements are done for specific problems and not generalized in any way. Authors [23] have utilized such sub-heuristics for ACO algorithms for probability calculations where branching can occur while building the solution. This sub-heuristic method allowed them to have a transition operation that otherwise could not be accounted for from previously explored solutions. The sub-heuristics are distinctly different from more commonly used algorithmic augmentations such as local search or hyper-heuristics. The sub-heuristic method is a primitive component of the metaheuristic search core and can be used independently of local search and hyper-heuristics.

The contributions of this research to science are as following:

- Introduction of Dynamic Impact a sub-heuristic method for the Ant Colony Optimization algorithm that helps more accurately calculate edge's probability while considering remaining constrained resources of the problem and non-linear fitness.
- Description of the methodology to apply Dynamic Impact effectively for a broad range of problems, both academic and real life.
- Proof that Dynamic Impact is beneficial for finding better results using the same computational efforts in theoretical MKP and real-life MMPPFO problems.
- Additionally, this research also shows that ACO with Dynamic Impact is superior in finding better solutions for MKP benchmark instances than previous peer research. Ant sets ACO with Dynamic Impact as the new state-of-the-art for benchmark MKP.
- This research also provides MMPPFO problem description and implementation details and the dataset used for this research for verification and repeatability studies.

The rest of this paper is organized as follows. Section 2 explains in detail the Dynamic Impact method for the Ant Colony Optimization algorithm with a simple usage example. Section 3 introduces the MMPPFO problem definition, constraints, and objectives. Section 4 is dedicated to the study of the Dynamic Impact method. The study is conducted for two optimization problems MMPPFO and MKP. Also, the exact implementation is of Dynamic Impact is described for each of the problems. Then both problems are solved using Ant Colony Optimization with Dynamic Impact and compared to solutions obtained without Dynamic Impact. Furthermore, for external validation, the MKP problem is compared to other recently published research results. Finally, conclusions and future directions are presented in Section 5.

2. Ant Colony Optimization algorithm with Dynamic Impact

2.1 Ant Colony Optimization algorithm

Frequently used Min-Max Ant System introduced by Stützle and Hoos [15] is used as a baseline for the proposed Dynamic Impact Algorithm extension and experimental work. The first step of algorithm execution is search space initialization, in which search space is filtered for all nodes to have only feasible edges. Then each edge pheromone is set to the maximum value of τ_{max}, also each edge j of the node i gets a precalculated static heuristic value $\eta_{j,i}$. Once the search space is prepared, the iterative search starts. In the iterative search, a set of ants that each builds a complete solution. Each ant starts building with an empty partial solution $s_p = \emptyset$. Then the ant searches for a single edge to add to the partial solution. The ant can choose of two modes of finding an edge. First is the exploration mode, where each edge is added stochastically to the solution, and second is the exploitation mode, where only the best edge is selected deterministically. The ant search mode is chosen stochastically with $q0$ exploitation

parameter for each edge added to the partial solution. The following probability equation of the edge:

$$p_{j,i} = \frac{\tau_{j,i}^{\alpha} * \eta_{j,i}^{\beta}}{\sum(\tau_{j,i}^{\alpha} * \eta_{j,i}^{\beta})}, \quad \forall(j,i) \in N(s_p) \tag{1}$$

where τ is edge's pheromone, η is edge's heuristic information, $N(s_p)$ is the set of all feasible edges to allowed add to the partial solution s_p, α is a relative pheromone importance, and β is a relative heuristic information importance, j and i are the edges and nodes of the search space respectively. The search mode is chosen every time an edge is added to the partial solution using proportional choice:

$$(j,i) = \begin{cases} argmax(p_{j,i}), & if\ q \leq q0 \\ draw(p_{j,i}), & if\ q > q0 \end{cases} \tag{2}$$

where q is uniformly distributed random number $0 \leq q < 1$, $q0$ is the exploitation parameter, $argmax(p_{j,i})$ is the exploitation mode function which gives the edge with the highest probability, and $draw(p_{j,i})$ is the exploration mode function that draws the edge according to its calculated probability in formula (1). Once an ant search is finished, the solution gets evaluated for solution fitness value, and the best solution is passed to influence the global pheromone. At global pheromone update, the pheromone is evaporated using percentage indicated by ρ parameter as in the following equation:

$$\tau_{j,i} := \tau_{j,i} * (1 - \rho), \quad \forall(j,i) \tag{3}$$

where ρ is a constant parameter of the pheromone evaporation rate introduced by Dorigo and Stützle [24]. The best ant solution is taken to lay down pheromone on edges that it has visited while building the solution as in the following equation:

$$\tau_{j,i} := \tau_{j,i} + \rho * \Delta\tau_0, \quad \forall(j,i) \in s_p \tag{4}$$

where $\Delta\tau_0$ is the pheromone update rate, s_p is the solution of the chosen ant to lay down the pheromone.

The basis of serial algorithm implementation is courtesy of M. Veluscek et al. [25]. However, to utilize modern computer multicore architectures efficiently, parallel ant optimization architecture has been implemented. The parallel ant optimization architecture used in this paper courtesy of I. Dzalbs et al. [26]

This Ant Colony Optimization algorithm is well suited for constrained optimization problems [2]. Heuristic information gives ants a sense of direction when pheromone trails are not intense, and all edges appear similarly strong in the search space. It plays a crucial part in optimization convergence speed.

Some optimization problems may not have reliable static heuristic information that can be precalculated before the search. These optimization problems are usually resource-constrained, and fitness relies on a collection of edges rather than individual edges. For such an optimization problem, the state of a partial solution becomes a vital factor when choosing which edges to add to that partial solution. This research proposes the Dynamic Impact evaluation method as an extension to the Ant Colony Optimization algorithm core to improve solution quality and convergence speed.

2.2 Dynamic Impact for Ant Colony Optimization algorithm

The goal of Dynamic Impact is to enable rapid search identification of the good collection of edges for the solution. Dynamic Impact evaluation is a novel method of calculating each edge's contribution to the fitness value and evaluating the potential consumption of the remaining problem resources before including the edge to the partial solution. This method allows ants to choose edges more accurately that benefit the search's fitness value of the solution the most and uses the least fraction of remaining resources. This method is a third component in an edge's probability calculation, along with pheromone and heuristic information. The Dynamic Impact method is also a myopic search component, and it provides search accuracy improvement similar to the heuristic information approach.

Edge's probability calculation using Dynamic Impact:

$$p_{j,i} = \frac{\tau_{j,i}^{\alpha} * \eta_{j,i}^{\beta} * DI_{j,i}^{\gamma}(s_p)}{\sum\left(\tau_{j,i}^{\alpha} * \eta_{j,i}^{\beta} * DI_{j,i}^{\gamma}(s_p)\right)}, \quad \forall(j,i) \in N(s_p) \tag{5}$$

where $DI_{j,i}^{\gamma}(s_p)$ is Dynamic Impact component in probability calculation at the partial solution state s_p, γ (gamma) is a relative importance of Dynamic Impact, j and i are the edges and nodes of the search space, respectively.

The proposed Dynamic Impact component evaluation is unlike static heuristic information and pheromone. This component depends on the current state of a partial solution and is not pre-calculated like heuristic information. It is designed to change every time an edge is added to a solution. Therefore, it cannot be updated after each solution is completed, like the pheromone.

The best formula for Dynamic Impact calculation depends on the optimization problem and optimization goals. A fitness function or a simplified version of a fitness function is used to calculate Dynamic Impact. In the cases where the fitness function is a non-linear relationship of the combination of edges, the Dynamic Impact measures how much each edge impacts the fitness value for a partial solution. Also, it measures the consumption of remaining resources defined as problem constraints in relation to a reward received from using this edge. The general formula of Dynamic Impact can be expressed as follows:

$$DI_e = \left(f(s_p + e) - f(s_p)\right)^A \times \left(\frac{\Omega(s_p + e)}{\Omega(s_p)}\right) \tag{6}$$

where DI_e is Dynamic Impact for e edge. A is a sign constant of optimization goal: $+1$ for maximization and -1 for minimization objectives. $f(s_p)$ and $f(s_p + e)$ note the fitness values of a partial solution without and with an added edge, respectively. Similarly, $\Omega(s_p)$ and $\Omega(s_p + e)$ are notations of remaining constraints of the partial solution without and with an added edge, respectively. In this theoretical Dynamic Impact calculation, the value is a difference in fitness value multiplied by the proportion of the remaining resources with the edge. For example, Dynamic Impact is a perceived value in a given state for the maximization objective where the highest increase of fitness may not be the most beneficial if it takes a disproportionally large part of the remaining constraints. Some parts of this Dynamic Impact function may be simplified

depending on an optimization problem. For example, in cases where fitness is a linear sum of its solution components $f(s_p + e) - f(s_p)$ can be simplified to just individual fitness of an edge: $f(e)$. Also, the equation's constraints part could be simplified too, depending on if it has non-linear nature or omitted if constraints have no relevance to the solution. Lastly, the Dynamic Impact formula must always be formulated such that it is always more than zero $i.e. DI_e > 0$.

The concept of Dynamic Impact is similar to the dynamic heuristic information described in [6] [27] research work. The Dynamic Impact and dynamic heuristic information both depend on the state of the partial solution. However, the Dynamic Impact is an additional component in the probability calculation and can be used along with static heuristic information if optimization problems can benefit. The Dynamic Impact is a broader operator in edges probability calculation that captures remaining resources consumption and exploits the non-linearity of the fitness function. Furthermore, the ACO algorithm that supports static heuristic information and Dynamic Impact at the same time is more useful in the general setting to optimize combinatorial optimization problems.

In summary, Dynamic Impact evaluation, similarly to static heuristic information, is a myopic search component. However, it is evaluated as each edge is added to a partial solution, making it more versatile in optimization problems where static heuristic information values cannot be calculated in advance or have a non-linear fitness function.

2.3 Dynamic Impact example

Let us consider a simplistic example of vehicle routing where the objective is to minimize the total time spent on a road for each vehicle, but the constraint is fuel in a tank. For simplicity, we will assume that using a motorway is faster but use the most fuel. While using an alternative route would be slower but use less fuel. In such an example, using a motorway, the vehicle might reach the destination faster while using

Table 1: Simplistic example of Dynamic Impact. 3 parallel scenarios that have 3 equivalent routes each. Dynamic Impact is calculated for each route in each scenario individually.

Scenario	Route number	Route distance	Average route speed	Route time	Fuel consumption	Remaining fuel	Dynamic Impact
1	**1**	**25**	**10**	**2.5**	**15**	60	**0.3**
	2	30	15	2	25		0.291667
	3	60	60	1	60		0
2	1	25	10	2.5	15	80	0.325
	2	**30**	**15**	**2**	**25**		**0.34375**
	3	60	60	1	60		0.25
3	1	25	10	2.5	15	120	0.35
	2	30	15	2	25		0.395833
	3	**60**	**60**	**1**	**60**		**0.5**

more fuel than the more direct route in city traffic that is also much slower. Referring to formula (6), this example is a minimization problem therefore $A := -1$. The fitness impact of the edge for a linear fitness function is the linear fitness of the edge $f(s_p + e) - f(s_p) = f(e) := Time(Route)$ which is the time taken for a route. The constraint of the problem is the remaining fuel $\Omega(s_p) := RemainingFuel$, and each edge uses the constraint by consuming the fuel $\Omega(s_p + e) := RemainingFuel - FuelCons(Route)$. Adding all components together, the final formula of the Dynamic Impact example arithmetically simplifies to maximize the inverse time of the route while using the least portion of the remaining fuel.

$$DI_{Route} = \big(Time(Route)\big)^{-1} \times \frac{RemainingFuel - FuelCons(Route)}{RemainingFuel} \quad (7)$$

$$DI_{Route} = \frac{RemainingFuel - FuelCons(Route)}{RemainingFuel * Time(Route)} \quad (8)$$

In Table 1, this formula has been used to demonstrate the difference in Dynamic Impact, considering the only remaining fuel variable. There are three routes (edges) to be considered in this table: first fuel-efficient but slow, second medium-fast and medium fuel-efficient, and third fast with high fuel consumption. Three scenarios of remaining fuel are considered: low, medium, and high amount of remaining fuel. In scenario number one, route number one has the highest Dynamic Impact because a slower but fuel-efficient route is considered to be more attractive in a low fuel scenario. In the second scenario, with a medium amount of fuel, an average fast route is the most attractive. And lastly, in the third scenario, where there is a lot of fuel left to use, the Dynamic Impact strongly suggests the fastest route. The remaining fuel level would not typically be considered in the standard ACO probability calculation, and it would take many iterations for the ants to learn the best complete travel path without having a myopic understanding of which of the routes are in their best interest considering the partial solution an ant has already built. Using Dynamic Impact, ACO can build better initial solutions and let pheromone continue the fine-tuning towards optimal solution along with situation awareness provided by Dynamic Impact. Pheromone and heuristic information do not capture fuel information while building a solution. The pheromone is updated after each iteration using a fully built solution, and static heuristic information is precalculated before the optimization begins.

3. Microchip manufacturing plant production floor optimization (MMPPFO) problem detailed description

In this section, the real-world MMPPFO problem is introduced and described. The details include real-world problem purpose, problem terminology, abbreviations, formulas, problem constraints, and optimization objectives. This information is sufficient to recreate and thoroughly verify the experimental work's findings when using the same dataset.

The optimization problem starts with the initial wafer-lot production schedule and new die request. The wafer-lots schedule has to be altered to support all the changed

and existing planned demands to solve the problem. Schedules can be altered by changing the individual wafer-lot schedule in three major ways: pull-in, push-out, and offload. Pull-in wafer-lot means to produce the wafer-lot earlier. Push-out means to produce the wafer-lot later. Offload means to produce the wafer-lot in another fab. All wafer-lot schedule alterations must comply with existing constraints, therefore making the problem combinatorial NP-hard. Wafer production is a complex process in a microchip manufacturing plant. Each fab can produce a limited quantity of wafers in a selected time window. For the problem solved in this paper, the time window is one week. With known or predicted future die demand, it is possible to create a wafer-lot production schedule that maximizes fabs' efficiency and supports all the requested demand. Moreover, it is desired to support this new demand while having the lowest number of changes to the schedule possible.

3.1 Problem definition

The following are the definitions of MMPPFO used for this research.

Wafer-lot (WL_i) is a non-divisible collection of silicon wafers of a single product to be manufactured all at once and can support only one request. Wafer-lot is noted as WL_i, where i is the index of the wafer-lot. Each wafer-lot has an original schedule slot that can be altered in the problem optimization. For example, wafer-lot WL_{100} can have its commit week changed from $W = 5$ to $W = 3$, which is a pull-in operation as well as at the same time it can be offloaded from $F = F30$ fab to $F = F20$.

Order is a silicon wafer product demand to be manufactured in a fab at a specified week. Order is noted as O_j, where j is the index of the order. Demand may not be fully satisfied – *undersupported*, or it may have too many wafers scheduled – *oversupported*. For example, order number 5 requests for 55 wafers, $O_5 = 55$. This demand can be supported using multiple wafer-lots.

Equipped capacity is the number of wafers of a specified product group that a fab is able to produce at a given week. Equipped capacity is noted as $C_{P,F,W}$, where P is product group, F is fab, W is commit week at which the capacity is defined. Specified fab capacity must not be violated as it is a physical equipment limitation. For example, $C_{P1,F30,W5} = 400$ is the capacity at fab $F30$ in week $W5$ to make product group $P100$ is 400 wafers. The fab may produce more than one product group and will have their capacity defined individually. Also, the fab capacity is defined for each week, as production capacities can vary weekly.

Supported request is a sum of wafers of all wafer-lots that is scheduled to support the request of O_j order

$$SR(O_j) = \sum_i Q(WL_i), \qquad WL_i \in s_p \tag{9}$$

where $SR(O_j)$ is the supported request of O_j order, $Q(WL_i)$ is wafer quantity of WL_i wafer-lot, and wafer-lot WL_i belongs to a solution where it is used for O_j order.

Undersupported request is a number of wafers lacking to support a given request in full for O_j order.

$$USR(O_j) = D(O_j) - SR(O_j) \Big|^{if\ D(O_j) > SR(O_j)}_{otherwise\ 0} \quad (10)$$

where $USR(O_j)$ is the undersupported request of O_j order, $D(O_j)$ is the demand of the order.

Oversupported request is a number of wafers above the requested demand for O_j order.

$$OSR(O_j) = SR(O_j) - D(O_j) \Big|^{if\ D(O_j) < SR(O_j)}_{otherwise\ 0} \quad (11)$$

where $OSR(O_j)$ is the undersupported request of O_j order.

Capacity utilization is a capacity that has been used for wafer production, calculated from an output schedule of an optimization.

$$U(C_{P,F,W}) = \sum_i Q(WL_i), \qquad WL_i \in s_p \quad (12)$$

where $U(C_{P,F,W})$ is the utilization of specified fab capacity $C_{P,F,W}$, and wafer-lot WL_i belongs to the solution where it is using fab capacity $C_{P,F,W}$.

Capacity waste is a capacity that has been left unused. Capacity waste cannot be negative.

$$WA(C_{P,F,W}) = C_{P,F,W} - U(C_{P,F,W}) \quad (13)$$

where $WA(C_{P,F,W})$ is the waste of specified fab capacity $C_{P,F,W}$.

Problem solution is a schedule of wafer-lots to be manufactured, noted as s_p. The schedule indicates what wafer-lots WL_i are manufactured at given commit week W, and given fab F. A fully assembled solution must comply with all problem constraints.

Problem search space noted as N is a collection of all vertices and all edges of feasible combinatorial permutations.

3.2 Constraints

This optimization problem has a set of constraints that the optimization engine must simultaneously consider when building a solution. Some constraints are the combinatorial constraints, meaning that a combination of wafer-lots must satisfy a given constraint. Other constraints can be the search space constraints that are applied for an individual wafer-lot. Search space constraints limit the total search space to be explored as a consequence.

Capacity constraint

Fabs have equipped capacity that is a hard limit on how many wafers of a specified product group can be scheduled for a given commit week. The sum of wafers must always be lower or equal to equipped capacity. The limit is in effect as a sum of wafers of wafer-lot collection for a given week and fab, thus it is a combinatorial constraint.

$$C_{P,F,W} > U(C_{P,F,W}) \quad (14)$$

Order support constraint

All wafers supporting an order must be committed on time or ahead of time. This way all wafer-lot permutations that are too late are not included as edges of search space, therefore constraining search space.

$$W(WL_i) \leq W(O_j), \quad \forall (j,i) \in N \tag{15}$$

where $W(WL_i)$ is commit week of WL_i, $W(O_j)$ is commit week of O_j order, for all permutations of j, i that belong to search space N.

Pull-in, push-out constraint

Wafer-lot schedule changes must follow specified pull-in (bring forward production) push-out (delay production) information, i.e. not all products can be pulled-in or pushed-out. Pull-in operations for specific products can only be done in fabs that allow such an operation. Push-out can be done only for a corresponding pull-in operation if necessary to stay within capacity constraint. This constraint limits the search space by allowing only limited pull-in or push-out operations out of all possible combinations. Moreover, each push-out must have a corresponding pull-in operation applied in the solution, thus making it a combinatorial constraint.

Offload constraint

Each wafer-lot can be offloaded to fabs that support the product group and product itself. This limits the search space by not including wafer-lot permutations of offload to fabs that cannot produce the wafer-lot product.

3.3 Optimization objective

In microchip manufacturing, efficiency can be expressed in several different ways. Each solution produced by the ACO must be evaluated to get the fitness value. Then solution fitness value is compared to other solutions. A solution with a lower fitness value is a better solution for a minimization objective.

This optimization problem's primary objective is to minimize undersupported requests that ensure that all customer orders get silicon chips fulfilled on time. Minimizing the undersupported request means that all orders should have wafer request supported fully or have the least possible number of wafers undersupported.

$$min \sum_j USR(O_j) \tag{16}$$

where $USR(O_j)$ stands for UnderSupported Request of O_j order.

For new silicon chip demand, it is possible that requested wafers could not be met with an integer number of wafer-lots where wafer-lot has a fixed number of wafers that do not match the demand precisely. In such scenario, the request will be either undersupported and have the orders not fully complete or oversupported and waste the production that could be utilized to support other demand.

4. Experimental work

In this section, two experiments have been conducted. First, solving the real-world Microchip Manufacturing Plant Production Floor Optimization (MMPPFO). And second, solving the theoretical Multidimensional Knapsack Problem (MKP). Both experiments introduce the specific implementations of ACO and tuned parameters.

Most importantly, both experiments provide the formula used for Dynamic Impact evaluation. And finally, the experiment results are analyzed.

4.1 ACO solving Microchip Manufacturing Plant Production Floor Optimization (MMPPFO)

Search space preparation

Ants can only navigate efficiently in the prepared search space where all edges are filtered for feasibility and have pheromone and heuristic information values attached to them. In MMPPFO, a possible wafer-lot allocation for production is an edge of a search space. One wafer-lot can have multiple permutations, including different production weeks and production fabs.

Heuristic information

Ant Colony Optimization uses heuristic information that plays a crucial role in the algorithm's convergence [28]. Heuristic information gives ants a myopic benefit and directs them to explore more promising parts of the search space and obtain good initial solutions before strong pheromone trails are laid. Static heuristic information is calculated during search space preparation and remains constant throughout the entire algorithm run. For this experiment, the preliminary edge's static heuristic information is the following:

$$\eta_{j,i} = \frac{O_j}{Q(WL_i)} \tag{17}$$

where O_j is the number of wafers in the order and $Q(WL_i)$ is wafer quantity of WL_i wafer-lot. However, for the MMPPFO problem, the main objective, a minimum undersupported request, preliminary testing has shown that ACO performed best using $\beta = 0$, in which case the heuristic information had not been used. This shows that individual wafer-lots do not carry any significance over others as only the total collection of wafer-lots is essential.

Experimental dataset

For algorithm validity and performance testing, a synthetic dataset is used to cover various corner cases that could occur in real optimization scenarios. The dataset was generated with an industry partner using a real dataset basis with masked industry secrets but preserved patterns and dynamics. The dataset used for this experiment is published in the figshare repository [29].

Dynamic Impact for MMPPFO optimization

Dynamic Impact's goal for the MMPPFO problem is to quickly identify a good collection of wafer-lots to support the order. The formula of Dynamic Impact for MMPPFO minimum undersupported request objective has been obtained using the general formula (6) as a guide with some adjustments.

The minimum undersupported request objective is a special case of minimization objectives. An empty solution starts with a high fitness value, and each edge added to the partial solution reduces the fitness. Such dynamics are treated as the negative of maximization objective, and therefore fitness impact term of Dynamic Impact formula is adjusted as follows:

$$\left(f(s_p + e) - f(s_p)\right)^A \rightarrow f(s_p) - f(s_p + e) \quad (18)$$

Fitness impact of the edge is defined as non-linear support of the O_j order and WL_i wafer-lot:

$$f(s_p + e) := max\{RD(O_j) - Q(WL_i), 0\} \quad (19)$$

$$f(s_p) := RD(O_j) \quad (20)$$

$$RD(O_j) = D(O_j) - SR(O_j) \quad (21)$$

$$f(s_p) - f(s_p + e) = RD(O_j) - max\{RD(O_j) - Q(WL_i), 0\} \quad (22)$$

where $f(s_p + e)$ is given the remaining demand minus wafer quantity of the WL_i wafer-lot, $f(s_p)$ is given the remaining demand, $RD(O_j)$ is remaining demand for the O_j order, $D(O_j)$ the total demand of the order, and $SR(O_j)$ is supported request of the order. $Q(WL_i)$ is the wafer quantity of WL_i wafer-lot.

The constrained resource of this problem is the fab capacity:

$$\Omega(s_p + e) := RC(C_{P,F,W}) - Q(WL_i) \quad (23)$$

$$\Omega(s_p) := RC(C_{P,F,W}) \quad (24)$$

$$RC(C_{P,F,W}) = C_{P,F,W} - U(C_{P,F,W}) \quad (25)$$

where $\Omega(s_p + e)$ is given the remaining fab capacity minus wafer quantity of the WL_i wafer-lot, $\Omega(s_p)$ is given the remaining fab capacity. $RC(C_{P,F,W})$ is the remaining capacity of the equipped fab capacity $C_{P,F,W}$.

It is important to notice that remaining capacity $RC(C_{P,F,W})$ and capacity waste $WA(C_{P,F,W})$ are equivalent expressions and ideally, wasted capacity should be as low as possible. Also, every wafer-lot produced on time contributes to fitness the same amount as it consumes the fab capacity if the demand is higher than the wafer quantity of the wafer-lot. However, the wafers produced over the demand do consume the fab capacity and do not contribute to the fitness value. Both fitness and constraints calculations are expressed in units of wafer quantity. This similarity of the units can be exploited to increase computational efficiency. For this optimization problem, it is more beneficial to count only the wafers over the remaining demand, which would consume the equipped fab capacity that could potentially be used to produce other wafers and support more demand:

$$\frac{\Omega(s_p + e)}{\Omega(s_p)} := max\{Q(WL_i) - RD(O_j), 0\} \quad (26)$$

This constraint impact part is a count and not a ratio, therefore it should be added to the fitness impact part and not multiplied:

$$\begin{aligned} f(s_p) - f(s_p + e) + \frac{\Omega(s_p + e)}{\Omega(s_p)} &= RD(O_j) \\ &\quad - max\{RD(O_j) - Q(WL_i), 0\} \\ &\quad + max\{Q(WL_i) - RD(O_j), 0\} \\ &= RD(O_j) - \left|RD(O_j) - Q(WL_i)\right| \end{aligned} \quad (27)$$

Finally, the Dynamic Impact must respect $DI_e > 0$ rule, therefore for this optimization problem, it is chosen to constrain the Dynamic Impact value not less than 0.1 such that the algorithm does not calculate zero or negative probability for the edge $DI_{j,i} \geq 0.1$.

Then added all parts together, the final Dynamic Impact formula of minimum undersupported request objective is as follows:

$$DI_{j,i} = max\left\{RD\left(O_j\right) - \left|RD\left(O_j\right) - Q(WL_i)\right|, 0.1\right\} \quad (28)$$

This Dynamic Impact evaluation formula represents a simplified fitness function and considers the wasteful fab capacity utilization. Once the demand is supported, producing more wafers does not benefit fitness while wasting the wafer production resources.

4.2 MMPPFO experiment results

The experiment is designed to test the benefit of using Dynamic Impact for the Min-Max Ant System in order to achieve the best final result. In this experiment, two probability parameters will be tested $q0$ and γ. γ is the main variable that defines the importance of Dynamic Impact. The experiment baseline is $\gamma = 0$ (Dynamic Impact has no contribution to search probabilities). Moreover, in this experiment $q0$ – the ant exploration hyperparameter is tested, as the optimal value of $q0$ often depends on the other hyperparameters. γ and $q0$ are tested with a wide range of values to determine the best possible combination of γ and $q0$, as well as to assert the baseline of the experiment with $\gamma = 0$ parameter. In this experiment, the range of γ is from 0.125 growing exponentially to 16 by a factor of 2, and $q0$ is from 0 increasing linearly to 0.95 by an increment of 0.05.

The remaining parameters of Min-Max Ant System have been established by preliminary experimentation. The best combination of pheromone parameters are: $\tau_{max} = 1$, $\tau_{min} = 0.001$, $\rho = 0.1$, $\Delta\tau_0 = 1$. Configuration of probability parameters: $\alpha = 1$, $\beta = 0$. Solutions are achieved running 3,000 iterations using 2 sequential ants, using 16 parallel ants as per Dzlabs et al. described architectural model [26].

In Table 2 the undersupported score is displayed for each of $q0$ and gamma configuration combinations. Each data is an average score of 50 independent algorithm runs. Firstly, the asserted baseline of $\gamma = 0$, which means Dynamic Impact does not influence the search probability calculation. The best configuration of γ is $\gamma = 0$, $q0 = 0.3$, the corresponding result at this configuration is 31.0 wafers of undersupported request. For the runs using Dynamic Impact, the best results are obtained with configuration $\gamma = 4$, $q0 = 0$, and the result is 19.0 average wafers of undersupported

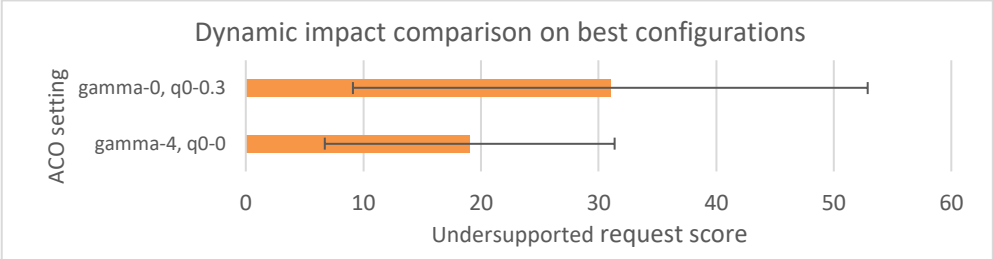

Fig 1. Dynamic Impact comparison on best configurations. Average of 50 runs. Error Bars indicate one standard deviation.

request score. With higher Dynamic Impact importance, ACO performs the best when using lower $q0$ values, higher exploration. Using Dynamic Impact with $\gamma = 4$, consistently outperforms $\gamma = 0$ across wider range of $q0$ values. In the real-world deployment scenarios where the algorithm's $q0$ value is not tuned perfectly, but only roughly estimated $q0$ value. Using the average of 5 best $q0$ settings, at $\gamma = 4$ is 22.1 wafers of undersupported request. In comparison, for imperfectly tuned baseline, the average of 5 best $q0$ settings at $\gamma = 0$ is 35.3 wafers of undersupported request.

Moreover, in Fig 1, a more detailed comparison of best configurations among baseline $\gamma = 0$, $q0 = 0.3$ and best configuration using Dynamic Impact evaluation $\gamma = 4$, $q0 = 0$. In the Fig 1, the main bar represents the average undersupported score of 50 algorithm runs of the shown setting. The error bars indicate one standard deviation of the scores across the runs.

On this optimization problem, with iterations limited to 3,000, using Dynamic Impact, the undersupported request score has been improved on average by 38.5%. Moreover, using Dynamic Impact, the standard deviation is reduced from 20.8 to 12.3. This smaller standard deviation means lower quality solutions occur significantly less

Table 2. Undersupported result map for γ and q0, where $\gamma = 0$ is an algorithm run without Dynamic Impact. Each data point represents the average of 50 runs. Results of optimizing the heuristically generated dataset.

		Gamma γ								
		0	0.125	0.25	0.5	1	2	4	8	16
q0	0	56.6	50.2	52.3	45.9	33.9	23.3	**19.0**	36.1	70.9
	0.05	57.5	49.6	45.4	41.2	31.9	22.9	29.1	32.3	68.4
	0.1	48.0	47.1	42.0	36.9	31.9	20.2	21.5	43.5	70.1
	0.15	41.5	38.7	33.5	35.2	30.6	21.8	19.5	36.9	69.7
	0.2	42.4	38.8	35.2	33.9	29.6	23.4	21.5	36.4	72.7
	0.25	38.1	33.0	33.2	31.7	31.0	27.9	35.8	44.0	73.9
	0.3	**31.0**	32.0	40.0	29.9	27.1	35.2	33.2	49.2	95.2
	0.35	37.9	32.1	38.2	36.6	36.8	31.2	32.0	54.6	99.5
	0.4	37.3	46.5	45.0	54.0	48.3	43.0	44.0	58.1	110.5
	0.45	32.5	51.5	51.3	55.0	57.2	65.9	65.6	77.3	151.5
	0.5	49.1	53.6	63.1	64.9	71.6	73.5	87.0	101.0	171.9
	0.55	49.1	69.0	77.4	102.2	105.2	93.2	104.9	107.3	163.0
	0.6	53.4	84.6	92.7	98.4	108.6	120.1	104.9	133.1	184.3
	0.65	81.3	113.5	120.3	127.1	146.2	147.5	141.5	150.1	194.6
	0.7	105.8	134.3	162.4	157.9	161.4	163.6	164.3	175.2	218.3
	0.75	129.9	161.7	178.2	192.8	187.9	192.2	204.2	223.6	226.3
	0.8	177.6	191.3	209.3	207.6	214.9	222.4	232.0	236.2	249.4
	0.85	207.5	221.4	225.8	233.7	254.5	265.9	255.0	274.3	294.0
	0.9	275.9	286.0	311.4	319.7	324.8	323.4	330.7	334.1	392.3
	0.95	375.8	362.5	401.3	387.1	409.2	440.1	442.7	489.9	464.5

often, making the performance more reliable in fast-paced environments or solving large-scale optimization problems where a good solution is needed as soon as possible.

Dynamic Impact comes with a small computational performance cost since Dynamic Impact needs to be calculated for each wafer-lot probability calculation. ACO at best configuration without Dynamic Impact runs on average 86.8 seconds. Using the best configuration algorithm, ACO with Dynamic Impact took on average 96.9 seconds, Dynamic Impact evaluation added, on average 11.6% to the computational overhead to compute the iteration. Such a small overhead was possible due to the simplified wafer-lot impact on solution fitness value, making the evaluation not a computationally expensive operation.

In conclusion, the Dynamic Impact method has proven to be highly beneficial for an objective where the aim is to have a combination of elements adding up to the specific requested size or number. This experiment has demonstrated that real-world problems can be solved using an Ant Colony optimization algorithm within acceptable computational limits.

4.3 ACO solving Multidimensional Knapsack Problem (MKP)

In addition to solving MMPPFO, we have implemented the ACO to solve MKP. The purpose of solving MKP is to test the Dynamic Impact evaluation method on a benchmark optimization problem and compare it against peer research results.

Search space preparation

The search space of the MKP is simple. The search space is expressed in a single dimension of binary option, to take an item in the knapsack or not. Pheromone τ_i, heuristic information η_i, and Dynamic Impact DI_i are, in this case, also single-dimensional. Each item's probability calculation is done all at once before adding any item into the partial solution.

Heuristic information

Similarly to MMPPFO, MKP's maximum profit objective depends on the total profit of the collection of all items taken in the knapsack. For this experiment, the preliminary edge's static heuristic information is the following:

$$\eta_i = \frac{P(I_i)}{W(I_i)} \tag{29}$$

where $W(I_i)$ is weight, and $P(I_i)$ is the profit of the item defined in the input dataset. The preliminary testing has shown that ACO performed best using $\beta = 0$, in which case the heuristic information had not been used. This shows that none of the items are more important in the knapsack than the others. Only a combination of the items that all simultaneously fit in all knapsack dimensions must have the highest profit possible.

Experimental dataset

MKP optimization has been chosen in part due to the large availability of benchmark datasets as well as available research publishing the results that state-of-the-art optimization algorithms have achieved. The datasets are obtained from the ResearchGate repository [30]. From this repository, small SAC94 datasets and large GK datasets will be solved. For small SAC94 datasets, the focus is on achieving optimal

values with the highest possible success rate. On larger GK datasets, the goal is to get the highest profit on average.

Dynamic Impact for MKP optimization

Dynamic Impact evaluation equation to solve MKP is different from the MMPPFO problem as problem domains are not the same. For this problem, the Dynamic Impact formula is the following:

$$DI_i = \frac{NP(I_i)}{CI(I_i)} \tag{30}$$

$$CI(I_i) = max_{\forall j}\left\{\frac{W(I_i)}{RC(K_j)}\right\} + \frac{\sum_j \left\{\frac{W(I_i)}{RC(K_j)}\right\}}{j} \tag{31}$$

$$NP(I_i) = \frac{P(I_i)}{max\{\forall P(I)\}} \tag{32}$$

where DI_i – is Dynamic Impact for item I_i, calculated using normalized item profit over the capacity impact of the item. Normalized profit $NP(I_i)$ of the item I_i is a constant parameter precalculated using the profit of the item and the highest profit of all items. It is essential to have normalized profit from 0 to 1 in Dynamic Impact such that probability calculations have a constant range of inputs for any item profit magnitude range across various input datasets. $CI(I_i)$ is a capacity impact of the item I_i. This is the most intense compute operation of the Dynamic Impact evaluation. It finds the maximum weight utilization combined with average weight utilization of remaining knapsack capacities. The capacity impact has to be recalculated whenever doing the probability calculations as it uses the remaining knapsack capacities $RC(K_j)$ in contrast to the total capacity that does not change while building the solution. When using remaining the knapsack capacity, the current state of the solution is well reflected and can impact the probability calculation to pick an item that does consume a lower portion of available knapsack space for the same profit reward. $W(I_i)$ is weight, and $P(I_i)$ is the profit of the item defined in the input dataset.

Dynamic Impact formula for MKP is obtained in relation to the general Dynamic Impact formula (6). The formula (6) component values are: MKP is a maximization problem $A := 1$. The fitness impact of the edge is the linear normalized profit value of the item I_i $f(s_p + e) - f(s_p) = f(e) := NP(I_i)$. The resource of the MKP problem is capacity therefore $\Omega(s_p) := \sum_j \{RC(K_j)\}$ which is the remaining capacity of all knapsacks, and $\Omega(s_p + e) := \sum_j \{RC(K_j) - W(I_i)\}$ which is the remaining capacity of all knapsacks minus the weight of the item. However, for this problem, to optimize Dynamic Impact for computational efficiency, the constraints are adapted to use only one items weight over the remaining capacity like in the following formula:

$$DI_i = \frac{f(e)}{\left(\frac{\Omega(e)}{\Omega(s_p)}\right)} \tag{33}$$

It is slightly computationally cheaper to compute only items weight and get the same overall result $\Omega(e) := \sum_j \{W(I_i)\}/j$. Additionally, items of the MKP use multiple knapsacks in disproportional quantities, and it is important to track not only the average resource consumption impact but also the maximum impact on a single knapsack too

$\Omega(e) := max_{\forall j}\{W(I_i)\}$. For this reason, the capacity impact formula has two components of resource consumption that complement each other.

4.4 MKP experiment results

This MKP experiment is chosen, in addition to solving the MMPPFO problem, to solve a commonly available benchmark problem that has similar multiple item collection characteristics. There are no recent papers published on Ant Colony Algorithm solving MKP benchmark datasets. Therefore, it is logical to assume that there have not been any successful attempts to achieve results on public benchmark datasets to a comparable level to other published works.

Two sets of benchmarks MKP datasets are considered in this experiment. The first set SAC94 are small datasets and are easy enough to find the optimal solutions of those datasets within a reasonable amount of time. For these small datasets, the algorithm success rate is analyzed and compared to which algorithm on average reaches optimal solution quicker. The second set is large GK benchmark datasets. These benchmark datasets' combinatorial complexity is high enough such that not all of GK datasets have known optimal value. Therefore, in Table 5 for comparison, the most recent best-known values will be taken from [31] that combines their own reached highest values as well as [32] and authors of the GK datasets [33]. For large GK datasets, the aim is to get the highest possible profit or, in other words, to minimize a profit gap to the best-known solution.

SAC94 results

For the SAC94 experiment, Min-Max Ant System parameters have been tuned with preliminary experimentation. The best combination of pheromone parameters are: $\tau_{max} = 1$, $\tau_{min} = 0.001$, $\rho = 0.1$. Configuration of probability parameters: $\alpha = 1$, $\beta = 0$, $q0 = 0.01$. Solutions are achieved running 3,000 iterations using two sequential ants, using 64 parallel ants as per [26] described architectural model. Experiment measures success rate, best successful iteration, average successful iteration, and an average profit of each dataset using Dynamic Impact versus algorithm without Dynamic Impact implemented. Each data point is an average of 100 algorithm runs. In Table 3 SAC94 dataset results are presented. Ant Colony Optimization using Dynamic Impact preliminary tests showed that the best convergence is achieved using Gamma (γ) value set to 8. ACO with Dynamic Impact shows a 100% success rate in every single dataset while the same algorithm without Dynamic Impact manages to do so in 41 out of 54 datasets and the remaining datasets average a 74.7% success rate. Moreover, optimization with Dynamic Impact on average takes just 12.40 iterations and 0.046 seconds to reach optimal value. On average, without Dynamic Impact, it takes 128.96 iterations and 0.25 seconds to reach optimal on 41 datasets that managed successfully converge 100% of the time.

Table 3: MKP SAC94 datasets. Dynamic Impact result comparison of ACO without Dynamic Impact and ACO with Dynamic Impact. Each dataset is a result of 100 runs.

			ACO without Dynamic Impact					ACO with Dynamic Impact				
Dataset	Problem size (N x M)	Optimal	Success rate	Best successful iteration	Average successful iteration	Average time to success (seconds)	Average profit	Success rate	Best successful iteration	Average successful iteration	Average time to success (seconds)	Average profit
hp1	28 x 4	3418	0.97	3	n/a	n/a	3417.58	1	0	0.75	0.00308	3418
hp2	35 x 4	3186	0.95	7	n/a	n/a	3185.1	1	5	36.65	0.04048	3186
pb1	27 x 4	3090	1	4	334.51	0.25203	3090	1	0	0.59	0.00303	3090
pb2	34 x 4	3186	0.97	10	n/a	n/a	3185.46	1	0	33.87	0.03768	3186
pb4	29 x 2	95168	1	6	17.97	0.01701	95168	1	0	0.71	0.00285	95168
pb5	20 x 10	2139	1	0	40.53	0.02307	2139	1	0	26.5	0.01661	2139
pb6	40 x 30	776	1	4	18.68	0.01815	776	1	0	0.14	0.00242	776
pb7	37 x 30	1035	0.94	10	n/a	n/a	1034.47	1	0	4.6	0.00853	1035
pet2	10 x 10	87061	1	0	0.08	0.00169	87061	1	0	8.44	0.00514	87061
pet3	15 x 10	4015	1	0	4.02	0.00453	4015	1	0	0	0.00179	4015
pet4	20 x 10	6120	1	0	10.81	0.00924	6120	1	0	0	0.00211	6120
pet5	28 x 10	12400	1	7	13.92	0.0177	12400	1	0	0	0.00195	12400
pet6	39 x 5	10618	0.44	32	n/a	n/a	10610.16	1	0	10.61	0.01599	10618
pet7	50 x 5	16537	1	36	249.55	0.41771	16537	1	12	67.62	0.12189	16537
sento1	60 x 30	7772	1	39	319.23	0.59452	7772	1	0	0.11	0.00396	7772
sento2	60 x 30	8722	0.65	53	n/a	n/a	8718.54	1	0	1.94	0.01163	8722
weing1	28 x 2	141278	1	13	32.6	0.03052	141278	1	0	0	0.00155	141278
weing2	28 x 2	130883	1	14	36.05	0.02862	130883	1	0	0	0.00163	130883
weing3	28 x 2	95677	1	6	29.44	0.01889	95677	1	0	0	0.00154	95677
weing4	28 x 2	119337	1	7	21.87	0.01853	119337	1	0	0	0.00193	119337
weing5	28 x 2	98796	1	4	18.06	0.01286	98796	1	0	0	0.00164	98796
weing6	28 x 2	130623	1	11	43.77	0.03164	130623	1	0	0	0.00165	130623
weing7	105 x 2	1095445	0	n/a	n/a	n/a	1095136	1	4	456.14	2.06904	1095445
weing8	105 x 2	624319	0.03	1981	n/a	n/a	620481.5	1	0	0.7	0.006	624319
weish01	30 x 5	4554	1	12	27.83	0.02154	4554	1	0	0	0.00212	4554
weish02	30 x 5	4536	0.91	7	n/a	n/a	4535.55	1	0	0	0.0024	4536
weish03	30 x 5	4115	1	3	21.84	0.01619	4115	1	0	0	0.00211	4115
weish04	30 x 5	4561	1	1	12.33	0.0094	4561	1	0	0	0.0022	4561
weish05	30 x 5	4514	1	2	10.61	0.00862	4514	1	0	0	0.002	4514
weish06	40 x 5	5557	1	19	189.83	0.18289	5557	1	0	0.08	0.00251	5557
weish07	40 x 5	5567	1	14	35.38	0.03701	5567	1	0	0	0.00247	5567
weish08	40 x 5	5605	1	15	37.97	0.04175	5605	1	0	0	0.00254	5605
weish09	40 x 5	5246	1	18	31.22	0.02959	5246	1	0	0	0.00248	5246
weish10	50 x 5	6339	1	28	65.49	0.08092	6339	1	0	12.08	0.01763	6339
weish11	50 x 5	5643	1	18	62.45	0.06658	5643	1	0	0	0.00248	5643
weish12	50 x 5	6339	1	20	56.96	0.06909	6339	1	0	7.5	0.01246	6339
weish13	50 x 5	6159	1	18	35.51	0.04445	6159	1	0	0	0.00263	6159
weish14	60 x 5	6954	1	27	44.24	0.06997	6954	1	0	0	0.00267	6954
weish15	60 x 5	7486	1	35	74.64	0.11307	7486	1	0	0	0.00325	7486
weish16	60 x 5	7289	1	39	545.29	0.85691	7289	1	0	0.01	0.00308	7289
weish17	60 x 5	8633	1	30	78.55	0.1655	8633	1	0	0	0.00374	8633
weish18	70 x 5	9580	1	52	265.71	0.614	9580	1	0	0.52	0.00531	9580
weish19	70 x 5	7698	0.93	40	n/a	n/a	7697.09	1	0	0	0.00346	7698
weish20	70 x 5	9450	1	61	398.67	0.85951	9450	1	0	0	0.00387	9450
weish21	70 x 5	9074	1	44	246.19	0.50368	9074	1	0	0.02	0.00369	9074
weish22	80 x 5	8947	0.56	54	n/a	n/a	8939.08	1	0	0	0.00391	8947
weish23	80 x 5	8344	1	44	109.6	0.24405	8344	1	0	0.05	0.00383	8344
weish24	80 x 5	10220	1	74	476.98	1.34094	10220	1	0	0	0.00444	10220
weish25	80 x 5	9939	0.94	71	n/a	n/a	9938.17	1	0	0	0.00403	9939
weish26	90 x 5	9584	0.48	71	n/a	n/a	9567.44	1	0	0	0.00449	9584
weish27	90 x 5	9819	1	62	135.06	0.38311	9819	1	0	0	0.00448	9819
weish28	90 x 5	9492	1	65	421.75	1.14258	9492	1	0	0	0.00442	9492
weish29	90 x 5	9410	1	73	386.61	1.03829	9410	1	0	0	0.00436	9410
weish30	90 x 5	11191	1	64	325.52	1.1109	11191	1	0	0.01	0.00503	11191

Table 4: SAC94 results comparison with recently published research.

Dataset	Problem size (N x M)	Optimal	ACO without Dynamic Impact	ACO with Dynamic Impact	BPSOTVAC - [34] 2014	DBDE - [35] 2017	MFPA - [36] 2018	HPSOGO - [37] 2018	TR-BDS - [38] 2016	BAAA - [39] 2016
hp1	28 x 4	3418	0.97	1	0.38		1		0.4	0.93
hp2	35 x 4	3186	0.95	1	0.67				0.97	0.27
pb1	27 x 4	3090	1	1	0.46		1		0.5	1
pb2	34 x 4	3186	0.97	1	0.73				0.97	1
pb4	29 x 2	95168	1	1	0.91				1	1
pb5	20 x 10	2139	1	1	0.84		1		0.8	1
pb6	40 x 30	776	1	1	0.5		1		0.57	1
pb7	37 x 30	1035	0.94	1	0.47		1		0.8	1
pet2	10 x 10	87061	1	1			1			
pet3	15 x 10	4015	1	1						
pet4	20 x 10	6120	1	1						
pet5	28 x 10	12400	1	1						
pet6	39 x 5	10618	0.44	1						
pet7	50 x 5	16537	1	1						
sento1	60 x 30	7772	1	1	0.57	0.43	1	0.16	0.8	1
sento2	60 x 30	8722	0.65	1	0.27	0	1	0.25	0.73	1
weing1	28 x 2	141278	1	1	1	1		0.1	1	1
weing2	28 x 2	130883	1	1	1	0.97		1	1	1
weing3	28 x 2	95677	1	1	0.92	0.6	1	1	0	1
weing4	28 x 2	119337	1	1	1	1	1	1	1	1
weing5	28 x 2	98796	1	1	1	0.3		1	0.7	1
weing6	28 x 2	130623	1	1	0.97	0.97	1	1	1	1
weing7	105 x 2	1E+06	0	1	0	0		1	0	0.58
weing8	105 x 2	624319	0.03	1	0.35	0		1	0.5	0.93
weish01	30 x 5	4554	1	1	1	1	1	1	1	1
weish02	30 x 5	4536	0.91	1	0.64	1	1	1	1	1
weish03	30 x 5	4115	1	1	0.99	1	1	1	1	1
weish04	30 x 5	4561	1	1	1	1	1	1	1	1
weish05	30 x 5	4514	1	1	1	1	1	1	1	1
weish06	40 x 5	5557	1	1	0.59	0.3	1	1	1	1
weish07	40 x 5	5567	1	1	0.96	0.33	1	1	0.98	1
weish08	40 x 5	5605	1	1	0.79	0.87	1	1	0.98	1
weish09	40 x 5	5246	1	1	1	1	1	1	1	1
weish10	50 x 5	6339	1	1	0.91	1	1	1	1	1
weish11	50 x 5	5643	1	1	0.88	0.63	1	1	0.92	1
weish12	50 x 5	6339	1	1	0.89	1	0.82	1	0.96	1
weish13	50 x 5	6159	1	1	1	1	1	0.35	0.98	1
weish14	60 x 5	6954	1	1	0.98	1	1	1	0.92	1
weish15	60 x 5	7486	1	1	1	1	1	1	0.96	1
weish16	60 x 5	7289	1	1	0.54	0.87	1	1	1	1
weish17	60 x 5	8633	1	1	1	0.67		1	1	1
weish18	70 x 5	9580	1	1	0.75	1		1	0.98	1
weish19	70 x 5	7698	0.93	1	0.65	1	1	1	0.96	1
weish20	70 x 5	9450	1	1	0.78	1	1	1	0.96	1
weish21	70 x 5	9074	1	1	0.74	1	1	0.1	0.96	1
weish22	80 x 5	8947	0.56	1	0.16	1		1	0.98	1
weish23	80 x 5	8344	1	1	0.85	0.23		1	0.92	0.45
weish24	80 x 5	10220	1	1	0.7	1		1	0.68	0.54
weish25	80 x 5	9939	0.94	1	0.49	0.97		1	0.84	1
weish26	90 x 5	9584	0.48	1	0.36	1	1	1	0.94	1
weish27	90 x 5	9819	1	1	0.99	0.97		1	0.98	1
weish28	90 x 5	9492	1	1	0.87	1		1	0.94	1
weish29	90 x 5	9410	1	1	0.86	1		1	0.92	1
weish30	90 x 5	11191	1	1	0.87	0.83		1	0.32	1

Our achieved results of SAC94 are compared to recently published research on the state-of-the-art optimization algorithms solving SAC94 datasets in Table 4. A Binary PSO with Time-Varying Acceleration Coefficients (BPSOTVAC) proposed by Chih et al. [34]. A Dichotomous Binary Differential Evolution (DBDE) proposed by Peng et al. [35]. A Modified version of the Flower Pollination Algorithm (MFPA) proposed by Abdel-Basset et al. [36]. A Binary Particle Swarm Optimization with Genetic Operations (HPSOGO) introduced by Mingo López et al. [37]. A Random Binary Differential Search algorithm using the Tanh function (TR-BDS) introduced by Liu et al. [38]. A Binary Artificial Algae Algorithm (BAAA) introduced by Zhang et al. [39].

The primary comparison metric of all results is the success rate. Proposed ACO with Dynamic Impact shows superiority in solving small datasets. None of the reviewed algorithms have such versatility in solving all of the datasets reliably to the optimal value 100% of the time. The closest algorithm MFPA solves, on average, 99.42% successfully on the datasets published. However, it is important to note that this research paper [36] is inconclusive and does not have complete SAC94 dataset results hence the versatility of the algorithm is not proven since the success rate is unknown for the remaining datasets. Secondly, BAAA has 95.2% on the average success rate of 48 datasets. 42 out of 48 datasets have reached a 100% success rate. None of the authors has considered pet2-pet7 datasets part of SAC94. “pet” datasets seem to be an edge

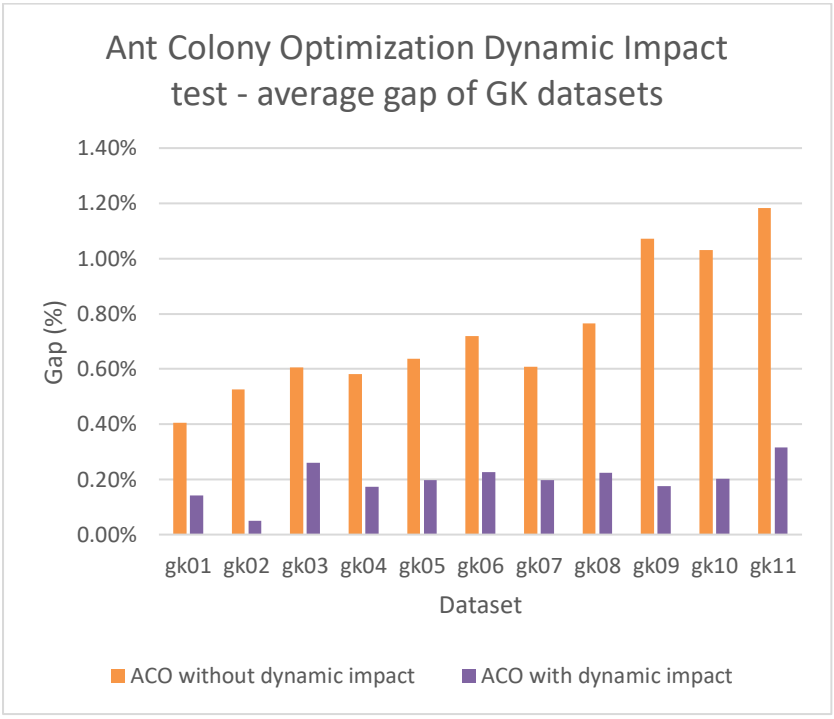

Fig 2. ACO Dynamic Impact test - GK dataset results graph of average gap.

case, especially problematic for any optimization algorithm with observed highly sparse nature. Despite small theoretical combinatorial complexity, and are challenging to solve. None of the other research has published results solving "pet" datasets, possibly due to difficulty handling a high degree of sparseness, especially when it is expected to be easily solved as theoretical combinatorial complexity is low.

GK results

The algorithm has been tuned slightly differently to solve large GK datasets. Dynamic Impact importance parameter Gamma (γ) value is set to 32, and the algorithm is run for 10000 iterations. Experiment measures average profit obtained over 10 algorithm runs. Then the average profit is turned into the average gap using best-known profit values. In Fig 2 ACO with Dynamic Impact is compared to the same algorithm without implemented Dynamic Impact running the same probability settings. In absolute terms, ACO with Dynamic Impact gets an average gap reduction of 0.54%, where the highest difference is in gk09 – 0.9% and lowest in gk01 – 0.27%. In relative terms, the difference in the profit gap is average 4.27 times lower, where the highest is gk02 reducing the gap 10.4 times, and the lowest in gk03 reducing the gap by 2.33 times.

Furthermore, in Fig 3 well-performing ACO with Dynamic Impact algorithm is stacked up against recently published solutions of GK dataset implementations. Dantas – GPGPU SA [40] is GPU accelerated Simulated Annealing algorithm. Kong – NBHS2 [41] out of several algorithms compared their proposed New Binary Harmony Search

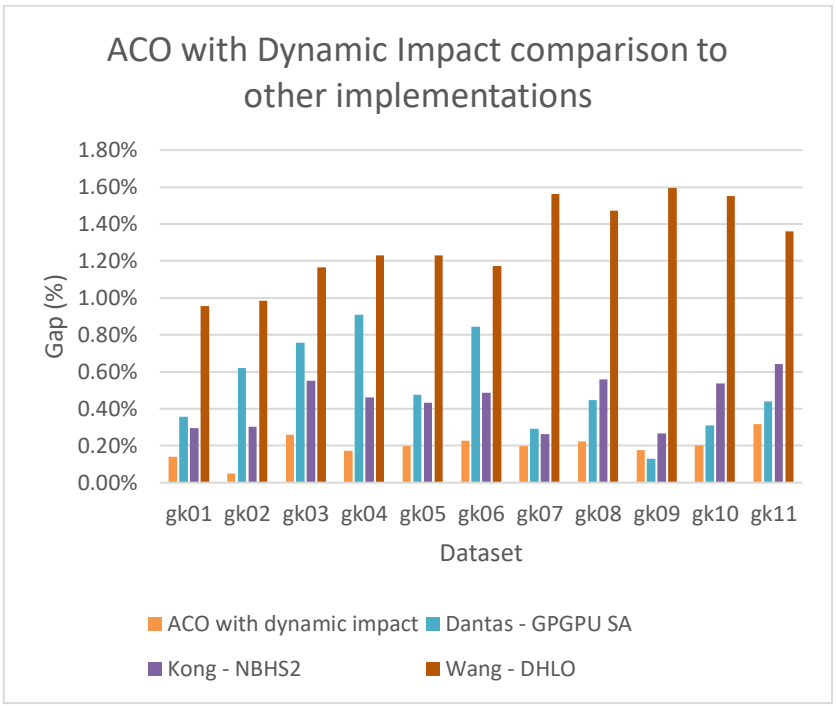

Fig 3. ACO with Dynamic Impact comparison to other recently published GK dataset solution results.

type 2 was best performing for GK datasets. Wang – DLHO [42] proposed a Diverse Human Learning Optimization algorithm that has performed the best among compared solutions. On average, ACO with Dynamic Impact has a 0.31% or 3.3 times lower gap than Dantas – GPGPU SA. However, ACO is outperformed by a 0.05% gap difference on a single gk09 instance. Kong – NBHS2 has closer performance and is, on average, 0.24% or 2.48 times behind ACO. However, no instances outperform ACO, and the closest instance is gk07, falling behind by 0.07% or 1.35 times. Lastly, ACO outperforms Wang – DLHO on average by 1.10% or 7.72 times.

In conclusion, Dynamic Impact proved to significantly aid the search for small datasets reliably reaching optimal value and large datasets significantly lower gap to the optimal or best-known value. This ACO with Dynamic Impact currently is the best performing algorithm for solving both large and small MKP benchmark instances.

5. Conclusions

This research has studied the Ant Colony Optimization algorithm applied to the MMPPFO problem. The main optimization objective depends on a collection of smaller parts where each part does not have any priority over another. Therefore useful heuristic information that could be predefined does not exist. The research has proposed an additional component to the ACO algorithm probability calculation, which is called Dynamic Impact. Dynamic Impact, similarly to heuristic information, is a myopic component of the search. The difference is, Dynamic Impact is calculated each time probability is calculated, and it depends on the state of the partial solution. In other words, Dynamic Impact is a simplified evaluation of each edge's impact on fitness

Table 5: MKP GK datasets. ACO results with Dynamic Impact are compared against ACO without Dynamic Impact as well as best performing other algorithm results taken from recently published papers.

			Average profit		Average gap				
Dataset	problem size (N x M)	Best known profit	ACO without Dynamic Impact	ACO with Dynamic Impact	ACO without Dynamic Impact	ACO with Dynamic Impact	Dantas-GPGPU SA [40] 2018	Kong-NBHS2 [41] 2015	Wang-DHLO [42] 2017
gk01	100 x 15	3766	3750.7	3760.7	0.41%	**0.14%**	0.36%	0.29%	0.96%
gk02	100 x 25	3958	3937.2	3956	0.53%	**0.05%**	0.62%	0.30%	0.99%
gk03	150 x 25	5656	5621.8	5641.3	0.60%	**0.26%**	0.76%	0.55%	1.17%
gk04	150 x 50	5767	5733.5	5757	0.58%	**0.17%**	0.91%	0.46%	1.23%
gk05	200 x 25	7560	7511.8	7545	0.64%	**0.20%**	0.48%	0.43%	1.23%
gk06	200 x 50	7677	7621.8	7659.7	0.72%	**0.23%**	0.85%	0.49%	1.17%
gk07	500 x 25	19221	19104.1	19183.29	0.61%	**0.20%**	0.29%	0.26%	1.56%
gk08	500 x 50	18806	18662.3	18764	0.76%	**0.22%**	0.45%	0.56%	1.47%
gk09	1500 x 25	58089	57466.1	57987.2	1.07%	0.18%	**0.13%**	0.27%	1.59%
gk10	1500 x 50	57295	56703.9	57179.2	1.03%	**0.20%**	0.31%	0.54%	1.55%
gk11	2500 x 100	95238	94111.6	94937.6	1.18%	**0.32%**	0.44%	0.64%	1.36%

function and resource consumption. Computational overhead to use this method is shown to be low when implementation is optimized for the specific problem. For the MMPPFO problem, this research has demonstrated that using ACO with Dynamic Impact has significantly improved final solution quality over the ACO without Dynamic Impact over the same number of search iterations.

Furthermore, ACO with Dynamic Impact showed significant improvements when applied to the Multidimensional knapsack problem. For the small SAC94 benchmark datasets, Dynamic Impact solves all instances to the optimal solution very quickly, which is a significant improvement compared to peer published research. For the large GK benchmark datasets, Dynamic Impact, on average, 4.26 times closer to the best-known or optimal result within the same search efforts. All the results show that ACO with Dynamic Impact is a new state of the art algorithm to solve resource-constrained optimization problems.

6. References

[1] M. Dorigo, "Optimization, Learning and Natural Algorithms," 1992.

[2] D. Merkle, M. Middendorf and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling.," *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation,* pp. 893-900, 2000.

[3] A. Bauer, B. Bullnheimer, R. Hartl and C. Strauss, "An ant colony optimization approach for the single machine total tardiness problem," *IEEE,* vol. 2, pp. 1445 - 1450, 1999.

[4] A. Colorni, M. Dorigo, V. Maniezzo and M. & Trubian, "Ant system for job-shop scheduling," *Belgian Journal of Operations Research, Statistics and Computer Science,* vol. 34, no. 1, pp. 39-53, 1994.

[5] R.-H. Huang and C.-L. Yang, "Ant colony system for job shop scheduling with time windows," *The International Journal of Advanced Manufacturing Technology,* vol. 39, no. 1, pp. 151 - 157, 2008.

[6] G. Leguizamon and Z. Michalewicz, "A new version of ant system for subset problems," *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406),* vol. 2, pp. 1459 - 1464, 1999.

[7] M. Kong, P. Tian and Y. Kao, "A new ant colony optimization algorithm for the multidimensional Knapsack problem," *Computers and Operations Research,* vol. 35, no. 8, pp. 2672 - 2683, 2008.

[8] D. Favaretto, E. Moretti and P. Pellegrini, "Ant colony system for a VRP with multiple time windows and multiple visits," *Journal of Interdisciplinary Mathematics,* vol. 10, no. 2, pp. 263 - 284, 04/2007.

[9] A. Fayeez and E. Keedwell, "H-ACO: A Heterogeneous Ant Colony Optimisation approach with Application to the Travelling Salesman," *Artificial Evolution,* pp. 144-161, 2018.

[10] M. Dorigo and T. Stützle, "Ant colony optimization: overview and recent advances," *Handbook of metaheuristics,* pp. 311-351, 2019.

[11] X. Li and R. Y. K. Fung, "Optimal K-unit cycle scheduling of two-cluster tools with residency constraints and general robot moving times," *Journal of Scheduling,* vol. 19, no. 2, pp. 165 - 176, 2016.

[12] C. Schwenke and K. Kabitzsch, "Continuous flow transport scheduling for conveyor-based AMHS in wafer fabs," *2017 Winter Simulation Conference (WSC),* pp. 3588 - 3599, 2017.

[13] C. Guo, Z. Jiang, H. Zhang and N. Li, "Decomposition-based classified ant colony optimization algorithm for scheduling semiconductor wafer fabrication system," *Computers & Industrial Engineering,* vol. 62, no. 1, pp. 141-151, 2012.

[14] S. Khuri, T. Bäck and J. Heitkötter, "The zero/one multiple knapsack problem and genetic algorithms," *Proceedings of the 1994 ACM symposium on applied computing,* pp. 188 - 193, 1994.

[15] T. Stutzle and H. Hoos, "MAX-MIN Ant System and local search for the traveling salesman problem," *IEEE,* pp. 309 - 314, 1997.

[16] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan and J. R. Woodward, "Exploring Hyper-heuristic Methodologies with Genetic Programming," *Computational intelligence: Collaboration, fusion and emeregense,* pp. 177-201, 2009.

[17] Z. A. Aziz, "Problem, Ant Colony Hyper-heuristics for Travelling Salesman," *Procedia computer science,* vol. 76, pp. 534 - 538, 2015.

[18] B. Duhart, F. Camarena, J. C. Ortiz-Bayliss, I. Amaya and H. Terashima-Marín, "An Experimental Study on Ant Colony Optimization Hyper-Heuristics for Solving the Knapsack Problem," *Pattern Recognition,* pp. 62 - 71, 2018.

[19] I. Dzalbs and T. Kalganova, "Simple generate-evaluate strategy for tight-budget parameter tuning problems," *2020 IEEE Symposium Series on Computational Intelligence (SSCI),* pp. 783 - 790, 2020.

[20] M. Mavrovouniotis, F. M. Muller and S. Yang, "Ant Colony Optimization With Local Search for Dynamic Traveling Salesman Problems," *IEEE transactions on cybernetics,* vol. 47, no. 7, pp. 1743 - 1756, 2017.

[21] R. Zhang, S. Song and C. Wu, "Robust Scheduling of Hot Rolling Production by Local Search Enhanced Ant Colony Optimization Algorithm," *IEEE transactions on industrial informatics,* vol. 16, no. 4, pp. 2809 - 2819, 2020.

[22] R. Keller and R. Poli, "Toward subheuristic search," *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence),* pp. 3148 - 3155, 2008.

[23] S. Xu, Y. Liu and M. Chen, "Optimization of partial collaborative transportation scheduling in supply chain management with 3PL using ACO," *Expert Systems With Applications,* vol. 71, pp. 173 - 191, 2017.

[24] M. Dorigo and T. Stutzle, Ant Colony Optimization, 1st ed., Massachusetts: The MIT Press, 2004.

[25] M. Veluscek, T. Kalganova, P. Broomhead and A. Grichnik, “Composite goal methods for transportation network optimization,” *Expert Systems With Applications,* vol. 42, no. 8, pp. 3852 - 3867, 05/2015.

[26] I. Dzalbs and T. Kalganova, “Accelerating supply chains with Ant Colony Optimization across range of hardware solutions,” *Computers & industrial engineering,* vol. 147, no. arXiv:2001.08102, p. 106610, 2020.

[27] S. Fidanova, “Heuristics for multiple knapsack problem,” *IADIS AC,* pp. 255-260, 2005.

[28] M. Dorigo and L. Gambardella, “Ant colony system: a cooperative learning approach to the traveling salesman problem,” *IEEE Transactions on Evolutionary Computation,* vol. 1, no. 1, pp. 53 - 66, 1997.

[29] S. Jonas, “Microchip Manufacturing Plant Production Floor Optimization (MMPPFO) problem,” Figshare, 2020.

[30] J. H. Drake, “Benchmark instances for the Multidimensional Knapsack Problem,” 2015. [Online]. Available: https://www.researchgate.net/publication/271198281_Benchmark_instances_for_the_Multidimensional_Knapsack_Problem. [Accessed 30 04 2019].

[31] S. V. Chupov, “An Approximate Algorithm for Lexicographic Search in Multiple Orders for the Solution of the Multidimensional Boolean Knapsack Problem,” *Cybernetics and Systems Analysis,* vol. 54, no. 4, pp. 563 - 575, 07/2018.

[32] M. Vasquez and J.-K. Hao, “A hybrid approach for the 0-1 multidimensional knapsack problem,” *IJCAI* , pp. 328-333, 2001.

[33] F. Glover and G. A. Kochenberger, “Critical event tabu search for multidimensional knapsack problems.,” *Meta-Heuristics Springer,* pp. 407-427, 1996.

[34] M. Chih, C.-J. Lin, M.-S. Chern and T.-Y. Ou, “Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem,” *Applied Mathematical Modelling,* vol. 38, no. 4, pp. 1338 - 1350, 02/2014.

[35] H. Peng, Z. Wu, P. Shao and C. Deng, “Dichotomous Binary Differential Evolution for Knapsack Problems,” *Mathematical Problems in Engineering,* pp. 1 - 12, 2017.

[36] M. Abdel-Basset, D. El-Shahat, I. El-Henawy and A. K. Sangaiah, “A modified flower pollination algorithm for the multidimensional knapsack problem: human-centric decision making,” *Springer Berlin Heidelberg,* vol. 22, no. 13, pp. 4221 - 4239, 07/2018.

[37] L. F. Mingo López, N. Gómez Blas and A. Arteta Albert, “Multidimensional knapsack problem optimization using a binary particle swarm model with genetic operations,” *Soft Computing,* vol. 22, no. 8, pp. 2567 - 2582, 04/2018.

[38] J. Liu, C. Wu, J. Cao, X. Wang and K. L. Teo, “A Binary differential search algorithm for the 0–1 multidimensional knapsack problem,” *Applied Mathematical Modelling,* vol. 40, no. 23-24, pp. 9788 - 9805, 12/2016.

[39] X. Zhang, C. Wu, J. Li, X. Wang, Z. Yang, J.-M. Lee and K.-H. Jung, "Binary artificial algae algorithm for multidimensional knapsack problems," *Applied Soft Computing,* vol. 43, pp. 583 - 595, 06/2016.

[40] B. de Almeida Dantas and E. N. Cáceres, "An experimental evaluation of a parallel simulated annealing approach for the 0–1 multidimensional knapsack problem," *Journal of Parallel and Distributed Computing,* vol. 120, pp. 211 - 221, 10/2018.

[41] X. Kong, L. Gao, H. Ouyang and S. Li, "Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm," *Computers and Operations Research,* vol. 63, pp. 7 - 22, 11/2015.

[42] L. Wang, L. An, J. Pi, M. Fei and P. M. Pardalos, "A diverse human learning optimization algorithm," *Journal of Global Optimization,* vol. 67, no. 1, pp. 283 - 323, 01/2017.

[44] C. Solnon, "Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization," *European Journal of Operational Research,* vol. 191, no. 3, pp. 1043 - 1055, 2008.

[45] B. Guan and Y. Zhao, "Self-Adjusting Ant Colony Optimization Based on Information Entropy for Detecting Epistatic Interactions," *Genes,* vol. 10, no. 2, p. 114, 02/2019.

[46] S. Tabakhi and P. Moradi, "Relevance–redundancy feature selection based on ant colony optimization," *Pattern Recognition,* vol. 48, no. 9, pp. 2798 - 2811, 09/2015.

[47] R. Devi Priya and R. Sivaraj, "Imputation of Discrete and Continuous Missing Values in Large Datasets Using Bayesian Based Ant Colony Optimization," *Arabian Journal for Science and Engineering,* vol. 41, no. 12, pp. 4981 - 4993, 12/2016.

[48] W. Long and Y. Jinjiang, "Single-machine scheduling to minimize the total earliness and tardiness is strongly NP-hard," *Operations Research Letters,* vol. 41, no. 4, pp. 363-365, 2013.

[49] H.-H. Huang, C.-H. Huang and W. Pei, "Solving Multi-Resource Constrained Project Scheduling Problem using Ant Colony Optimization," *Journal of Engineering,* vol. 5, no. 1, pp. 2 - 12, 2015.

[50] J. Drake, E. Ozcan and E. K. Burke, "A Case Study of Controlling Crossover in a Selection Hyper-heuristic Framework Using the Multidimensional Knapsack Problem," *Evolutionary Computation,* vol. 24, no. 1, pp. 113-141, 2016.

[51] D. M., Z. M., M. N. and B. M., "Updating ACO Pheromones Using Stochastic Gradient Ascent and Cross-Entropy Methods," *Applications of Evolutionary Computing,* vol. 2279, pp. 21-30, 2002.

[52] M. Tuba and R. Jovanovic, "Improved ACO Algorithm with Pheromone Correction Strategy for the Traveling Salesman Problem," *International Journal of Computers Communications & Control,* vol. 8, no. 3, p. 477, 2013/06.

[53] D. K. Karpouzos and K. L. Katsifarakis, “A Set of New Benchmark Optimization Problems for Water Resources Management,” *Water Resources Management,* vol. 27, no. 9, pp. 3333 - 3348, 2013.

[54] P. Baniasadi, V. Ejov, M. Haythorpe and S. Rossomakhine, “A new benchmark set for Traveling salesman problem and Hamiltonian cycle problem,” *arXiv preprint arXiv:1806.09285,* 2018.

[55] G. Leguizamon and Z. Michalewicz, “A new version of ant system for subset problems,” *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406),* vol. 2, pp. 1459 - 1464, 1999.