HEDCOS: High Efficiency Dynamic Combinatorial Optimization System using Ant Colony Optimization algorithm

A Thesis Submitted for the Degree of

Doctor of Philosophy

By

Jonas Skackauskas

Department of Electronic and Electrical Engineering

Brunel University London

September 2022

Declaration of authorship

I, Jonas Skackauskas, declare that the work completed in this thesis is my original work conducted in accordance with the Code of Practice for Research Degrees. The work presented in this thesis has not been used in any other submission for an academic award.

Signature: [illegible] Date: 24/09/2022

Abstract

Dynamic combinatorial optimization is gaining popularity among industrial practitioners due to the ever-increasing scale of their optimization problems and efforts to solve them to remain competitive. Larger optimization problems are not only more computationally intense to optimize but also have more uncertainty within problem inputs. If some aspects of the problem are subject to dynamic change, it becomes a Dynamic Optimization Problem (DOP).

In this thesis, a High Efficiency Dynamic Combinatorial Optimization System is built to solve challenging DOPs with high-quality solutions. The system is created using Ant Colony Optimization (ACO) baseline algorithm with three novel developments.

First, introduced an extension method for ACO algorithm called Dynamic Impact. Dynamic Impact is designed to improve convergence and solution quality by solving challenging optimization problems with 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).

Second, a non-stochastic dataset generation method was introduced to solve the dynamic optimization research replicability problem. This method uses a static benchmark dataset as a starting point and source of entropy to generate a sequence of dynamic states. Then using this method, 1405 Dynamic Multidimensional Knapsack Problem (DMKP) benchmark datasets were generated and published using famous static MKP benchmark instances as the initial state.

Third, introduced a nature-inspired discrete dynamic optimization strategy for ACO by modelling real-world ants' symbiotic relationship with aphids. ACO with Aphids strategy is designed to solve discrete domain DOPs with event-triggered discrete dynamism. The strategy improved inter-state convergence by allowing better solution recovery after dynamic environment changes. Aphids mediate the information from previous dynamic optimization states to maximize initial results performance and minimize the impact on convergence speed. This strategy is tested for DMKP and against identical ACO implementations using Full-Restart and Pheromone-Sharing strategies, with all other variables isolated.

Overall, Dynamic Impact and ACO with Aphids developments are compounding. Using Dynamic Impact on single objective optimization of MMPPFO, the fitness value was improved by 33.2% over the ACO algorithm without Dynamic Impact. MKP benchmark instances of low complexity have been solved to a 100% success rate even when a high degree of solution sparseness is observed, and large complexity instances have shown the average gap improved by 4.26 times. ACO with Aphids has also demonstrated superior performance over the Pheromone-Sharing strategy in every test on average gap reduced by 29.2% for a total compounded dynamic optimization performance improvement of 6.02 times. Also, ACO with Aphids has outperformed the Full-Restart strategy for large datasets groups, and the overall average gap is reduced by 52.5% for a total compounded dynamic optimization performance improvement of 8.99 times.

Acknowledgements

Firstly, I would like to thank my principal supervisor Prof. Tatiana Kalganova for giving me the opportunity to do the PhD degree and diligently guiding me to completion; and my second supervisor Dr Ian Dear for helping me with writing.

Then, I would like to thank my partner for continuous support and patience throughout the entire duration of my studies.

And finally, I would like to express my gratitude to my father, who provided moral support, led by example, and encouraged me to persevere.

Content

List of tables

List of figures

List of Abbreviations

AAM	Aphid–Ant Mutualism
ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
ACOCA	Ant Colony Optimization with Cooperative Aphid
ACS	Ant Colony System
AIS	Artificial Immune System
ARA	Artificial Raindrop Algorithm
AS	Ant System
ASrank	Rank-based Ant System
BAAA	Binary Artificial Algae Algorithm
BPSOTVAC	Binary Particle Swarm Optimization Time-Varying Acceleration Coefficients
CAP	Cartesian Ant Programming
CGP	Cartesian Genetic Programming
CS	Cuckoo Search
CSSA	Chaotic Salp Swarm algorithm
DBDE	Dichotomous Binary Differential Evolution
DE	Differential evolution
DGAA	Distributed Guidance Anti-flocking Algorithm
DLEA	Dynamic Learning Evolution Algorithm
DLHO	Diverse Human Learning Optimization
DMFO	Dynamic Multimodal Function Optimization
DMKP	Dynamic Multidimensional Knapsack Problem
DMOOP	Dynamic Multi-Objective Optimization Problem
DOP	Dynamic Optimization Problem
DTSP	Dynamic Traveling Salesman Problem
DVRP	Dynamic Vehicle Routing Problem
EA	Evolutionary Algorithm
FA	Firefly Algorithm
FA	Firefly Algorithm
GA	Genetic Algorithm
GPGPU	General-Purpose Graphics Processing Units
HEDCOS	High Efficiency Dynamic Combinatorial Optimization System
HPSOGO	Binary Particle Swarm Optimization with Genetic Operations
JSP	Job-shop Scheduling Problem
MBO	Monarch Butterfly Optimization
MC	Membrane Computing
MFPA	Flower Pollination Algorithm
MKP	Multidimensional Knapsack Problem
MMAS	Min-Max Ant System
MMPPFO	Microchip Manufacturing Plant Production Floor Optimization

MPB	Moving Peaks Benchmark
MS	Moth Search
MWN	Mobile Wireless Network
NBHS	New Binary Harmony Search
NSGA	Non-dominated Sorting Genetic Algorithm
PA	Parallel Ant
P-ACO	Population-based ACO
PAES	Pareto Archived Evolution Strategy
PSO	Particle Swarm Optimization
ROOT	Robust Optimization Over Time
SA	Simulated Annealing
SAM	State Adjustment Magnitude
SDN	Software Defined Network
SMA	Slime Mould Algorithm
TR-BDS	Random Binary Differential Search algorithm using the Tanh function
VRP	Vehicle Routing Problem

Chapter 1. Introduction

In the current economic environment, where industries are expected to have ever-increasing production output, the drive towards cost-effectiveness and efficient resource allocation creates an ever-increasing demand for high-quality solutions to business decision problems. Even a tiny fraction of cost-saving improvement on a large business operation can significantly improve the business's bottom line. Combinatorial optimization can answer business questions such as “What is the best factory production schedule to have the lowest overall delays of the finished product?” or “What collection of stocks will provide the highest risk-adjusted return for a set budget?”. While small optimization problems are easy to solve, many of us implicitly make optimal decisions in our daily routines, like taking the shortest path to visit a grocery store. Larger problems like university timetables can become challenging. The complexity of such problems is exponential, meaning a tiny increase in problem size significantly increases the efforts required to evaluate all possible combinations required to find the optimal one. Such complexity decision problems are classified as NP-Hard optimization problems, where an optimal solution cannot be found or verified within polynomial time.

For large optimization problems finding the optimal solution is practically impossible. Many of those problems present the search space that even the fastest supercomputers would take thousands of years to find the optimal solution. However, finding near-optimal solutions is usually good enough for practical applications if the time taken to find the solution is significantly reduced. Some metaheuristic optimization algorithms allow intelligently searching through a small subset of the total search space and achieving a good solution.

Not all optimization algorithms are made equal. Every optimization algorithm has strengths and weaknesses, and no optimization algorithm is the best choice for all optimization problems. Over the last several decades, researchers have proposed numerous algorithmic improvements and extensions that improve the algorithm's strengths or mitigate weaknesses. However, growing industrial competitiveness constantly demands for better optimization algorithms and methods.

Some optimization problems are also subject to dynamically changing environments. The change may occur to any aspect of the optimization problem, which changes the optimal solution and possibly invalidates the solutions that are found. Dynamic optimization is more difficult because the optimization algorithm has to find a new solution every time dynamic change occurs. In a more recent development, metaheuristic optimization algorithms were attempted to adapt to dynamic optimization problems, where learned information is reused for further optimization after the dynamic change.

This thesis explores algorithmic improvements of one metaheuristic optimization algorithm called Ant Colony Optimization. The goal of algorithmic improvements is to increase dynamic optimization solution quality. This goal is achieved through improvements to dynamic optimization algorithmic methods and improvements to static optimization convergence.

1.1. Motivation

Many industries rely on optimization for everyday operations. The most practical uses of optimization are in the fields of scheduling, transportation, supply chain management, financial portfolio management, and production control. Optimization algorithm efficiency matters a lot in a fiercely competitive market. The quality of solutions and optimization methods can enable more efficient use of resources and allow a business to adapt to disruptions quicker. These disruptions can be as minor as delayed container truck delivery or as significant as labour shortages caused by Covid-19 pandemic lockdowns or disrupted material supplies due to political sanctions.

Complex optimization problems often produce fragile solutions. When an optimization problem accounts for a multitude of variables from many sources, the likelihood of some variables changing and invalidating the solution is very high. For those large optimization problems, changes can occur so frequently that the time taken to find an acceptable quality solution is longer than the interval between changes. Dynamic implementations of optimization algorithms, aiming to minimize the time needed to find new solutions after the dynamic change, have gained more traction recently in the research community. Usually, those dynamic changes are reasonably small, such that overall solutions do not change significantly. Dynamic optimization algorithms exploit this similarity and reuse learned information to find new solutions faster after the dynamic change.

Ant Colony Optimization is an excellent algorithm to implement dynamic optimization because ants' behaviour is highly adaptable in nature. However, previous implementations of the ACO algorithm for dynamic optimization are fairly rudimentary. Those implementations exploit the strengths of regular ACO like pheromone or population for the benefit of dynamic optimization. To further improve the quality and performance of ACO for dynamic optimization, the development of dedicated methods is necessary.

1.2. Thesis contributions

All of the research work presented in this thesis combines into the High Efficiency Dynamic Combinatorial Optimization System. HEDCOS is a significant original artefact that improves ACO solution quality for both static and dynamic optimization problems without sacrificing the length of run time. The research on HEDCOS consists of three major contributions to the science, which are presented in Chapters 3, 4, and 5. Chapter 3 and Chapter 5 propose new methods for the ACO algorithm that significantly improve the quality of the solutions for static and dynamic optimization, respectively. Chapter 4 proposes a new, fully defined combinatorial dynamic optimization benchmark created using deterministic procedures. These benchmark datasets are necessary to prove the validity of the work presented in Chapter 5.

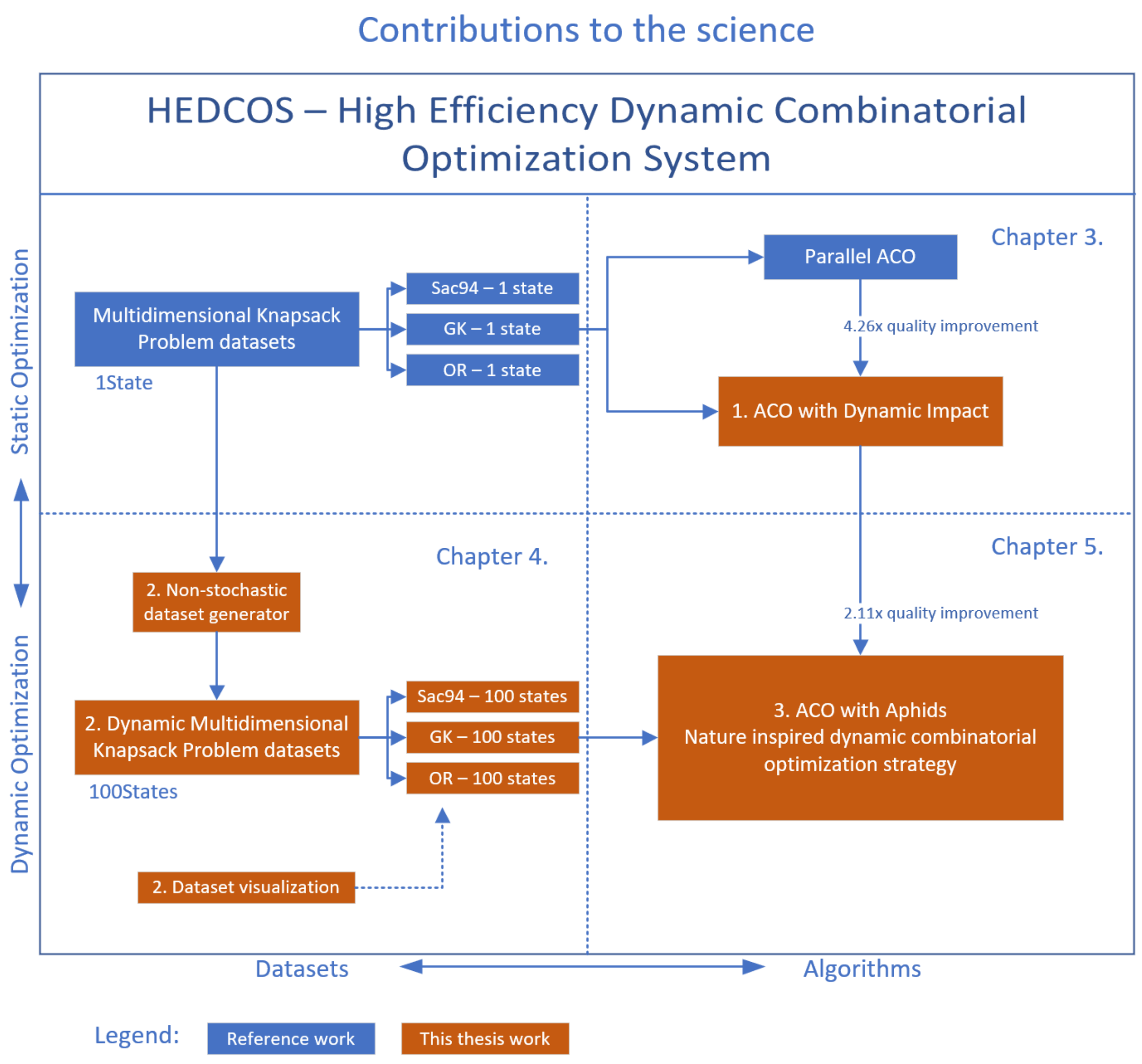
Contributions to the science
HEDCOS – High Efficiency Dynamic Combinatorial Optimization System
Static Optimization
Dynamic Optimization
Multidimensional Knapsack Problem datasets
1State
Sac94 – 1 state
GK – 1 state
OR – 1 state
Parallel ACO
Chapter 3.
4.26x quality improvement
1. ACO with Dynamic Impact
Chapter 4.
Chapter 5.
2. Non-stochastic dataset generator
2.11x quality improvement
2. Dynamic Multidimensional Knapsack Problem datasets
100States
Sac94 – 100 states
GK – 100 states
OR – 100 states
3. ACO with Aphids Nature inspired dynamic combinatorial optimization strategy
2. Dataset visualization
Datasets
Algorithms
Legend:
Reference work
This thesis work

Figure 1-1: HEDCOS – High Efficiency Dynamic Combinatorial Optimization System, contribution to the science of this thesis. HEDCOS is a collection of advancements for ACO algorithm aimed at achieving high-quality solutions quickly.

This thesis research work contributions can be summarized as follows:

1. **Dynamic Impact for the Ant Colony Optimization:** Dynamic Impact is an extension method for Ant Colony Optimization algorithm. Dynamic Impact is designed to improve convergence and solution quality solving complex optimization problems with a non-linear relationship between resource consumption and fitness, where resource consumption is the constraints of optimization problems and fitness is the optimization objective function. This proposed method is tested against the real-world Microchip Manufacturing Plant Production Floor Optimization problem and the theoretical benchmark Multidimensional Knapsack problem. Using Dynamic Impact on single objective

optimization, the fitness value is improved by 33.2% over the ACO algorithm without Dynamic Impact for the MMPPFO problem. Furthermore, the 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. The description of the ACO with Dynamic Impact and the results have been published in the peer-reviewed journal Swarm and Evolutionary Computation [1].

2. **Dynamic Multidimensional Knapsack Problem benchmark creation methodology and datasets:** A new non-stochastic dataset generation method is introduced to solve the research replicability problem. Implemented DMKP dataset generator and published it in GitHub [2]. Then, 1405 fully defined DMKP benchmark instances were generated using well-known static MKP benchmark instances as the initial state. The benchmark instances are published in GitHub [3] and IEEE Dataport [4]. Generated datasets were quantitatively and qualitatively analysed, including visualizations made with the tool published in GitHub [5]. Furthermore, 445 datasets have the optimal result found for each state using a linear solver. These fully defined DMKP datasets and the dataset generator contributes to solving the research replicability problem. The method description and dataset analysis have been published in the peer-reviewed journal Systems and Soft Computing [6].

3. **Ant Colony Optimization with Aphids for dynamic optimization problems:** A nature-inspired dynamic optimization strategy for ACO modelled after real-world ants' symbiotic relationship with aphids. The strategy improves the solution recovery after dynamic environment change, such that using information from previous dynamic optimization states initial results are as good as possible, and convergence speed is minimally impacted. This strategy is tested for DMKP and against identical ACO implementations using Full restart and Pheromone sharing strategies, with all other variables isolated. ACO with Aphids has demonstrated superior performance than the Pheromone share strategy in every test on average gap reduced by 29.2%. Also, ACO with Aphids has outperformed the Full restart strategy for large datasets groups, and the overall average gap is reduced by 52.5%. The algorithm description and results are submitted to the peer-reviewed journal Swarm and Evolutionary Computation 3 Sep 2022 [7].

1.3. List of publications

Following are the materials published as part of this research:

- Published articles in peer-reviewed journals:
    - J. Skackauskas, T. Kalganova, I. Dear and M. Janakiram, “Dynamic impact for ant colony optimization algorithm,” *Swarm and Evolutionary Computation,* 2021. DOI: https://doi.org/10.1016/j.swevo.2021.100993
    - J. Skackauskas and T. Kalganova, “Dynamic Multidimensional Knapsack Problem benchmark datasets,” *Systems and Soft Computing,* 2022. DOI: https://doi.org/10.1016/j.sasc.2022.200041
- Submitted articles to peer-reviewed journals:
    - J. Skackauskas and T. Kalganova, "Herder Ants: Ant Colony Optimization with Aphids for Discrete Event-Triggered Dynamic Optimization Problems," *Submitted to Swarm and Evolutionary Computation,* 3/9/2022.
- Published dataset materials:
    - J. Skackauskas, T. Kalganova and M. Janakiram, “Microchip Manufacturing Plant Production Floor Optimization (MMPPFO) problem,” Figshare, 2020. DOI: https://doi.org/10.17633/rd.brunel.11638323.v1
    - J. Skackauskas and T. Kalganova, "Dynamic-MKP Benchmark Datasets," *IEEE Dataport,* 2022. DOI: https://dx.doi.org/10.21227/6bfm-bj82
    - J. Skackauskas, “GitHub - Dynamic MKP Benchmark Datasets,” 2021. [Online]. Available: https://github.com/jonasska/Dynamic-MKP-Benchmark-Datasets.
    - J. Skackauskas, “GitHub - Dynamic MKP Benchmark Best Known Results,” 2021. [Online]. Available:

https://github.com/jonasska/Dynamic-MKP-Benchmark-Best-Known-results.

- Published software tools:
    - J. Skackauskas, “GitHub - Dynamic MKP Datasets Generator,” 2021. [Online]. Available: https://github.com/jonasska/Dynamic-MKP-Datasets-Generator.
    - J. Skackauskas, “GitHub - Dynamic MKP Datasets Visualization,” 2021. [Online]. Available: https://github.com/jonasska/Dynamic-MKP-Datasets-Visualization.

1.4. Thesis contents

This thesis is presented in 6 chapters. Chapter 1 introduces the background of the research, outlines the problem and why it is important to solve it. Also, Chapter 1 presents an overview of the research work conducted in this thesis.

Chapter 2 presents a literature review on dynamic optimization with a deeper focus on combinatorial dynamic optimization. The literature review analyses optimization problems and algorithms to solve them. Although there is a variety of optimization problems and algorithms, the literature review points out that it is difficult to compare the optimization algorithms solving any combinatorial optimization problems due to a lack of reputable combinatorial dynamic optimization benchmarks. Then, the literature review analyses in depth Ant Colony Optimization algorithm to solve combinatorial Dynamic Optimization Problems. Finally, in literature review presents two optimization problems applied in experimental work.

Chapter 3 proposes an extension method for the ACO algorithm called Dynamic Impact. Dynamic Impact 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. Two optimization problems were used to test this method, real-world MMPPFO and theoretical benchmark MKP. For both problems, adding Dynamic Impact measurably improved solution quality. Furthermore, in Chapter 3, the description of Dynamic Impact includes generalized formulas and a simplistic example that are easy to adopt for any optimization problem.

Chapter 4 introduces the deterministic dataset generation method that takes an existing static benchmark dataset as an initial state and generates a dynamic dataset with the desired number of states, where each state is an evolution from the previous state. Then using this method, generated 1405 DMKP benchmark datasets of five dynamism levels from the existing static MKP benchmark dataset library. Furthermore, Chapter 4 presents a statistical analysis of DMKP benchmark datasets and an optimal result analysis of small generated benchmark datasets. This benchmark solves the problem of research replicability for combinatorial dynamic optimization. All previously published research used stochastic methods to create dynamic optimization problems, making it difficult to verify the research claims.

Chapter 5 introduces a novel dynamic optimization strategy for the ACO algorithm by modelling real-world ants' symbiotic relationship with aphids. Detailed ACO with Aphids model includes pseudo-code and generalized formulas needed to implement the model for a wide range of dynamic optimization problems. Then ACO with Aphids was used to solve the DMKP benchmark and compared against Full-Restart and Pheromone-Sharing strategies. Compared to Full-Restart and Pheromone-Sharing, ACO with aphids has demonstrated better performance.

Finally, Chapter 6 presents the thesis conclusion and potential future research directions.

1.5. Aims and objectives

The central aim of this research was to develop and investigate innovative techniques that enhance the Ant Colony Optimization (ACO) algorithm for both static and dynamic optimization problems. The objectives that facilitated this aim are enumerated as follows:

1. **To enhance the static problem-solving capabilities of the ACO algorithm:** This was achieved by proposing the Dynamic Impact method, designed to improve convergence and solution quality for optimization problems with non-linear relationships between resource consumption and fitness.
2. **To contribute to the academic community through the creation of Dynamic Multidimensional Knapsack Problem benchmark datasets:** These datasets were created to solve the research replicability issue in the field

of combinatorial dynamic optimization. The datasets provide a basis for researchers to test and compare the efficacy of different dynamic optimization strategies.

3. **To augment the ACO algorithm for better performance in dynamic optimization problems:** The development of the 'Ant Colony Optimization with Aphids' technique was a major step in this direction. By leveraging the symbiotic relationship between ants and aphids, this technique was designed to enhance the solution recovery after dynamic environment changes, enabling the ACO algorithm to maintain its effectiveness in dynamically changing environments.
4. **To empirically evaluate the developed techniques:** The novel techniques were rigorously tested on real-world and theoretical benchmark problems. The results of these tests were used to assess the performance of the new techniques, providing valuable insights and leading to further refinements.
5. **To communicate the research outcomes:** The findings of the research have been published in peer-reviewed journals and other public platforms to contribute to the collective knowledge in the field of combinatorial optimization.

By achieving these objectives, the research aimed to contribute significantly to the body of knowledge on combinatorial optimization, providing practical tools for industries and pushing the boundaries of current scientific understanding.

1.6. Methodology

This research adopted a comprehensive, well-structured approach to ensure a thorough exploration of the topic. It began with an extensive literature review, which provided a deep understanding of the Ant Colony Optimization (ACO) algorithm, its modifications, and its various applications in tackling both static and dynamic optimization problems. This groundwork was crucial, as it highlighted the gaps in current research, thereby underscoring the need for enhanced, adaptable optimization strategies.

After the literature review, the research then focused on the design and development of the algorithmic enhancements to the ACO algorithm. The enhancements are systematically aimed at improving ACO ability to solve combinatorial dynamic optimization problems. Dynamic Impact method aimed to increase ACO algorithm capacity to handle complex, non-linear optimization problems. Ant Colony Optimization with Aphids aimed to boost the ACO algorithm's performance in dynamic environments. And finally, development of DMKP benchmark datasets and methodology which serves as a key resource for robust testing of the Dynamic Impact method, the ACO with Aphids algorithm, and other dynamic optimization strategies in the future.

Also, research methodology involves comprehensive empirical testing, data analysis, and results dissemination. Both the Dynamic Impact method and the ACO with Aphids technique are put through rigorous tests with established benchmarks. This enabled a robust assessment of their effectiveness and facilitated their comparison with other existing solutions. The outcomes were then meticulously analyzed and interpreted, leading to an exhaustive examination of the results, the understanding of the strengths and limitations of the developed methods, and the identification of potential avenues for future research.

Chapter 2. Literature review

Robust dynamic optimization approaches are of particular interest for real-world applications where problems at hand have a tendency to evolve over time and are not predictable accurately in advance. Often such problems must have some mechanisms to improve solutions in real-time as new data is obtained about disturbances or incremental accuracy improvements of the optimization data. Although the concept of dynamic optimization is not new, the research remains active to this day. Due to the real-world business interests in ever-increasing efficiency and growth, any incremental improvements in dynamic optimization are appreciated, according to a survey conducted by Mavrovouniotis et al. [8]. Also, according to the No Free Lunch theorem (NFL), all of the proposed strategies show an equivalent performance when applied to all possible optimization problems [9]. The NFL theorem states that a general-purpose optimization algorithm cannot be regarded as a universally-best choice. As a result, the NFL encourages searching for more efficient methods and developing new optimization techniques and strategies for different optimization problems.

This thesis aims to advance algorithmic methods that allow finding higher quality solutions for dynamic optimization problems using Ant Colony Optimization algorithm. Therefore, this literature review introduces the reader to the field of dynamic optimization with Dynamic Optimization Problems and reviews available benchmarks used to compare the algorithms solving DOPs and points out the dynamic combinatorial optimization replicability limitations. Then, the literature review provides a detailed look into the ACO algorithm and how ACO is currently used to solve DOPs. The literature review points out two research gaps to improve the efficiency of the ACO algorithm for combinatorial dynamic optimization. The need to improve sub-heuristics for general algorithm convergence and the need to improve the combinatorial dynamic optimization capabilities, possibly mimicking naturally occurring symbiotic relationships between ants and aphids. Then, two optimization problems are reviewed in detail to use in experimental work.

In the field of optimization, there are two major categories of optimization algorithms, exact heuristic algorithms and approximate metaheuristic algorithms. The exact heuristic algorithms are usually computationally efficient and find optimal solutions significantly faster than a brute force method. Such algorithms are Branch and Bound method [10], Linear Programming optimization [11], and Dynamic Programming [12]. However, these algorithms' computational complexity remains exponential, in big “O” notation $O(\mathrm{x}^n)$, where x is base of exponential intrinsic to the algorithm, and n is size of the problem, or these algorithms only find the local optimum [13]. For large NP-Hard optimization problems, only approximate intelligent metaheuristic methods have a chance to find a good solution with polynomial computational complexity $O(\mathrm{n}^x)$. The approximate method mean that final solutions may not be optimal, but for practical reasons those solutions are good enough. This literature review section focuses on these approximate intelligent metaheuristic algorithms.

2.1. Dynamic Optimization

Over the last couple of decades, the industry trend towards more efficient optimization of some business aspects has pushed researchers to work on dedicated optimization solutions that give a competitive advantage. One such solution is the algorithms dedicated to solving Dynamic Optimization Problems. DOPs are defined as a sequence of time-dependent problems where the fitness landscape shows some exploitable similarities before and after the dynamic change [14]. Such dynamic changes may have some parameters of the initial problem that can change, which does not entirely invalidate the existing solution, but remain useful for further optimization with some adjustments or fixes. Dynamic optimization is crucial when some optimization problem details are unknown beforehand and subject to unexpected changes. The most popular family of algorithms to solve such dynamic optimization problems is the Evolutionary Algorithms (EAs) family [15].

Dynamic Optimization Problems are problems where the fitness landscape changes over time due to various factors. Based on the characteristics of the change, the problem instances, and the nature of the solution space, DOPs can be classified into several types.

- Based on the Nature of the Search Space:
    - Continuous DOPs: These involve problems where the variables to be optimized are continuous in nature. They could be parameters of a system that can take real-number values within certain constraints.

    These problems include the optimization of multimodal functions, among others.
  - Discrete DOPs: These involve problems where the variables to be optimized are discrete in nature. They could be a sequence of decisions or selections that are to be made from a finite set. Problems like routing, scheduling, and placement are examples of discrete DOPs.
- Based on the Nature of the Changes:
  - Predictable DOPs: These are problems where the changes in the fitness landscape occur in a predictable manner. These changes could follow a certain pattern or trend that can be anticipated and prepared for in advance.
  - Unpredictable DOPs: These are problems where the changes in the fitness landscape are random or chaotic and cannot be predicted accurately. This unpredictability could be due to the nature of the system or the environment in which the problem is being solved.
- Based on the Frequency of Changes:
  - Periodic DOPs: These are problems where changes occur at regular intervals. The changes could be subtle or drastic, but the key characteristic is that they occur after fixed periods.
  - Aperiodic DOPs: These are problems where changes occur irregularly or at random intervals. The frequency of change is not constant and cannot be accurately predicted.
- Based on the Severity of Changes:
  - Small-scale DOPs: These are problems where changes in the fitness landscape are minimal and do not significantly affect the quality of the current solutions.
  - Large-scale DOPs: These are problems where changes in the fitness landscape are drastic and could render the current solutions completely irrelevant or sub-optimal.

Understanding this taxonomy of DOPs can be useful for choosing the most appropriate optimization algorithms and strategies. It helps in designing more effective and adaptive algorithms, especially for complex real-world problems that may involve multiple types of dynamic changes.

Continuous DOPs optimization is a widely researched topic with many published algorithms and improvements [8]. Continuous optimization problems are commonly modelled as functions mapping inputs to outputs, where both the input and output variables are real numbers and subject to constraints. It is possible for continuous optimization problems to be non-analytical and lack algebraic expressions of the search space [16]. The time-domain component can be either a real value for a continuously evolving problem domain or an integer value for a discretely evolving problem domain.

This thesis, focuses on Event-triggered DOPs that are Discrete, Unpredictable, Aperiodic, and Large-scale DOPs, as they pose some of the most challenging conditions for dynamic optimization. These Discrete DOPs are modelled as the combination of discrete decisions where solutions represent a logical set of conditions and are evaluated by an objective function [17]. The dynamism of Discrete DOPs is defined as a series of static optimization problem instances in sequential order called "states" [18]. Each state has a slight variation of the search space, where the larger variation makes the problem more dynamic and more challenging to solve.

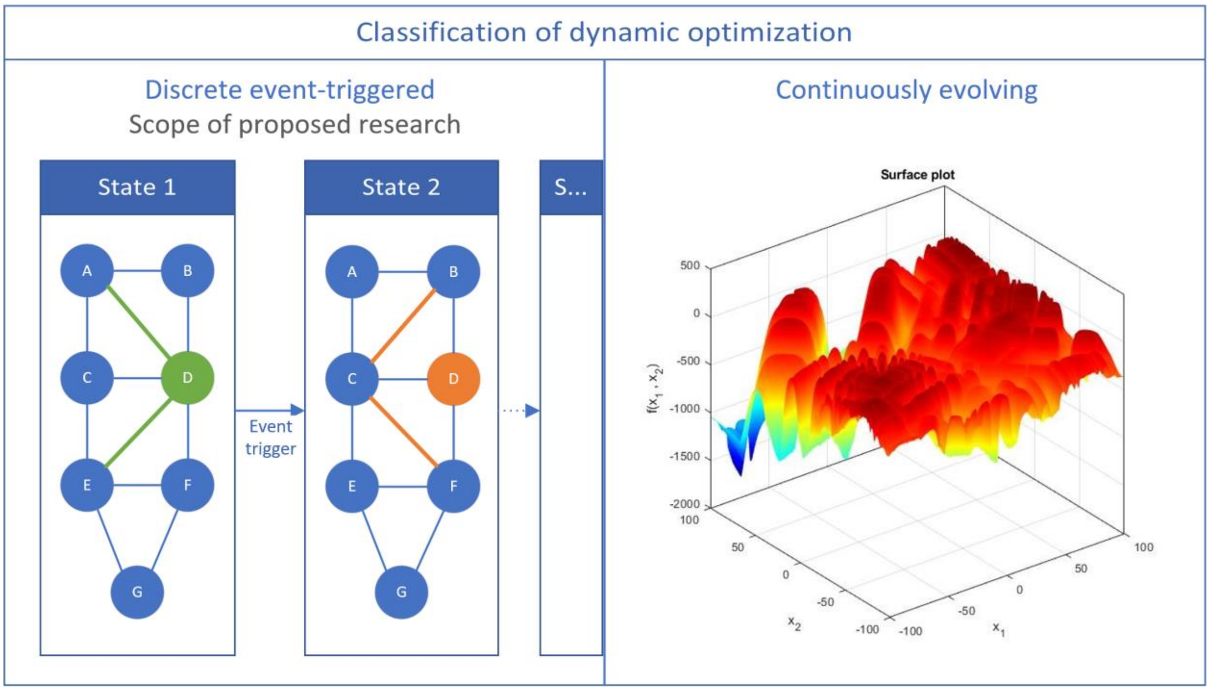

Figure 2-1: Classification of dynamic optimization. On the left-hand side, Discrete dynamic optimization is displayed with a discrete optimization problem state change based on an event trigger. On the right-hand-side popular Continuous DOP, a moving peaks benchmark is visualized for two input dimensions from IEEE CEC 2022

benchmark set [19]. This research focuses on solving types of problems portrayed on the left-hand side.

In the field of dynamic optimization, Dynamic Multimodal Function Optimization (DMFO), a continuous optimization problem, is by far the most popular theoretical dynamic optimization problem. Often, some researchers mistakenly consider it as the only Dynamic Optimization Problem, where most theoretical research comparisons are presented [20], [21], [22]. However, the field of dynamic optimization is not limited to continuous optimization problems. In fact, there are significantly more examples of discrete combinatorial optimization problems [8], [23].

2.1.1. Dynamic Optimization Problems and real-world applications

There are uncountably many real-world optimization problems. Every useful implementation of a real-world problem considers optimization with a unique set of parameters and goals to provide maximum benefit [24]. However, those differences usually are generalizable such that the majority of these problems can fall within one of the several major types of optimization problems known to researchers like routing, assignment, scheduling, subset, and packing problems [25], [26], [27]. Algorithms that solve one type of theoretical optimization problem often prove to be useful in solving equivalent real-world optimization problems [28]. Some of the real-world optimization problems published in the literature are categorized in Table 2-1.

Most of the practical use cases of dynamic optimization are in the fields of transportation, facility control, production, scheduling, and communications. These problems have a finite number of possible permutations and are formulated as combinatorial optimization problems. Several examples of applied dynamic optimization to a real-world problem are Traffic signal timing solved using a Genetic Algorithm (GA) [29]. Control parameter optimization using Particle Swarm Optimization (PSO) algorithm [30]. The Chaotic Salp Swarm algorithm (CSSA), which is based on the PSO algorithm, has been applied to optimize a Software Defined Network (SDN) to minimize deployment cost and latency [31]. Distributed Guidance Anti-flocking Algorithm (DGAA) has been applied to Mobile Wireless Network (MWN) optimization [32]. The Ant Colony Optimization algorithm has been applied to the

Railway Junction Rescheduling problem that aims to solve dynamic multi-objective optimization and minimize timetable deviation and energy expenditure [33].

Table 2-1: Problem types of real-world optimization solved in the sample literature.

Type	Real-world optimization problem	Problem type				
		Routing	Assignment	Scheduling	Subset	Packing
Dynamic	Traffic signal timing			[29]		
	Control parameter optimization		[30]			
	Software Defined Network	[31]				
	Mobile Wireless Network	[32]				
	Railway Junction Rescheduling			[33]		
	Wireless Sensor Network	[34]				
Static	Pipe Network Optimization	[35]				
	Vehicle Route	[36] [37]				
	Cargo loading					[38]
	Supply Chain Optimization	[39] [40]				
	Facility layout problem					[41] [42] [43]
	Project budgeting				[44]	
	Portfolio management				[45]	
	Cutting stock					[46]
	Integrated manufacturing floor optimization			[47] [48] [49] [50] [51] [52]		
	Tool sequencing			[53] [54]		
	School and university timetabling			[55] [56] [57]		
	Resource sharing problem		[58]			
	Brain tumour segmentation		[59]			
	Big-Data analysis		[60] [61] [62]			
	3D topology design		[63] [64]			
	Power consumption prediction		[65] [66]			
	Feature classification		[67] [68]			
	Reinforcement learning model design		[69] [70]			
	Manufacturing parameter optimization		[71] [72]			
	Operations scheduling			[73] [74]		
	Operations management			[75]		
	Minimum Spanning Tree	[76]				

2.1.2. Existing methods to solve DOPs

Dynamic optimization is more challenging to solve than static optimization because there are additional optimization goals besides the primary goal. In static optimization, the main goal is to maximize or minimize the fitness value of the optimization problem's objective, given that the optimization problem is a single objective. However, for dynamic optimization problems, the goal may not only be to minimize the fitness of the single objective but also to show some additional convergence dynamics required for successful optimization, like optimum tracking [77], best all states performance [78], best average fitness [79], algorithm stability throughout optimization [80]. Many developed dynamic optimization algorithms and algorithm extensions are designed to tackle one or more of these goals.

The swarm and evolutionary algorithms dominate the field of dynamic optimization. Genetic Algorithms [81], [82], [83], Particle Swarm Optimization [84], Artificial Raindrop Algorithm (ARA) [85], Membrane Computing (MC) [86], and Ant Colony Optimization [33], among many others [87], [88].

There are a lot of algorithmic improvements proposed to tackle the continuous DOP Moving Peaks Benchmark (MPB). For example, several combinations of PSO algorithm enhancements were tested to solve and track the optimum of MPB [89]. Several improved versions of EAs are used to solve these problems, like Non-dominated Sorting Genetic Algorithm II (NSGA-II) [90], [82], NSGA-III [91], Pareto Archived Evolution Strategy (PAES) [92], Dynamic Learning Evolution Algorithm (DLEA) [93]. Other dedicated algorithms for continuous dynamic optimization are Differential evolution (DE) [94], Firefly Algorithm (FA) [87], and Artificial Immune System (AIS) [88]. Overall scalability and performance of MPB have been explored on multiple algorithms with generated heterogeneous and multimodal benchmarks [95]. Robust Optimization Over Time (ROOT) methodology has been proposed to enable these algorithms to solve real-world optimization problems too [96].

Then, there are some examples adapted to solve Discrete DOPs and applied to both theoretical and real-world optimization problems [33]. Genetic Algorithms are used to optimize traffic signal timing control [29] as well as to solve the Dynamic Multidimensional Knapsack Problem [97]. Particle Swarm Optimization algorithms are used to optimize control parameters of a dynamic chemical process [30] as well as Dynamic Traveling Salesman Problem [98], [99]. Ant Colony Optimization algorithms are applied to railway junction scheduling problems [33] and DTSP [79].

The majority of purpose-built dynamic optimization algorithm research efforts are dedicated to continuous domain dynamic optimization, where existing optimization problems are well understood. Meanwhile, in the discrete domain, researchers primarily focus on problem-solving and adaptation of algorithms to meet the particular needs of those problems.

Table 2-2: Advantages and limitations of popular optimization algorithms solving discrete optimization problems in sample literature.

Algorithm	Advantages	Limitations
Genetic Algorithms [23] [29] [34] [77] [80] [97]	Versatile for large range of problems, both continuous and discrete.	Algorithm often require many function evaluations to converge, therefore can be slower than other algorithms.
Particle Swarm Optimization [30] [98] [99]	Simple algorithm that requires relatively few parameters to implement.	Algorithm can often get trapped in local optima and converge prematurely.
Ant Colony Optimization [25] [33] [78] [79]	Robust algorithm that can be applied to variety of combinatorial optimization problems without many changes.	Algorithm can struggle to solve continuous optimization problems.

2.1.3. Benchmarks to evaluate DOPs

A good benchmark dataset is necessary to evaluate dynamic optimization algorithm performance accurately. Currently, fully defined dynamic optimization benchmarks exist only for continuous domain Moving Peaks Benchmark (MPB), and discrete domain problems use either benchmark generators or real-world optimization instances.

The Moving Peaks Benchmark is a popular Continuous DOP type of benchmark with a wide selection of benchmark suites like IEEE Congress on Evolutionary Computation (IEEE CEC) benchmark suites [100], [101], [19], Composition Function (CF) Library [102], and other test functions of multiple authors [103], [104]. Moving Peaks Benchmark is a Dynamic Multimodal Function Optimization (DMFO) problem where the modal surface is changing. Such change is usually defined as one or more problem's input parameters changing over time continuously or in discrete increments [105]. The aim of DMFO is to track the global optimum point moving on the hypersurface over time [16]. However, researchers usually track more near optimal points too [106], [107]. The DMFO problem is well researched dynamic optimization problem type because it is easy to use already available benchmark datasets of static MFO problems [20]. Dynamic Multi-Objective Optimization Problem (DMOOP) is a very popular subset problem of DMFO, where multiple different functions must be optimized simultaneously as independent goals using the same input vector [21], [22].

For the discrete domain of dynamic optimization problems, researchers use benchmark generators where optimization states are generated during the optimization. A stochastic benchmark generator was used to create DMKP, where item profits, item weights and knapsack capacities are changed with a normally distributed random operator [97]. For a DTSP [98], [99] research work used a published TSPLIB [108] library and randomly modified vertex's location, which led to a generation of new problems each time. Then solved, the DTSP using the PSO algorithm. For the Dynamic Vehicle Routing Problem (DVRP) [109], research work used benchmark instances of static VRP and applied a stochastic modification algorithm to change the demands of the destinations and therefore recalculate the partially executed plan. Similarly, static benchmark instances of Job-shop Scheduling Problem (JSP) like FT instances introduced by Fischer and Thompson [110], LA instances introduced by Lawrence [111], and ABZ instances introduced by Adams et al. [112] are often extended with stochastic dataset generators [50].

There are many great examples of algorithmic advancements to benefit real-world dynamic optimization, as mentioned in the earlier section. However, there remains one problem with research conducted to solve real-world DOPs. Which is a lack of publicly available datasets and, in some cases, also lack the implementation details in order to reproduce the claims stated in the research conclusions. It is unreasonable to expect businesses to publish real recorded historical data that could be reused in further research, as it may be sensitive business information.

One apparent separation among academic research on dynamic optimization is that all of the fully defined benchmark dynamic optimization problems solved are in the continuous domain. All of the discrete benchmark problems are obtained from benchmark datasets of static optimization problems, modifying the dataset using stochastic methods. Stochastically generated problems can only be fairly compared if the initial seed of the random operator is used the same each time. Otherwise, the dataset optimums and the final result of the optimization problem would be different with each algorithm run. The comparison of different algorithms could be improved if the seed or better the dynamic optimization problem dataset instances or states were recorded and shared in full detail. However, none of the research work has shared neither seed nor dataset states, making it impossible to verify research claims and compare results with future research advancements. Ideally, the dynamic optimization problem datasets should be well-defined for each intermediate instance or state. The datasets should be created using a non-stochastic method such that the dynamic aspect of the dataset could be extended forward in the time domain.

2.2. Ant Colony Optimization algorithm

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 searching for food, it deposits pheromone on its path, which attracts more ants. If the food is better, an ant will deposit more pheromone, and therefore more ants will be attracted to the food source, making the path even more dense pheromone. All pheromone is evaporating slowly, and the paths that do not favour the search quickly become unattractive to the ants, see Figure 2-2. Originally Ant Colony Optimization algorithm was designed for the travelling salesman problem (TSP) described in Dorigo's [113] doctoral thesis in 1992. In ACO, ants use a stochastic construction heuristic to make probabilistic decisions based on the pheromone trails and heuristic information. While the heuristic information is based on the problem's search space, the pheromone represents cumulatively learned ants’ experience.

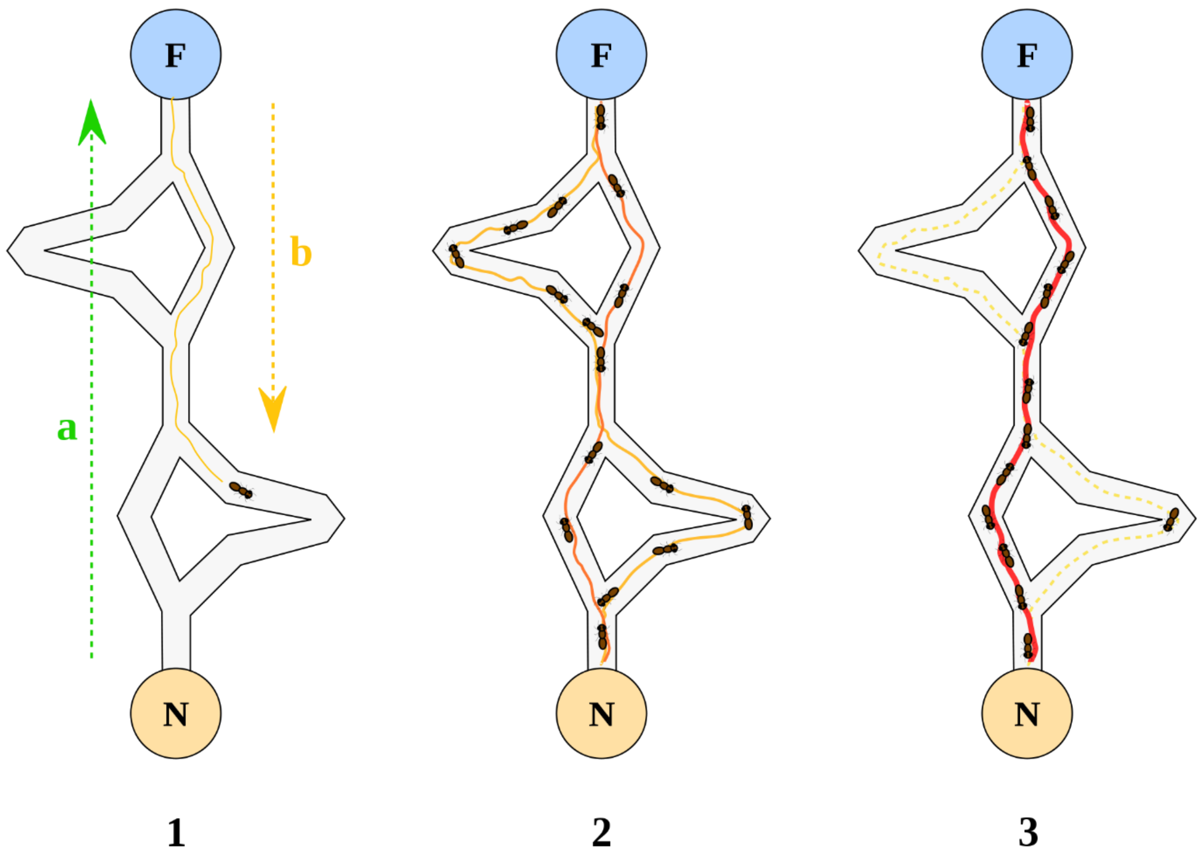

Figure 2-2: Illustration of ants searching for the shortest path

There are several ACO algorithm implementations based on ACO heuristics, the original Ant System (AS), and the most popular improvements of the Ant System are the rank-based Ant System (ASrank) introduced by Bullnheimer et al. [114], Min-Max Ant System (MMAS) introduced by Stützle and Hoos [115], Ant Colony System (ACS) introduced by Dorigo and Gambardella [116], and Population-based ACO (P-ACO) introduced by Guntsch and Middendorf [117]. The ASrank is an elitist strategy which sorts the ants according to their solution performance and only allows several best ants to deposit their pheromones after each iteration. The MMAS strategy uses the pheromones bound to a minimum and maximum pheromone levels. At the start, the MMAS strategy initializes pheromone values to the upper limit, which makes the algorithm begin the search with a higher level of exploration. Like ASrank, the MMAS strategy uses an elitist strategy for pheromone update, where local-best and potentially global-best ant's solution is used to update the pheromone [118]. Furthermore, the ACS uses probabilistic exploitation, where a portion of the path ants travel is chosen deterministically rather than stochastically [119]. Finally, the P-ACO uses a different method to update the pheromone. Instead of a typical pheromone update with a selected ant's result on every iteration, the P-ACO method keeps several ranked ants' solutions which are then used to create a new pheromone for every iteration [120], [121].

2.2.1. Formal ACO definition

The baseline ACO algorithm implementation used for all research in this thesis is a combination of the MMAS and the ACS, courtesy of M. Veluscek et al. [122]. The first step of algorithm execution is search space initialization, in which search space N is filtered for all nodes to have only feasible edges and calculated heuristic information $\eta_{j,i}$. Then according to MMAS strategy each edge's pheromone is set to the maximum value of τ_{max}. Once the search space is prepared, the iterative search starts. In the iterative search, a set of ants 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 one of two modes of finding an edge according to the ACS strategy [116]. First is the exploration mode, where each edge is added stochastically to the solution. 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:

$$p_{j,i} = \frac{\tau_{j,i}^{\alpha} \times \eta_{j,i}^{\beta}}{\sum(\tau_{j,i}^{\alpha} \times \eta_{j,i}^{\beta})}, \qquad \forall(j,i) \in N(s_p) \tag{2-1}$$

where τ is edge's pheromone, η is edge's heuristic information, $N(s_p)$ is the set of all feasible edges allowed to be added 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:

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

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 (2-1). 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 the percentage indicated by ρ parameter as in the following equation:

$$\tau_{j,i} := \tau_{j,i} * (1 - \rho), \qquad \forall(j,i) \tag{2-3}$$

where ρ is a constant parameter of the pheromone evaporation rate introduced by Dorigo and Stützle [123]. 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:

$$\tau_{j,i} := \tau_{j,i} + \rho * \Delta\tau_0, \qquad \forall(j,i) \in s_p \tag{2-4}$$

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

However, to utilize modern computer multicore architectures efficiently, Parallel Ant (PA) optimization architecture is implemented courtesy of I. Dzalbs et al. [39]. For each iteration, ants are split into several isolated local groups. At the start of the iteration, each group gets a new local pheromone copied from the global pheromone. Then each group performs a local search normally for all ants within the local group. All groups perform the search and evaluate the fitness of each solution in parallel. After all groups complete the search, the best performing ant’s solution is taken to perform the update on the global pheromone.

2.2.2. Strengths, weaknesses and typical applications

The Ant Colony Optimization algorithm was initially intended to solve the TSP, which aims to minimize the total path required to visit all cities [124]. The TSP is a graph problem where nodes represent cities and edges represent paths between cities, and each node must be visited only once [125]. Unlike other algorithms that maintain and modify a population of solutions, ACO fully builds all ant solutions. When ant searches for the next edge to add to the solution, all infeasible edges are calculated with zero probability [126]. This building heuristic allows ants to build only feasible solutions, which is a unique advantage for the ACO algorithm. The ACO algorithm is very good at solving constrained combinatorial optimization problems that can be expressed as a graph [127]. Furthermore, 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 and produces excellent results when little time is given to run the algorithm.

On the other hand, ACO is not an ideal algorithm for optimization problems that cannot be easily expressed as a graph problem. For example, in order to solve continuous optimization problems with ACO, the problem domain must be discretised [128]. There is also a dedicated implementation of ACO for continuous optimization problems called ACOR with the use of probability density functions generated from the population of ant solutions [129]. However, besides this algorithm's authors, researchers agree that other algorithms are better suited to solve continuous optimization [8].

ACO algorithm has been applied to a wide range of optimization problems that benefit from ACO solution building heuristics, both academic and real-world demonstrating the algorithm’s potential [25]. In the literature, research on ACO algorithm covers all major combinatorial problems: scheduling problems [130], [53], [127], [131], [132], subset problems [133], [134], [135], routing problems [136], [36], [137], and assignment problems [138], [139]. Also, the ACO algorithm has been adopted for many real-world optimization problems like Vehicle Routing Problem (VRP) [140], airline company crew scheduling [141], grid power management [142], [143], network routing optimization for cars [144] and wireless sensors [145], and supply chain optimization [39], [146].

2.2.3. Sub-heuristics

Combinatorial search algorithms are designed to explore large search spaces efficiently and quickly converge to a good solution. 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 iteratively learned 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 [115] is the introduction of heuristic information, which has 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 learned pheromone.

As in the case of all algorithms, ACO has tuneable hyper-parameters that, once found, a good combination of them for a specific problem 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. [147], which fine-tunes hyperparameters of the search algorithm in an automatic way for a genetic algorithm. Then, later these methods were applied to the ACO algorithm too [148], [149], [150].

In pursuit of better algorithm performance, the ACO algorithm can be coupled with a 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 [78], [151].

Furthermore, purpose-made sub-heuristic enhancements can improve search efficiency, converge faster, and produce better final solutions for certain problems. Sub-heuristics are the heuristic methods used in the core of search algorithms [152]. Sub-heuristics for ACO is not a well-researched area. Therefore, for clarification, the sub-heuristic method is referred to 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 are not generalized in any way. Authors [153] 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.

Table 2-3: Comparison of Heuristic Methodologies in Combinatorial Search Algorithms: Descriptions and Limitations

Methodology	**Description**	**Limitations**
Pheromone (ACO)	Primary metaheuristic method in Ant Colony Optimization that is iteratively learned when solving the optimization problem.	Often requires many iterations to reach a good solution.
Heuristic Information	Introduced by Stützle and Hoos to significantly improve search convergence speed and solution quality. It uses precalculated values for each edge to guide the ant's search.	There are no clear limitations mentioned for heuristic information, but its effectiveness might be problem-specific.
Hyper-heuristic Methods	Introduced by Burke et al. for genetic algorithms and later applied to ACO. Fine-tunes hyperparameters of the search algorithm automatically.	Finding a good combination of hyperparameters for a specific problem can be very time-consuming or require expert knowledge of the problem domain.
Local Search	A problem-specific local search coupled with ACO can improve performance. The local search applies a deterministic set of rules to a completed candidate solution to improve it before it is used to lay down the pheromones.	The effectiveness of local search might be limited by the specifics of the problem being solved and the quality of the rules applied.
Sub-heuristic Enhancements	These methods used in the core of search algorithms can improve search efficiency, convergence speed, and final solution quality for certain problems.	Sub-heuristics are not a well-researched area and the existing enhancements are specific to certain problems and not generalized. They might not account for certain transitions that could occur while building the solution.

2.2.4. Existing ACO methods to solve DOPs

Numerous studies have been conducted on the Ant Colony Optimization algorithm to solve problems of evolving nature [154], [155], [156]. In the classical Ant Colony Optimization algorithm, changing any aspect of the optimization problem leads to explored solutions becoming infeasible and or suboptimal [157]. In nature, real ants constantly face environmental changes such as new food sources appearing or being removed and pathway blockage. However, ants do not start a complete search from scratch. Usually, algorithms solving static optimization problems are finely tuned to converge quickly onto optimum solutions in the search static search space. However, such behaviour might not be desirable for dynamic optimization problems because fast and precise algorithms usually struggle to explore search space after the dynamic change. The challenge is to enable an artificial ant colony to explore new changes in the problem space efficiently [158]. There are several explored methods of Ant Colony Optimization in the dynamic environment.

In dynamic optimization literature, research usually tackles one of the two types of dynamic optimization problems. The first type is the optimization problems executed in the time domain, which has predictable dynamism patterns. These patterns of problem dynamics are solved as an additional dimension of variable edge cost in single goal optimization. This type of optimization problem does not require a special dynamic optimization algorithm but requires modification for the optimization problem definition. Typical applications of such predictive optimization are in routing problems like Vehicle Routing Problem (VRP) [159] [160], electric grid energy management [143], or scheduling problems [161]. The second type of DOPs is unpredictable event-triggered DOPs. When dynamic environment changes cannot be predicted, the problem must be solved again. Two major strategies for the ACO algorithm have been researched to solve the unpredictable event-triggered DOPs, Full-Restart and Pheromone-Share strategies.

2.2.4.1. Full-Restart strategy

The most common approach to solving dynamic optimization problems when changes are unpredictable is to restart the search entirely. When changes occur in search space, the whole optimization is started from the beginning, clearing out all information associated with the previous state. This approach is the simplest as it does not require explicit algorithm modification and relies on the algorithm to quickly converge to the new good solution [162]. The Full-Restart strategy does not share any information between optimization states, and each state is optimized independently. After the full restart algorithm usually converges in an identical pattern to the state before, as shown in Figure 2-3.

In the literature, Two ACO dynamic optimization strategies, Full-Restart and Pheromone-Sharing, have been compared by Angus & Hendtlass [157]. The conclusion was reached that for Dynamic Traveling Salesman Problem, the Pheromone-Sharing strategy has given a faster convergence rate to a good solution. Still, the Full-Restart strategy has allowed converging to a more optimal solution. Then, ACO algorithm Full-Restart strategy was used in a railway routing problem, and the solutions rebuild and deployed in real-time, allowing the algorithm to run for a limited amount of time, and deploy the best solution that the algorithm has found [163]. Furthermore, the ACO Full-Restart strategy was investigated on real-time dynamic optimization problems, where optimal schedules and routes are already in use and partially completed. The dynamic changes alter the remaining part of the solution, which requires to be reoptimized for problems like VRP [156], and Job-shop Scheduling Problem (JSP) [164].

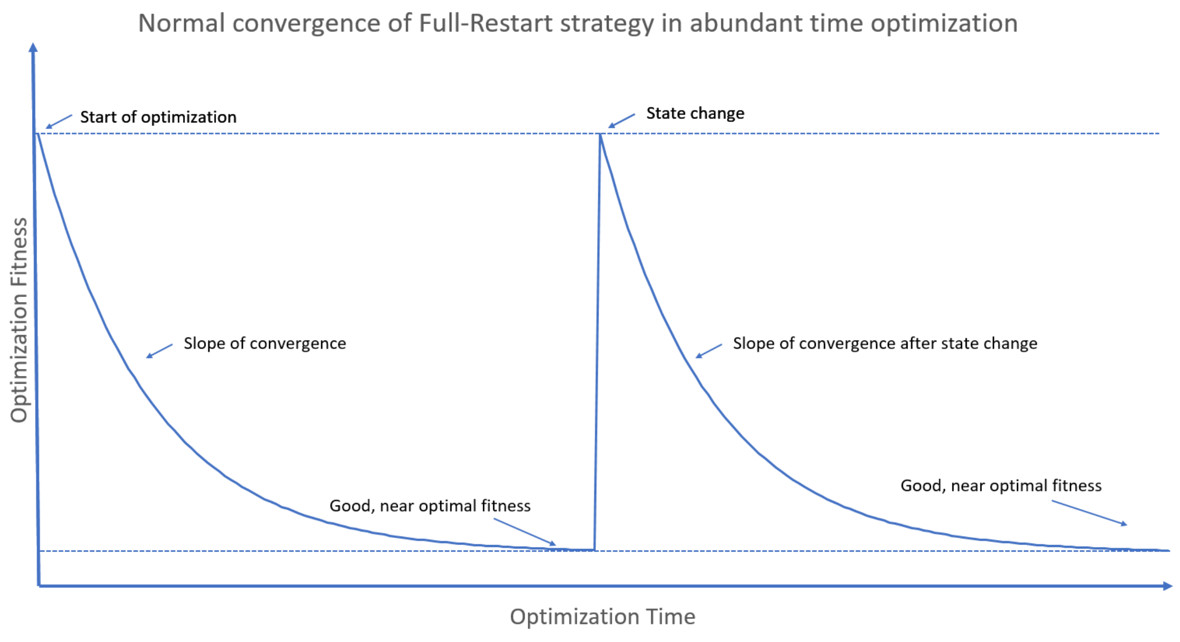

Figure 2-3: Normal Convergence of Full-Restart strategy in abundant time optimization. The chart displays a minimization problem's convergence for two dynamic states, where

the algorithm has plenty of time to converge to a "good" solution in both instances. Both initial and following states show identical convergence patterns.

2.2.4.2. Pheromone-Sharing strategy

When unpredictable dynamic change occurs in the environment, the search space changes too. However, changes in the nodes and the edges of the new search space are usually small enough and can map to the old search space. Therefore, the artificial ants can reuse a large part or the whole pheromone matrix from the previous optimization state. In cases where dynamic changes are significant to the extent that some parts of the search space do not map the pheromone matrix correctly, a heuristic fix can be applied to maximize the usefulness of the pheromone matrix from the previous state.

In literature, some ACO-based methods rebuild solutions on the existing pheromone matrix where new edges get applied to normalized pheromone level allowing ants to have a fair exploration in new search space [157]. Other development has tried several pheromone initialization strategies for new edges with the proposed Local random restart strategy, which initializes new edges with a random value, and the Local restart strategy, which initializes new edges with 0 pheromone value [155]. Approaches based on Population-ACO use the pheromone initialization process described by Guntsch & Middendorf [120], where an arbitrary number of elitist ant solutions create a pheromone matrix for every new iteration. Then Population-ACO based solutions can also be fixed using heuristic methods after the dynamic change, which give a good head start for the pheromone quality after the dynamic change [79], [165]. Generally, the Pheromone-Sharing strategy makes a trade-off, shown in Figure 2-4, to significantly reduce the fitness penalty after the dynamic change at the cost of reduced convergence speed due to the necessity to evaporate previous state dynamics.

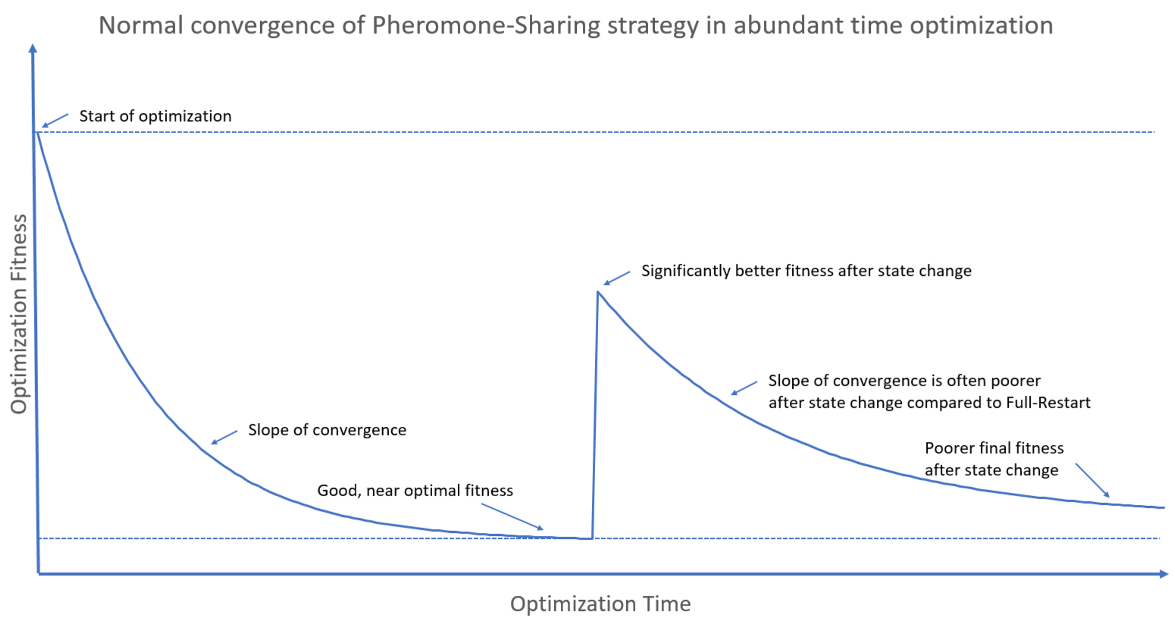

Figure 2-4: Normal Convergence of Pheromone-Sharing strategy in abundant time optimization. The chart displays a minimization problem's convergence for two dynamic states, where the algorithm has plenty of time to converge to a "good" solution in both instances. The initial state converges normally, and the following state starts at a significantly better fitness level but shows poorer convergence. Then poorer convergence leads to poorer final fitness results.

2.2.5. Need for discrete event-triggered dynamic optimization system

Both dynamic optimization strategies overviewed above have their specific advantages and disadvantages. The Pheromone-Sharing strategy significantly improves solution quality after the dynamic change because the pheromone matrix is reused for the new optimization state after the event triggers a dynamic change. When dynamic changes are triggered frequently, the Pheromone-Sharing strategy allows for gradual interstate convergence, which is better than the Full-Restart strategy, see Figure 2-6. However, some portion of the pheromone from the previous state must evaporate gradually, slowing the discovery of new paths. Therefore, the convergence of the Pheromone-Sharing strategy is usually slightly worse than the convergence if the search is restarted. On the other hand, the Full-Restart strategy performs good quality optimization when dynamic changes are triggered very infrequently, and the algorithm has plenty of time for convergence. However, when dynamic changes occur frequently, the Full-Restart strategy carries no information and restarts the search completely with poor results, see Figure 2-6.

Ideally, a purpose-built dynamic optimization method could combine the strengths of the Full-Restart strategy's convergence speed and the Pheromone-Sharing strategy's good solution quality after the dynamic change. This work introduces a purpose-built dynamic optimization strategy for the ACO algorithm called "ACO with Aphids". The ACO with Aphids is a nature-inspired dynamic optimization strategy that builds robust optimization without the drawbacks of poor convergence and poor restart after the dynamic change. In Figure 2-5, the ideal ACO with Aphids example is shown. In this example, after the dynamic change, fitness is equally good to the fitness of the Pheromone-Sharing strategy, but the convergence slope is as good as the Full-Restart strategy convergence.

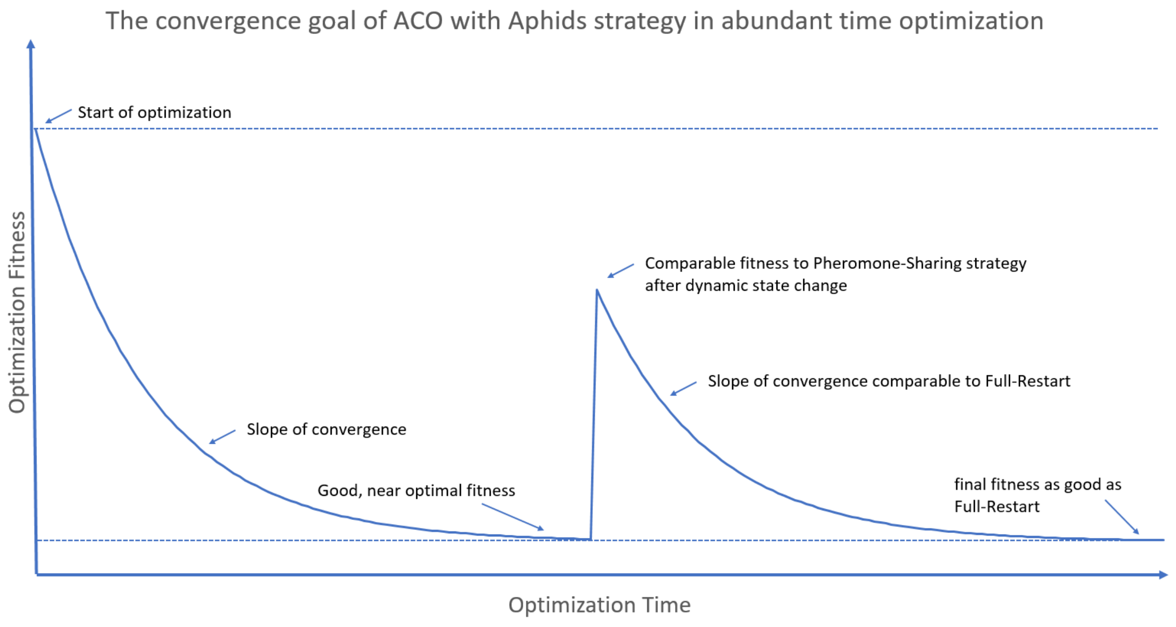

Figure 2-5: The convergence goal of ACO with Aphids strategy in abundant time optimization. The chart displays a minimization problem's convergence for two dynamic states, where the algorithm has plenty of time to converge to a "good" solution in both instances. The initial state converges normally, and the following state starts at a significantly better fitness level similar to the Pheromone-Sharing strategy and shows equally good convergence to the Full-Restart strategy. Then a good restart fitness after the dynamic change and a good convergence lead to even better results than the final result of the Full-Restart strategy.

A good dynamic optimization system should display the biggest solution quality improvements for frequently changing dynamic optimization. When the time is limited to perform the optimization of each state, at the start, fitness usually does not converge to an acceptable level, but the following states' optimization continues to improve the fitness further. ACO with Aphids strategy aims to intelligently reuse the information acquired during previous states' optimization such that after the dynamic change is triggered, the penalty to optimization fitness is minimal, and further convergence is unimpeded. This way, maximum interstate convergence equilibrium is possible, see Figure 2-6.

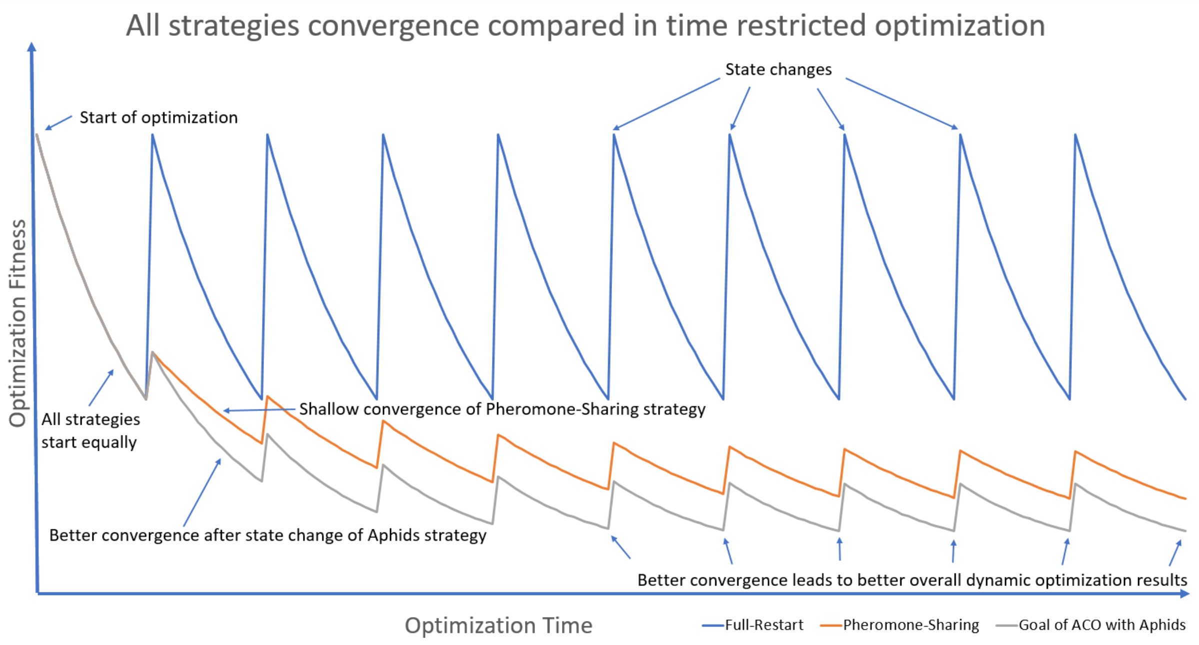

Figure 2-6: All strategies convergence compared in time-restricted optimization. The chart displays a minimization problem's convergence for ten dynamic states changing frequently. Aphids' strategy combines great optimization convergence observed in the Full-Restart Strategy with low state change fitness penalty observed in the Pheromone-Sharing strategy. The combination of these strengths allows for better interstate convergence.

2.2.6. Nature of Herder Ants

Some of the ant species, namely *Lasius niger* [166], care for and herd aphids. Aphids are tiny green bugs feeding plants and producing honeydew as waste. The honeydew is a sugar-rich liquid that is very nutritious to ants and acts as an additional food source. The relationship between ants and aphids is symbiotic. While ants feed on the aphids' wasted honeydew, ants also protect aphids from their natural predators [167]. The pheromone laid down by ants has a behavioural effect on aphids. In the ants' pheromone presence, aphids move slower and produce more honeydew [168]. There was also observed that ants may prey on aphids based on aphid density, honeydew production, and how much other ants tended to aphids [169].

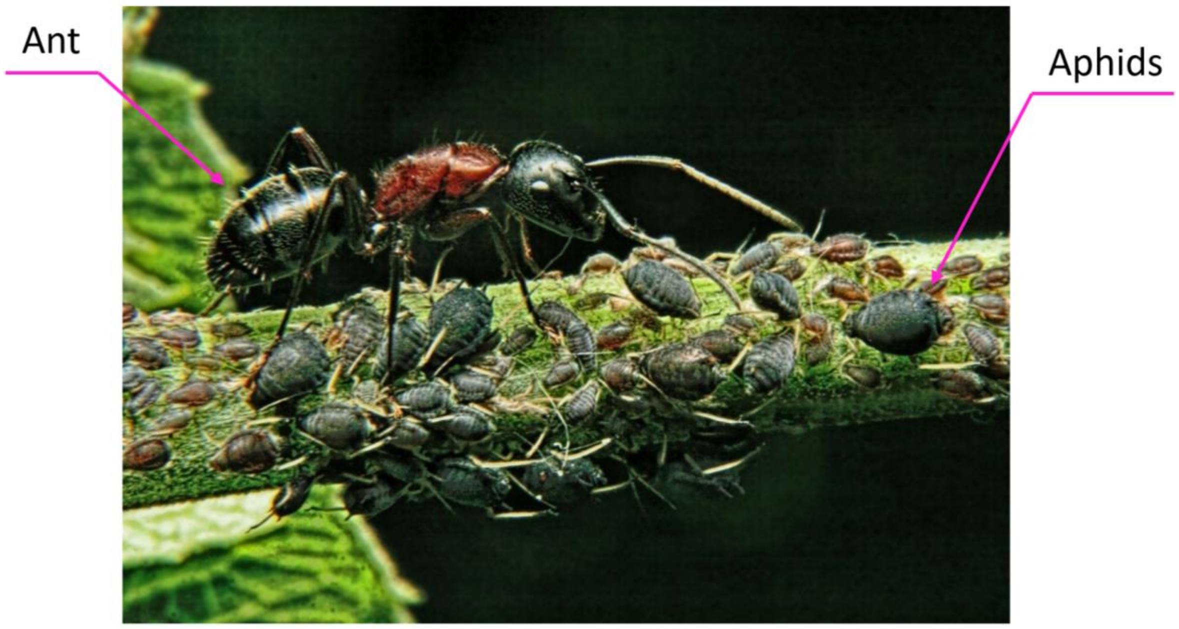

Figure 2-7: Close-up image of an ant guarding its aphids.

2.2.7. Use of aphids in other optimization algorithms

The novel use of the relationship between ants and aphids in optimization algorithms has seen promising developments. These include the Ant Colony Optimization with Cooperative Aphid, Cartesian Ant Programming, and Aphid-Ant Mutualism, each successfully addressing different optimization problems.

Aphid–Ant Mutualism (AAM) is a heterogeneous population-based algorithm that considers two types of individuals: ants and aphids [170]. In this algorithm, aphids perform a search in tandem with ants' search but with a different fitness function. Researchers have applied this algorithm to solve Multimodal Function Optimization benchmarks.

Cartesian Ant Programming (CAP) is a Cartesian Genetic Programming (CGP) algorithm that uses ants and aphids to optimize connections among function symbols [171]. In this algorithm, ants perform a search to find a valid set of connections and use the solution to deposit the pheromone on each connection to attract more ants. In addition to the pheromone, ants are also attracted to honeydew, which is deposited by aphids on every node.

Ant Colony Optimization with Cooperative Aphid (ACOCA) is a cooperative search method that combines the capabilities of an ant and aphid [172]. In this method, the ant and the aphid work together to give a solution. The information that the aphids provide is then treated as honey, and the search solutions of the ant are influenced by the honey. This method was tested to solve Traveling Salesman Problem (TSP).

So far, a symbiotic relationship between ants and aphids has not been attempted to model to benefit the discrete dynamic optimization. Aphids are tiny animals that ants tend to by placing them on plants, protecting them from predators, and collecting their waste honeydew. When too many aphids exist, a portion of them is killed to maintain the optimal aphid population. In simplistic terms, aphid’s honeydew production can represent the current state of the dynamic environment and use ants’ behaviour of tending, relocating, and killing aphids to optimize for maximum food supply from the original ant’s objective supplemented with honeydew supply from aphids.

2.3. Chapter Summary

This chapter has presented the general literature on dynamic optimization, Dynamic Optimization Problems with real-world applications, methods to solve them, and existing benchmarks to compare the performance of the dynamic optimization algorithms. Then this chapter presented the Ant Colony Optimization (ACO) algorithm in detail and showed how this algorithm had been applied to solve DOPs. Finally, this chapter presented two optimization problems in detail to use in experimental work.

In summary of this literature review, three main takeaways stand out as gaps in the knowledge and require further research.

Sub-heuristic methods have the potential to significantly increase ACO search efficiency to converge faster and converge to a better solution. For some optimization problems, additional heuristics within the ACO search core may yield significant search quality improvements. However, the methodology and applications are lacking in the literature review. Therefore, Chapter 3 presents a further investigation and a formal description of a new generalizable sub-heuristic method called Dynamic Impact. Also, ACO with Dynamic Impact is applied to MMFFPO and MKP problems. The advancement in the sub-heuristic methods of the ACO algorithm should improve the convergence within each state’s optimization.

So far, most theoretical research on dynamic optimization with comparable qualities has been done primarily on continuous domain dynamic optimization. The research on discrete domain dynamic optimization problems is either solving theoretical optimization problems by modifying datasets using stochastic methods or solving a real-world optimization problem. Optimization solutions of stochastically generated dynamic optimization datasets cannot be fairly compared because dataset results and optimal values will have some variance. Also, research on real-world optimization problems usually does not share the dataset used for the optimization. Therefore, Chapter 4 introduced a new non-stochastic dataset creation method and published fully defined Dynamic Multidimensional Knapsack Problem datasets.

Currently, ACO solving DOPs use one of two rudimentary dynamic optimization strategies that are easy to integrate into standard ACO algorithms, Full-Restart and Pheromone-Sharing. The standard ACO algorithm was not designed initially with dynamic optimization in mind. However, in nature ant species manifest behaviour that helps adapt to dynamic changes by herding aphids. Currently, no research is conducted on the relationship between ants and aphids for the benefit of effective discrete dynamic optimization techniques. Therefore, Chapter 5 proposed a new dynamic optimization strategy called ACO with Aphids to improve overall optimization performance by solving the DMKP benchmark and comparing it against Full-Restart and Pheromone-Sharing strategies. Dedicated ACO design for dynamic optimization should improve inter-state convergence. Those positive optimization performance gains will further compound with sub-heuristic ACO improvements.

Chapter 3. Dynamic Impact: a sub-heuristic method for ACO search

This chapter presents a design and experimentation on a sub-heuristic search method for the Ant Colony Optimization (ACO) algorithm called Dynamic Impact to address the limitations of the sub-heuristic search found in the previous chapter. The chapter also provides insight into how Dynamic Impact can be used for any constrained optimization problem. This method is then used to solve real-world Microchip Manufacturing Plant Production Floor Optimization (MMPPFO) problem and solve a theoretical Multidimensional Knapsack problem (MKP) for further validation and compare ACO with Dynamic Impact solution results to peer published work and their results.

3.1. Dynamic Impact methodology

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.

3.1.1. 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 use the lowest fraction of remaining resources. This method is the third component in an edge's probability calculation, along with pheromone and heuristic information. The Dynamic Impact method is also a myopic search component, improving search accuracy similar to the heuristic information.

Edge's probability calculation using Dynamic Impact:

$$p_{j,i} = \frac{\tau_{j,i}^{\alpha} * \eta_{j,i}^{\beta} * DI_{j,i}^{\gamma}(s_p)}{\sum\left(\tau_{j,i}^{\alpha} * \eta_{j,i}^{\beta} * DI_{j,i}^{\gamma}(s_p)\right)}, \quad \forall (j,i) \in N(s_p) \tag{3-1}$$

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, like the pheromone, it cannot be updated after each solution is completed.

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:

$$DI_e = \left(f(s_p + e) - f(s_p)\right)^{A} \times \left(\frac{\Omega(s_p + e)}{\Omega(s_p)}\right) \tag{3-2}$$

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 in fitness may not be the most beneficial if it takes a disproportionally large part of the remaining constraints. Depending on an optimisation problem, some parts of this Dynamic Impact function may be simplified. 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 [134], [173] 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.

3.1.2. 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 more fuel than the more direct route in city traffic which is also much slower. Referring to formula (3-2), 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.

$$DI_{Route} = \big(Time(Route)\big)^{-1} \times \frac{RemainingFuel - FuelCons(Route)}{RemainingFuel} = \frac{RemainingFuel - FuelCons(Route)}{RemainingFuel * Time(Route)} \tag{3-3}$$

In Table 3-1, 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. Third, fast with high fuel consumption. Three scenarios of remaining fuel are considered: low, medium, and high amounts of the 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. The pheromone and the 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.

Table 3-1: Simplistic example of Dynamic Impact. Three parallel scenarios are shown, which have three 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	**25**	**10**	**2.5**	**15**		**0.30**
1	2	30	15	2	25	60	0.29
	3	60	60	1	60		0.00
	1	25	10	2.5	15		0.33
2	**2**	**30**	**15**	**2**	**25**	80	**0.34**
	3	60	60	1	60		0.25
	1	25	10	2.5	15		0.35
3	2	30	15	2	25	120	0.40
	3	**60**	**60**	**1**	**60**		**0.50**

3.2. Applied optimization problems

In this chapter, two optimization problems are used to perform experimental work Multi-dimensional Knapsack Problem (MKP) and Microchip manufacturing plant production floor optimization (MMPPFO). This section serves as a comprehensive introduction to both optimization problems.

3.2.1. Multi-dimensional Knapsack Problem (MKP)

MKP is a well-known set covering academic benchmark optimization. This problem occurs in many different applications and is strongly NP-hard [174]. The MKP consists of a set of n items that have a profit $P_i > 0$ and m knapsacks with capacities $C_k > 0$. Each item i uses defined $W_{i,k} > 0$ amount of capacity from each knapsack k. MKP aims to find a set of items where the combined profit of those items is as high as possible while the combined weight fits in all knapsacks [175], [133]. The nature of packing different size items in all knapsacks simultaneously makes the feasible region of the search very sparse [176]. Such sparsity is a great challenge for optimization algorithms where good solutions are obtained by iterative convergence. The formal description of MKP is as follows:

$$maximize \sum_{i=1}^{n} x_i \times P_i \tag{3-5}$$

$$subject\ to\ \sum_{i=1}^{n}\left(x_i \times W_{i,k}\right) \leq C_k\,, \qquad \forall(k)\ where\ k \in (\mathbb{N} \leq m) \tag{3-6}$$

where n and m is a number of items and knapsacks in the problem. $x_i \in \{0,1\}$ is a decision vector to take the item I_i. P_i is the profit of the item I_i. $W_{i,k}$ is the weight of i^{th} item for the k^{th} knapsack. C_k is the capacity of the k^{th} knapsack.

The concept of items with multidimensional weights fitting into multiple knapsacks can be hard to imagine. This weight abstraction is much easier to understand with a simple, practical example of a backpack with a maximum weight limit of 16 kilograms and a maximum volume limit of 15 litres. All items have some weight and some volume. The goal is to pick a set of items with the highest profitability without exceeding both the weight and the volume of the backpack, see Figure 3-1.

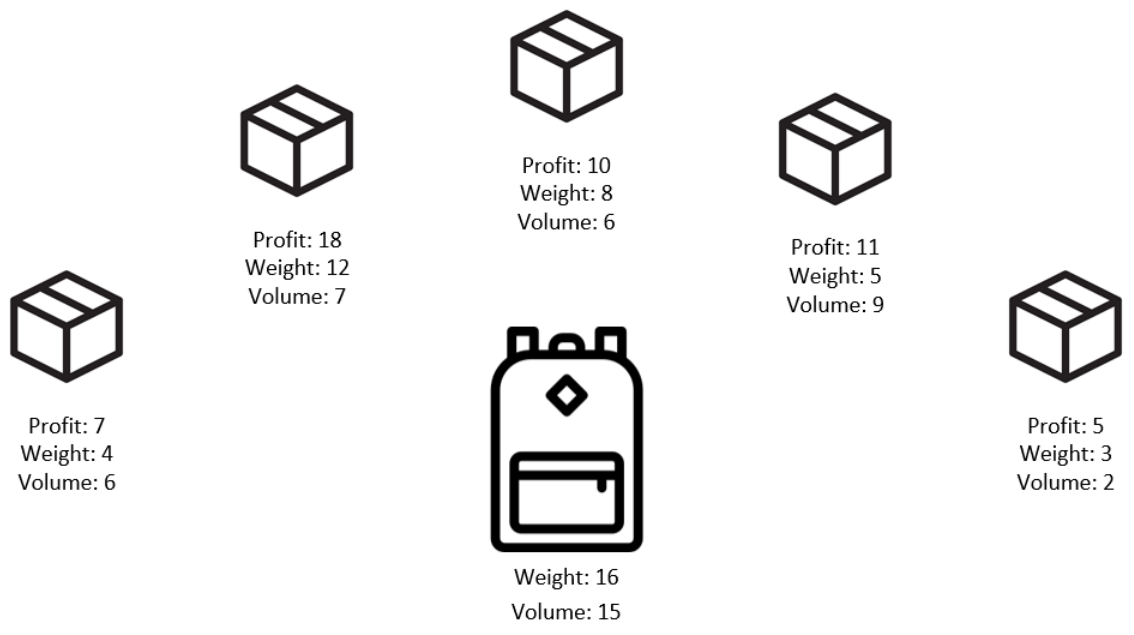

Figure 3-1: Simple example of Multidimensional Knapsack Problem with two packing dimensions of weight and volume

There are a lot of MKP benchmark datasets to solve and compare the results. Three most frequently used MKP benchmark libraries are SAC94 benchmark collection of 55 small MKP instances up to 105 items per dataset. GK collection introduced by Glover and Kochenbeger contains 11 much larger instances, up to 2500 items per dataset [177]. OR library introduced by Chu and Beasley has the most datasets, 270 instances of medium complexity up to 500 items [174]. The datasets can be found in this Research Gate repository [178].

Algorithms developed to solve MKP are incredibly useful in solving real-world optimization problems. Many practical optimization problems have an equivalent expression to MKP like cargo loading [38], layout problem [41], project budgeting [44], portfolio management [45], and cutting stock [46], to name a few.

Over the last several decades, many new improvements to optimization algorithms have been proposed while pursuing better results of static MKP. Genetic algorithm (GA) with sexual selection, where chromosomes selected for crossover must be the opposite gender, proved to solve consistently better than comparable algorithm without such method [179]. Improved GA with pattern substitution where bad genes are replaced with good using the greedy method [180]. Ant Colony Optimization (ACO) with adopted Min-Max Ant System (MMAS) has proved to be a competitive algorithm solving benchmark MKP [181]. Also, massively parallel approaches were developed for the ACO to improve result quality and reduce computation time using distributed cloud computing [182] and General-Purpose Graphics Processing Units (GPGPU) [183]. Set-based Particle Swarm Optimization (PSO) has been adopted to solve MKP, a discrete version of PSO, while PSO is typically used to solve optimization problems of continuous nature [184]. Then PSO further improved with added genetic crossover operation [185] and local search [186].

3.2.2. Microchip manufacturing plant production floor optimization (MMPPFO) problem

Microchip Manufacturing Plant Production Floor Optimization (MMPPFO) is a real-world optimization problem the industry partner faces. The source of the following problem description is agreed upon with the industry partner and is cleared for publication. The details include problem purpose, terminology, abbreviations, formulas, constraints, and optimization objectives. This information is sufficient to recreate and thoroughly verify the experimental work’s findings using the same dataset.

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 actual 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 [187], transport scheduling in automated material handling systems for wafer manufacturing plants [188], and wafer production as a job shop scheduling problem [189]. This research on the MMPPFO problem approaches optimization as a resource-constrained production scheduling problem.

The optimization problem starts with the initial wafer-lot production schedule and new die request. To solve the problem, the wafer-lots schedule must be altered to support all the changed and existing planned demands. 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, 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. The time window of this scheduling problem 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.

3.2.2.1. MMPPFO 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. Also, 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

$$SR(O_j) = \sum_i Q(WL_i), \qquad WL_i \in s_p \tag{3-7}$$

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.

$$USR(O_j) = D(O_j) - SR(O_j) \Big| \begin{matrix} if\ D(O_j) > SR(O_j) \\ otherwise\ 0 \end{matrix} \tag{3-8}$$

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.

$$OSR(O_j) = SR(O_j) - D(O_j) \Big|^{if\ D(O_j) < SR(O_j)}_{otherwise\ 0} \tag{3-9}$$

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.

$$U(C_{P,F,W}) = \sum_i Q(WL_i), \qquad WL_i \in s_p \tag{3-10}$$

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.

$$WA(C_{P,F,W}) = C_{P,F,W} - U(C_{P,F,W}) \tag{3-11}$$

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.

3.2.2.2. Constraints

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

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.

$$C_{P,F,W} > U\left(C_{P,F,W}\right) \tag{3-12}$$

Order support constraint

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

$$W(WL_i) \leq W\left(O_j\right), \quad \forall (j,i) \in N \tag{3-13}$$

where $W(WL_i)$ is commit week of WL_i, $W\left(O_j\right)$ 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. If necessary, push-out can be done only for a corresponding pull-in operation 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 the product itself. This limits the search space by not including wafer-lot permutations of offload to fabs that cannot produce the wafer-lot product.

3.2.2.3. *Optimization objectives*

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.

Minimum undersupported request

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.

$$min \sum_{j} USR(O_j) \tag{3-14}$$

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 a 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 demands.

3.3. ACO with Dynamic Impact algorithm performance investigation

In this section, two experiments have been conducted. First, solving the real-world MMPPFO, and second, solving the theoretical 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 analysed.

3.3.1. ACO solving Microchip Manufacturing Plant Production Floor Optimization

3.3.1.1. 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.

3.3.1.2. *Heuristic information*

Ant Colony Optimization uses heuristic information that plays a crucial role in the algorithm's convergence [116]. 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:

$$\eta_{j,i} = \frac{O_j}{Q(WL_i)} \tag{3-15}$$

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.

3.3.1.3. *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 [190].

3.3.1.4. *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 (3-2) 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 the maximization objective, and therefore fitness impact term of the Dynamic Impact formula is adjusted as follows:

$$\left(f(s_p + e) - f(s_p)\right)^A \rightarrow f(s_p) - f(s_p + e) \tag{3-16}$$

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

$$f(s_p + e) := max\{RD(O_j) - Q(WL_i), 0\} \tag{3-17}$$

$$f(s_p) := RD(O_j) \tag{3-18}$$

$$RD(O_j) = D(O_j) - SR(O_j) \tag{3-19}$$

$$f(s_p) - f(s_p + e) = RD(O_j) - max\{RD(O_j) - Q(WL_i), 0\} \tag{3-20}$$

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:

$$\Omega(s_p + e) := RC(C_{P,F,W}) - Q(WL_i) \tag{3-21}$$

$$\Omega(s_p) := RC(C_{P,F,W}) \tag{3-22}$$

$$RC(C_{P,F,W}) = C_{P,F,W} - U(C_{P,F,W}) \tag{3-23}$$

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 note that the 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 constraint 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:

$$\frac{\Omega(s_p + e)}{\Omega(s_p)} := max\{Q(WL_i) - RD(O_j), 0\} \tag{3-24}$$

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

$$f(s_p) - f(s_p + e) + \frac{\Omega(s_p + e)}{\Omega(s_p)} = RD(O_j) - max\{RD(O_j) - Q(WL_i), 0\} + max\{Q(WL_i) - RD(O_j), 0\} = RD(O_j) - |RD(O_j) - Q(WL_i)| \tag{3-25}$$

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:

$$DI_{j,i} = max\,\{RD(O_j) - |RD(O_j) - Q(WL_i)|, 0.1\} \tag{3-26}$$

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.

3.3.2. 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 the Min-Max Ant System have been established by preliminary experimentation. The best combination of pheromone parameters is: $\tau_{max} = 1$, $\tau_{min} = 0.001$, $\rho = 0.1$, $\Delta\tau_0 = 1$. Configuration of probability parameters: $\alpha = 1$, $\beta = 0$. Solutions are achieved by running 3,000 iterations using 2 sequential ants, using 16 parallel ants as per Dzalbs et al. described architectural model [39].

In Table 3-2, 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 request score. When using higher levels of Dynamic Impact importance, ACO algorithm tends to perform better using lower $q0$ values, which means a preference towards exploration over exploitation. 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 an imperfectly tuned baseline, the average of 5 best $q0$ settings at $\gamma = 0$ is 35.3 wafers of undersupported request.

Table 3-2: Undersupported result map for γ and q0, where γ = 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

Moreover, in Figure 3-2, 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 Figure 3-2, 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.

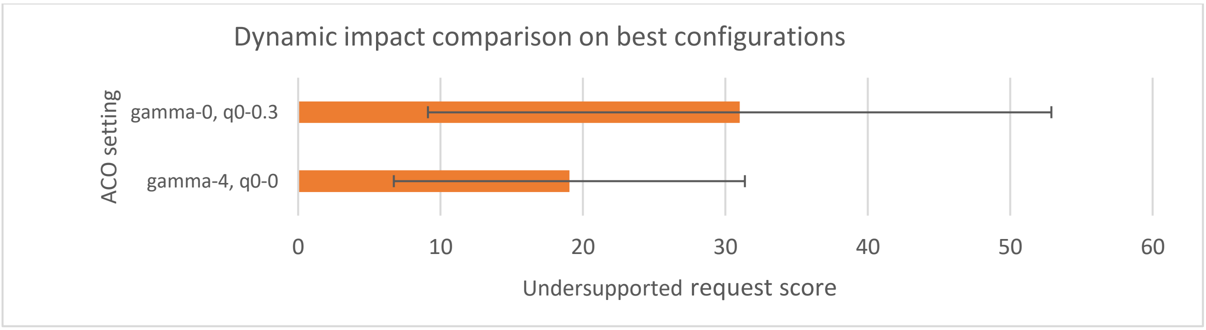

Figure 3-2: Dynamic Impact comparison on best configurations. Average of 50 runs. Error bars indicate one standard deviation.

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 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, on average, runs 86.8 seconds. Using the best configuration algorithm, ACO with Dynamic Impact took, on average, 96.9 seconds. Dynamic Impact evaluation is, on average, 11.6% more expensive to compute each iteration. Such a small addition in computational complexity was possible due to the simplified wafer-lot impact on solution fitness value, compared to an unoptimized version which could be several times more 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.

3.3.3. ACO solving Multidimensional Knapsack Problem (MKP)

In addition to solving MMPPFO, ACO to solve MKP has also been implemented. 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. The MKP is a suitable problem to test Dynamic Impact because the nature of the optimization goal is to find the collection of items of the highest profit that fit in the knapsack. There is no preference over which items should be taken as long as they all fit in the knapsack and ideally with the highest total profit. Such fungibility of items makes it a good candidate problem to benefit from the Dynamic Impact evaluation method.

3.3.3.1. 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.

3.3.3.2. 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:

$$\eta_i = \frac{P(I_i)}{W(I_i)} \tag{3-27}$$

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 was not 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.

3.3.3.3. Experimental dataset

The datasets are obtained from the ResearchGate repository [178]. 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.

3.3.3.4. Dynamic Impact for MKP optimization

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

$$DI_i = \frac{NP(I_i)}{CI(I_i)} \tag{3-28}$$

$$CI(I_i) = max_{\forall j}\left\{\frac{W(I_i)}{RC(K_j)}\right\} + \frac{\sum_j \left\{\frac{W(I_i)}{RC(K_j)}\right\}}{j} \tag{3-29}$$

$$NP(I_i) = \frac{P(I_i)}{max\{\forall P(I)\}} \tag{3-30}$$

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 the 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 the remaining 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.

The dynamic Impact formula for MKP is obtained in relation to the general Dynamic Impact formula (3-2). The formula component values are as follows: 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:

$$DI_i = \frac{f(e)}{\left(\frac{\Omega(e)}{\Omega(s_p)}\right)} \qquad (3\text{-}31)$$

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.

3.3.4. 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 benchmark MKP datasets are considered in this experiment. The first set of 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 analysed and compared to which algorithm, on average, reaches the optimal solution quicker. The second set is large GK benchmark datasets. These benchmark datasets' combinatorial complexity is high enough that not all GK datasets have known optimal values. Therefore, in Table 3-5 for comparison, the most recent best-known values will be taken from [191] that combines their own reached highest values as well as [192] and authors of the GK datasets [177]. 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.

3.3.4.1. SAC94 results

For the SAC94 experiment, Min-Max Ant System parameters have been tuned with preliminary experimentation. The best combination of pheromone parameters is:

$\tau_{max} = 1$, $\tau_{min} = 0.001$, $\rho = 0.1$. 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 [39] 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 3-3 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 the 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 3-3: 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 3-4: SAC94 results comparison with recently published research.

Dataset	Problem size (N x M)	Optimal	ACO without Dynamic Impact	ACO with Dynamic Impact	BPSOTVAC - [193] 2014	DBDE - [194] 2017	MFPA - [195] 2018	HPSOGO - [196] 2018	TR-BDS - [197] 2016	BAAA - [198] 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

Also, the results of SAC94 are compared to recently published research on the state-of-the-art optimization algorithms solving SAC94 datasets in Table 3-4. A Binary PSO with Time-Varying Acceleration Coefficients (BPSOTVAC) proposed by Chih et al. [193]. A Dichotomous Binary Differential Evolution (DBDE) proposed by Peng et al. [194]. A Modified version of the Flower Pollination Algorithm (MFPA) proposed by Abdel-Basset et al. [195]. A Binary Particle Swarm Optimization with Genetic Operations (HPSOGO) introduced by Mingo López et al. [196]. A Random Binary Differential Search algorithm using the Tanh function (TR-BDS) introduced by Liu et al. [197]. A Binary Artificial Algae Algorithm (BAAA) introduced by Zhang et al. [198].

The primary comparison metric of all results is the success rate. In this example, 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 essential to note that this research paper [195] is inconclusive and does not have complete SAC94 dataset results. Hence, the algorithm's versatility is not proven since the success rate is unknown for the remaining datasets. Secondly, BAAA has a 95.2% 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 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.

3.3.4.2. 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. The experiment measures the average profit obtained over ten algorithm runs. Then the average profit is turned into the average gap using the best-known profit values. In Figure 3-3, 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 the lowest is in gk01 – 0.27%. In relative terms, the difference in the profit gap is, on average, 4.26 times lower, where the highest is gk02, reducing the gap 10.4 times, and the lowest is gk03, reducing the gap by 2.33 times.

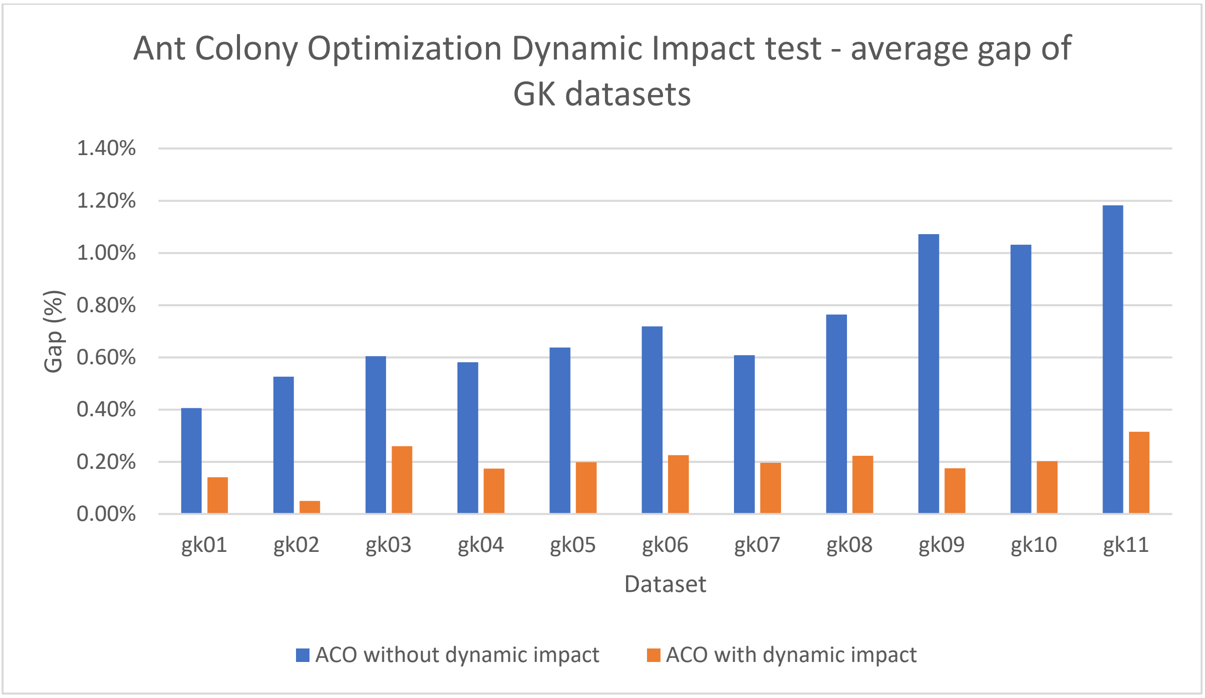

Figure 3-3: ACO Dynamic Impact test - GK dataset results graph of the average gap. Results are an average of 10 algorithm runs.

Furthermore, in Figure 3-4 well-performing ACO with Dynamic Impact algorithm is stacked up against recently published solutions of GK dataset implementations. Dantas – GPGPU SA [199] is GPU accelerated Simulated Annealing algorithm. Kong – NBHS2 [200] out of several algorithms compared their proposed New Binary Harmony Search type 2 was best performing for GK datasets. Wang – DLHO [201] 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.

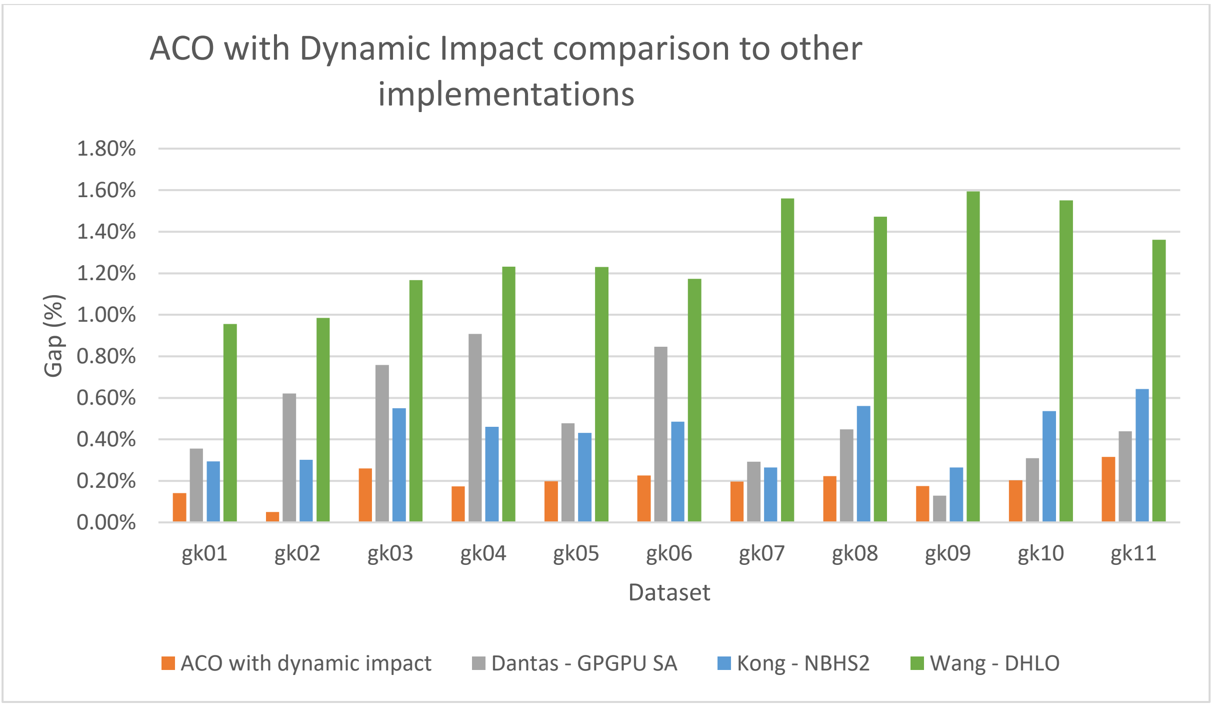

Figure 3-4: ACO with Dynamic Impact comparison to other recently published GK dataset solution results.

Table 3-5: 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 [199] 2018	Kong-NBHS2 [200] 2015	Wang-DHLO [201] 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%

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.

3.4. Chapter Summary

This research chapter has studied the Ant Colony Optimization sub-heuristic methods to improve algorithm search convergence. The research has proposed an additional component to the ACO algorithm probability calculation, which is called Dynamic Impact. This method improved the convergence of optimization problems, where the main optimization objective depends on a collection of smaller parts where each part does not have any priority over another. Dynamic Impact, similarly to heuristic information, is a myopic component of the search. The difference is that 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 function and resource consumption. The computational overhead of using 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 performance 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.

Finally, the benefit of using Dynamic Impact can be exploited while performing the iterative search. Dynamic Impact can also increase the efficiency of dynamic optimization. Therefore, The ACO with Dynamic Impact algorithm will be used as a baseline to develop a new nature-inspired dynamic optimization strategy in Chapter 5.

Contributions of this chapter to science are as follows:

- Introduction of Dynamic Impact a sub-heuristic method for the Ant Colony Optimization algorithm that helps more accurately calculate the edge's probability while considering the remaining constrained resources of the problem and non-linear fitness.

- Description of the methodology to apply Dynamic Impact effectively for a broad range of academic and real-world problems.
- Proof that Dynamic Impact is beneficial for finding better results using the same computational efforts in theoretical MKP and real-world MMPPFO problems.
- Also, shows that ACO with Dynamic Impact is superior in finding better solutions for MKP benchmark instances than previous peer research.

The work presented in this chapter has been published in the peer-reviewed journal Swarm and Evolutionary Computation, Elsevier [1].

Chapter 4. Dynamic MKP Benchmark methodology

Before dynamic optimization can be tackled in this thesis, the issue of the lack of replicable qualities in discrete dynamic optimization studies must be resolved. This research chapter introduces a non-stochastic dataset generation methodology applicable for discrete optimization problems. The dataset generation methodology relies on an existing static benchmark dataset as an initial state to generate a dynamic dataset with the desired number of states where each state is an evolution from the previous state. The state's creation is done in a non-stochastic way where generated state always is the same for the constant input state and depends only on the input state and datasets constraints defined in the initial state. Such generated dynamic dataset is a collection of sequential states of a static dataset. The evolution of these states is in a predictable and repeatable way such that one generated dynamic dataset could be further extended with more states if needed. This research is vital because fully defined datasets will be compatible with most dynamic optimization algorithms. No special environments are needed to use the dataset generators, and dataset generators do not need to rely on random operator seeds, which could be easily overlooked. Then such dynamic optimization algorithm results can be independently verified for result validity and directly compared with results obtained from other dynamic optimization algorithms.

4.1. Dynamic MKP Datasets

4.1.1. Dynamic Multidimensional Knapsack Problem

The multidimensional Knapsack problem is widely used for benchmarking combinatorial optimization algorithms [202]. Solutions to the MKP problem have numerous applications in the real world, such as loading cargo optimization, slicing problem, budget management, and investment portfolio management problem. Therefore, there is a lot of interest in developing algorithms to solve the MKP problem. Then, DMKP has also attracted some research community attention due to the nature of the problem that can be easily extended into dynamic variant using static datasets as the initial setting.

A dynamic variant of MKP, the Dynamic Multi-dimensional Knapsack Problem (DMKP) is also an academic problem that uniquely benefits large-scale real-world optimization problems with some degree of dynamism. The DMKP is a dynamic combinatorial optimization problem, and it is formulated as a sequential series of static MKP instances called states. Between sequential states, the numerical difference of each item's profit, item weights, and knapsack capacities should be reasonably small, indicative of problem dynamism that occurs in the real world. The DMKP problem aims to maximize the total profit of each state before a dynamic change occurs. The result of DMKP is the sum of each state's maximum profit.

$$maximize \sum_{s=0}^{S_{max}} \left(\sum_{i=1}^{n} x_{s,i} \times P_{s,i} \right) \tag{4-1}$$

$$subject\ to \sum_{i=1}^{n} \left(x_{s,i} \times W_{s,i,k} \right) \leq C_{s,k}, \tag{4-2}$$

$$\forall (s,k)\ where\ k \in (\mathbb{N} \leq m), \qquad s \in (0\ \leq \mathbb{N} \leq S_{max})$$

where s is the index of the state, and S_{max} is a number of states in the DMKP problem. The equations state maximize the profit of the items in the decision vector in every DMKP state, subject to item weights not exceeding all corresponding knapsack capacities in every state.

This DMKP benchmark is a good test suite for testing and comparing the performance of combinatorial optimization algorithms in a dynamic environment like Genetic Algorithm (GA) [203], Particle Swarm Optimization (PSO) [204], Firefly Algorithm (FA) [205], Monarch Butterfly Optimization (MBO) [206], Cuckoo Search (CS) [207], Artificial Bee Colony (ABC) [208], Moth Search (MS) [209], Slime Mould Algorithm (SMA) [210].

The DMKP can have one or multiple aspects of the dataset to be dynamic. In this study, the profit of items, item weights, and knapsack capacities are set to vary in discrete state intervals. The states are noted as $S_t,\ t \in \{0,1,2,\dots\}$, where S_0 is the initial state of the MKP dataset, and t is the state index. Each dynamic MKP state can be solved individually as a static MKP instance.

All existing attempts to solve DMKP have involved using static MKP benchmarks as the initial setting and introducing stochastic changes to some aspects of the optimization problem and process over the time domain. [97] uses normally distributed random operator to change item profits, item weights and knapsack capacities on OR library initial datasets. Meanwhile, the research [211] has dynamic changes in item profits, item weights and knapsack capacities, and all new randomly generated items.

4.1.2. Deterministic Dynamic MKP dataset creation methodology

The approach of the deterministic state generation method is designed to use the static instances of the existing benchmark MKP dataset as its initial state S_0. Then use the information from the initial dataset to create states in sequential order. The dynamic dataset is created using a deterministic set of formulas. The deterministic approach is essential to have the dataset reproducible. Using a stochastic method would make the research reproducibility and extension more difficult.

In this research, the item profits, item weights, and knapsack capacities are adjusted while generating a state. The new state's adjustment factors are determined from the values in the previous state and the constraints set by the initial state. The state generation method has a "State Adjustment Magnitude" Δ parameter to control the difference in the profit, weight, and capacity value differences between the states. This parameter is a constant for the entire dynamic dataset generation. The default value is 0.05 or 5% of the allowable adjustment range. This parameter ensures that the following states are reasonably similar to previous states, and none of the values has been modified more than the upper limit of the dataset value range.

For the purposes of creating a deterministic state generation that is reproducible yet chaotic, based on the information only taken from the previous state, the "3 value modifier" operator $X3V(v1, v2, v3)$ is introduced. This operator takes three values $v1, v2, v3$ and calculates them to one real value between -1 and 1:

$$X3V(v1, v2, v3) = (H3(v1, v2 * 2, v3 * 5) * 2 - 1)^3 \tag{4-3}$$

$$H3(v1, v2, v3) = frac(M(v1) * M(v2) * M(v3) + M(v1) + M(v2) + M(v3)) \tag{4-4}$$

$$frac(x) = x - \lfloor x \rfloor \tag{4-5}$$

$$M(x) = \frac{x}{10^{\lfloor \log_{10} x \rfloor}} \tag{4-6}$$

where $M(x)$ is a mantissa of the number x, $frac(x)$ is a fractional part of a number, $\lfloor x \rfloor$ is a rounded down number x, $H3$ is a simple numerical hash of three numbers $(v1, v2, v3)$ that returns a number between 0 and 1 evenly distributed, and finally, $X3V$ is a "3 value modifier", a normalized value between -1 and 1 and has probability density concentrated around 0. The property of having probability density concentrated around 0 in $X3V$ gives small adjustment values for the majority of the dataset with a few larger adjustments, which resembles in real-world optimization, the majority of minor operational adjustments and a few more significant disruptions.

The state of the dynamic dataset is created first, calculating the state's new item profits, then new item weights, and lastly, new knapsack capacities. The dynamic dataset generation method is designed to preserve each item's value within the range initial state's S_0 value range, which is an intrinsic item's property easily expressed as profit over average weight. Also, the new state's profit and weight values cannot cross their constraint boundaries. Then lastly, knapsack capacities are recalculated to keep the same tightness as the initial state's S_0 tightness. Following is the list of constraints that the new state must maintain:

- Minimum Profit $\min P = \min\limits_i P_{0,i}$
- Maximum Profit $\max P = \max\limits_i P_{0,i}$
- Minimum Value $\min V = \min\limits_i \frac{P_{0,i}}{\overline{W_{0,i}}}$
- Maximum Value $\max V = \max\limits_i \frac{P_{0,i}}{\overline{W_{0,i}}}$
- Minimum Weight $\min W = \min\limits_{i,k} W_{0,i,k}$
- Maximum Weight $\max W = \max\limits_{i,k} W_{0,i,k}$
- Knapsack Tightness $T_k = \frac{\sum_{i=1}^{m} W_{0,i,k}}{C_{0,k}} \; \forall \, k$

where $P_{0,i}$ is the profit of the i^{th} item in the initial state S_0, $\overline{W_{0,i}}$ is the average weight of the i^{th} item in the initial state S_0, $W_{0,i,k}$ is the weight of the i^{th} item for the k^{th} knapsack in the initial state S_0, and finally $C_{0,k}$ is the knapsack capacity of the k^{th} knapsack in the initial state S_0.

New states can then be generated using these calculated constraints from the original dataset and the current state data. The state generation method order is strictly sequential, where the new state depends only on the most recent predecessor. To make the explanation easily understandable, the process of creating a state involves a current state which is noted as S_t used as input in the state generation and a successor new state which is noted as S_{t+1}. Each state S_t has an independent set of item profits $P_{t,i}$, item weights $W_{t,i,k}$, and knapsack capacities $C_{t,k}$, where t notes the state, i notes the item, and k notes the knapsack of the dataset.

Furthermore, for the state generation, the adjustment limits have to be set in accordance with original constraints and the State Adjustment Magnitude $\varDelta$ parameter. The $\varDelta$ parameter is also called SAM in the code and charts with no special characters' support.

- Profit adjustment magnitude

$$\varDelta P = \varDelta * (\max P - \min P) \tag{4-7}$$

- Weight adjustment magnitude

$$\varDelta W = \varDelta * (\max W - \min W) \tag{4-8}$$

Profit generation is the first step of creating a new state S_{t+1}. For each item, profit $P_{t+1,i}$ the procedure uses the profits of 3 items to calculate profit modifier using $X3V$ operator and original constraints to calculate chaotic profit adjustment within the limits of the dataset characteristics.

$$P_{t+1,i} = Max\left(Min\left(P_{t,i} + Px\ , maxV * \overline{W_{t,i}}\right), minV * \overline{W_{t,i}}\right) \tag{4-9}$$

$$Px = X3V\left(P_{t,i-1}, P_{t,i}, P_{t,i+1}\right) * \varDelta P + Pxc \tag{4-10}$$

$$Pxc = Min\left(maxP - P_{t,i}, \varDelta P\right) - Max\left(P_{t,i} - minP, \varDelta P\right) \tag{4-11}$$

where, $P_{t+1,i}$ is a new item profit for the state S_{t+1} of the i^{th} item that is applied for all items $\forall i$. This new profit is calculated using the current state's profit $P_{t,i}$ and profit adjustment value Px, then it is constrained within a minimum and a maximum allowed item profit, which is a product of the item's average weight and original value: $maxV * \overline{W_{t,\iota}}$ and $minV * \overline{W_{t,\iota}}$. The profit adjustment Px value is calculated using the profit adjustment multiplier $\varDelta P$ multiplied by $X3V$ operator values taken from the current item's profit $P_{t,i}$ and two adjacent item profits $P_{t,i-1}, P_{t,i+1}$, then added profit adjustment correction Pxc. Pxc is a value that maintains the profit within initial dataset constraints but allows free manipulation when profit $P_{t,i}$ is within the profit range by at least a value of ΔP, in those cases $Pxc = 0$.

After profits are complete, the new state's S_{t+1} item weights are generated. For each item's weight $W_{t+1,i,k}$ the procedure uses weights of three items to calculate weight modifier using $X3V$ operator and original weight and value constraints to create a chaotic weight modifier within the limits of the dataset characteristics.

$$W_{t+1,i,k} = Max\left(Min\left(W_{t,i,k} + Wx\,, \frac{P_{t,i}}{minV} - \overline{W_{t,i}}\right), \overline{W_{t,i}} - \frac{P_{t,i}}{minV}\right) \tag{4-12}$$

$$Wx = \left(X3V\left(W_{t,i-1,k}, W_{t,i,k}, W_{t,i+1,k}\right) * \ \Delta W + Wxc\right) \tag{4-13}$$

$$Wxc = Min\left(maxW - W_{t,i,k}, \Delta W\right) - Max\left(W_{t,i,k} - minW, \Delta W\right) \tag{4-14}$$

where, $W_{t+1,i,k}$is new item weight for state S_{t+1} of the i^{th} item that is applied for all items i and all knapsacks k. In principle, the generation of weights is similar to the generation of profits, except that it is also executed for all knapsacks. New item weight is calculated using the weight of the current state $W_{t,i,k}$ added with weight adjustment Wx. This value is constrained between $\frac{P_{t,i}}{maxV} - \overline{W_{t,\iota}}$ and $\overline{W_{t,\iota}} - \frac{P_{t,i}}{minV}$ such that as a result of the new weight, the item's value does not exceed the initial dataset's limits. $\frac{P_{t,i}}{maxV} - \overline{W_{t,\iota}}$ is items profit over the maximum value that gives minimum weight, and removing average weight gives maximum allowed weight increase to the item. Similarly, $\overline{W_{t,\iota}} - \frac{P_{t,i}}{minV}$ gives maximum allowed weight decrease. The Wx weight adjustment is calculated using $X3V$ operator with weight values of 3 adjacent items of the same knapsack and is multiplied with the weight adjustment magnitude ΔW and added weight adjustment correction Wxc value. Wxc is a similar value to Pxc that it ensures each new weight is within dataset limits but does not restrict the adjustment.

And finally, the knapsack capacities are calculated for the state S_{t+1}. This is the simplest calculation of them all. It uses the initial state's knapsack tightness values and the current state's item weights to create new knapsack capacities to maintain the same tightness as the initial state.

$$C_{t+1,k} = \sum_{i=1}^{n} W_{t+1,i,k} * \ T_k, \ \ \forall k \tag{4-15}$$

where, $C_{t+1,k}$ is the new state's capacity of the k knapsack and is a sum of all item weights for that knapsack multiplied by initial knapsack tightness T_k of the k^{th} knapsack.

A simplified example of dataset state creation is shown in Figure 4-1 below. For a given input dataset state S_t, new item profit values $P_{t+1,i}$ are calculated for all items i. Each item's profit calculation uses the profit values of three items from the input dataset $P_{t,i-1}$, $P_{t,i}$, $P_{t,i+1}$. Then $X3V$ operator and Px functions are applied on selected inputs. Then constrained profit values are exported. Similarly, new item weight values $W_{t+1,i,k}$ are calculated for all items i and all knapsacks k. Also, each item's weights calculation uses weight values of three items for the same knapsack from the input dataset $W_{t,i-1,k}$, $W_{t,i,k}$, $W_{t,i+1,k}$. Then $X3V$ and Wx functions are applied on those input weights, and constrained item weight values are exported. Finally, new knapsack capacities $C_{t+1,k}$ are calculated for all knapsacks k. Each knapsack capacity calculation uses all newly calculated item weights $W_{t+1,i,k}$ for a knapsack k. Then all those weights are summed up and multiplied by the original knapsack tightness. Then final knapsack capacity values are exported.

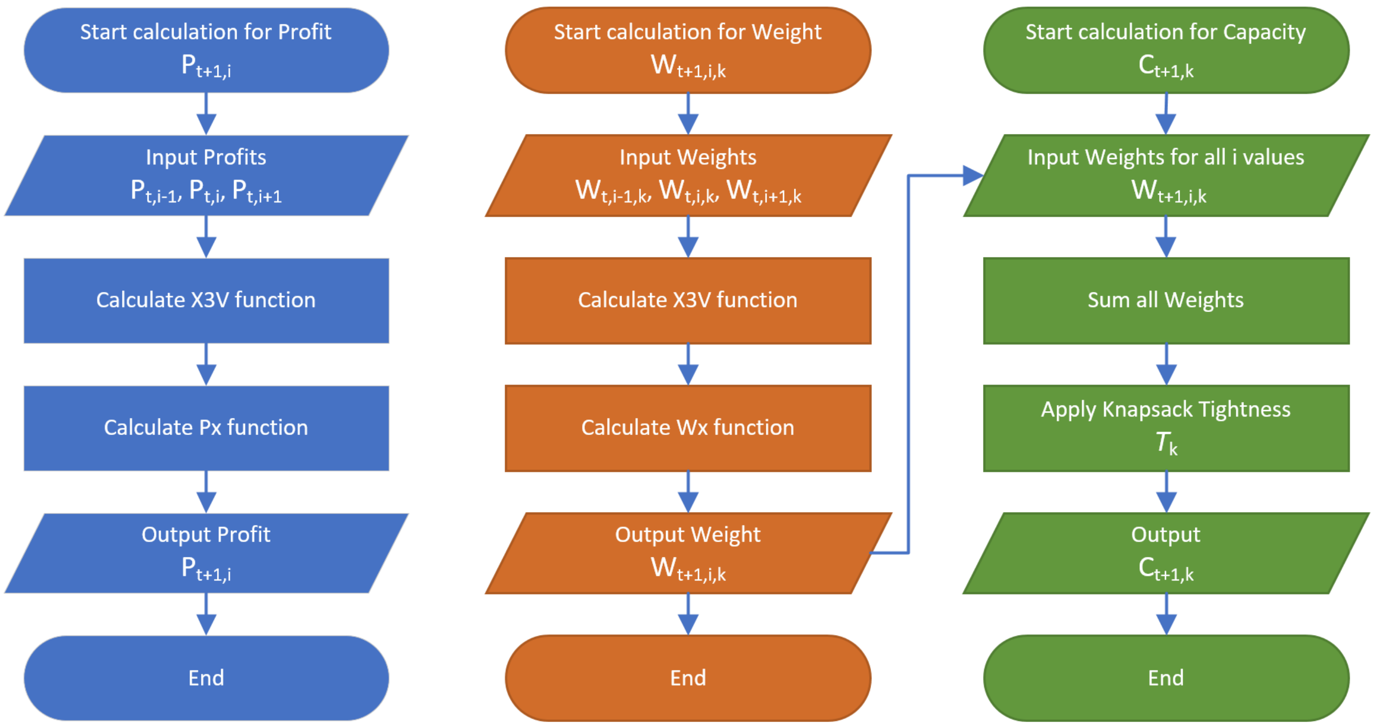
Start calculation for Profit $P_{t+1,i}$
Input Profits $P_{t,i-1}$, $P_{t,i}$, $P_{t,i+1}$
Calculate X3V function
Calculate Px function
Output Profit $P_{t+1,i}$
End
Start calculation for Weight $W_{t+1,i,k}$
Input Weights $W_{t,i-1,k}$, $W_{t,i,k}$, $W_{t,i+1,k}$
Calculate X3V function
Calculate Wx function
Output Weight $W_{t+1,i,k}$
End
Start calculation for Capacity $C_{t+1,k}$
Input Weights for all i values $W_{t+1,i,k}$
Sum all Weights
Apply Knapsack Tightness T_k
Output $C_{t+1,k}$
End

Figure 4-1: Dataset state creation flowchart of Item Profits (Blue), Item Weights (Orange), and Knapsack Capacities (Green). The flowchart shows the key dependencies of each value adjustment in the generated state.

4.1.3. Created dataset instances

Dynamic Multidimensional Knapsack Problem datasets are created using already existing benchmark datasets as a basis of dataset generation. The original benchmark datasets are taken from the ResearchGate repository [178]. For the purpose of this research, OR and GK datasets are used to create dynamic datasets, while SAC94 datasets are omitted due to low complexity and inconsistent sparseness.

Dataset deterministic state generation method requires input $\mathit{\Delta}$ SAM that sets the difficulty to generate the next state. This difficulty magnitude limits the percentage change applied for each adjusted value when generating a new state. If a value is too high, the next state can appear nothing like the previous state. If the value is too low, states might not differ at all due to the nature of integer numbers. Since $\mathit{\Delta}$ SAM is the maximum adjustment that will occur, it is recommended that minimum item weight $minW$ and minimum item profit $minP$ multiplied by $\mathit{\Delta}$ is more than 10. This number is chosen based on a reasonable probability that the item profit and weight adjustments will be more than one and have reasonably low discrete distortion of integer numbers.

$$\begin{cases} minW * \mathit{\Delta} \geq 10 \\ minP * \mathit{\Delta} \geq 10 \end{cases} \tag{4-16}$$

The MKP datasets can be modified that preserves the original combinatorial characteristics of the dataset by multiplying all item profits, item weights, and knapsack capacities by a constant value. Using this method, dynamic GK datasets will be modified by a factor of 123, which is large enough to eliminate the small adjustment magnitude problem and reduce discrete distortion to a minimum. These modified datasets will have slight adjustments to item weights, and profits affect the dataset more accurately.

Following is the list of configurations chosen to generate Dynamic MKP benchmark instances:

- 100 generated states with $\mathit{\Delta} = 0.2$
- 100 generated states with $\mathit{\Delta} = 0.1$
- 100 generated states with $\mathit{\Delta} = 0.05$
- 100 generated states with $\mathit{\Delta} = 0.02$
- 100 generated states with $\mathit{\Delta} = 0.01$

Using the method described, a total of 1405 dynamic datasets are generated. 55 dynamic dataset instances are generated from 11 static instances in the GK library, and 1350 dynamic dataset instances are generated from 270 static instances in the OR library.

4.2. Dataset Analysis

Generated dynamic MKP datasets are analysed in two ways: first, dataset statistical analysis, and second, dataset optimal result analysis. The dataset statistical analysis method is meant to determine what changes have occurred to each item from one state to the next state and what is cumulative item discrepancy from the initial state to the last generated state. Dataset states are analysed by profit distance, average weight distance, and absolute weight distance. For dataset results analysis, each state of the dynamic dataset is independently solved using a linear solver that finds the optimal result for the state. The results of each state are compared by finding the solution distance. The solution distance is calculated by counting how many different items are between two state optimal result vectors.

Solution distance:

$$SD = \frac{\sum_{i=0}^{n}(x_{1,i} \oplus x_{2,i})}{n} \tag{4-17}$$

where $x_1$and x_2 are optimal result vectors of dynamic dataset states 1 and 2. Each result point is counted if one result vector includes it and the other result vector does not. It is the normalized binary vector Hamming distance.

Profit distance:

$$PD = \frac{\sum_{i=0}^{n}(P_{1,i} - P_{2,i})}{n} \tag{4-18}$$

Average weight distance:

$$\overline{WD} = \frac{\sum_{i=0}^{n}\left|\sum_{k=0}^{m}(W_{1,i,k}) - \sum_{k=0}^{m}(W_{2,i,k})\right|}{n} \tag{4-19}$$

Absolute weight distance:

$$|WD| = \frac{\sum_{i=0}^{n}\left(\sum_{k=0}^{m}\left|W_{1,i,k} - W_{2,i,k}\right|\right)}{n} \tag{4-20}$$

4.2.1. Statistical analysis metrics

Each dataset has its unique properties and constraints. Therefore, to do analysis, it is essential to understand what boundaries are expected for each dataset due to its constraints.

First, the **theoretical solution distance** $\mathbb{E}(SD)$ is the statistically expected value of solution distance when results vectors of two non-correlated datasets in comparison have random distribution with a constant solution tightness ST. The formula can be reduced to the following.

$$\mathbb{E}(SD) = 2 \times (1 - ST) * ST \tag{4-21}$$

Then **theoretical profit distance** $\mathbb{E}(PD)$ is the statistically expected profit distance for two datasets that follow identical item size and item value constraints, but dataset contents do not have any correlation. The formula can be reduced to the following.

$$\mathbb{E}(PD) = \frac{maxP - minP}{3} \tag{4-22}$$

Theoretical average weight distance $\mathbb{E}\overline{WD}$ is the statistically expected average weight distance for two datasets that follow identical item size and item value constraints, but dataset contents do not have any correlation. The formula can be reduced to the following.

$$\mathbb{E}\overline{WD} = m^{0.5} \times \frac{maxW - minW}{3} \tag{4-23}$$

Theoretical absolute weight distance $\mathbb{E}|WD|$ is the statistically expected absolute weight distance for two datasets that follow identical item size and item value constraints, but dataset contents do not have any correlation. The formula can be reduced to the following.

$$\mathbb{E}|WD| = m \times \frac{maxW - minW}{3} \tag{4-24}$$

4.2.2. Example GK01 dynamic dataset statistical analysis

The graph in Figure 4-2 shows how much each state's all items' profits are, on average, different from the initial state. For the GK01 dataset, the theoretical profit distance is $\mathbb{E}(PD) = 2050$. The dynamic dataset generated with *Δ* SAM - 0.2 has the profit distance to the initial state approaching close to the theoretical profit distance possible for a dataset with these constraints. Dynamic datasets generated with lower *Δ* SAM value do not reach the theoretical value within 100 generated states of the dynamic dataset.

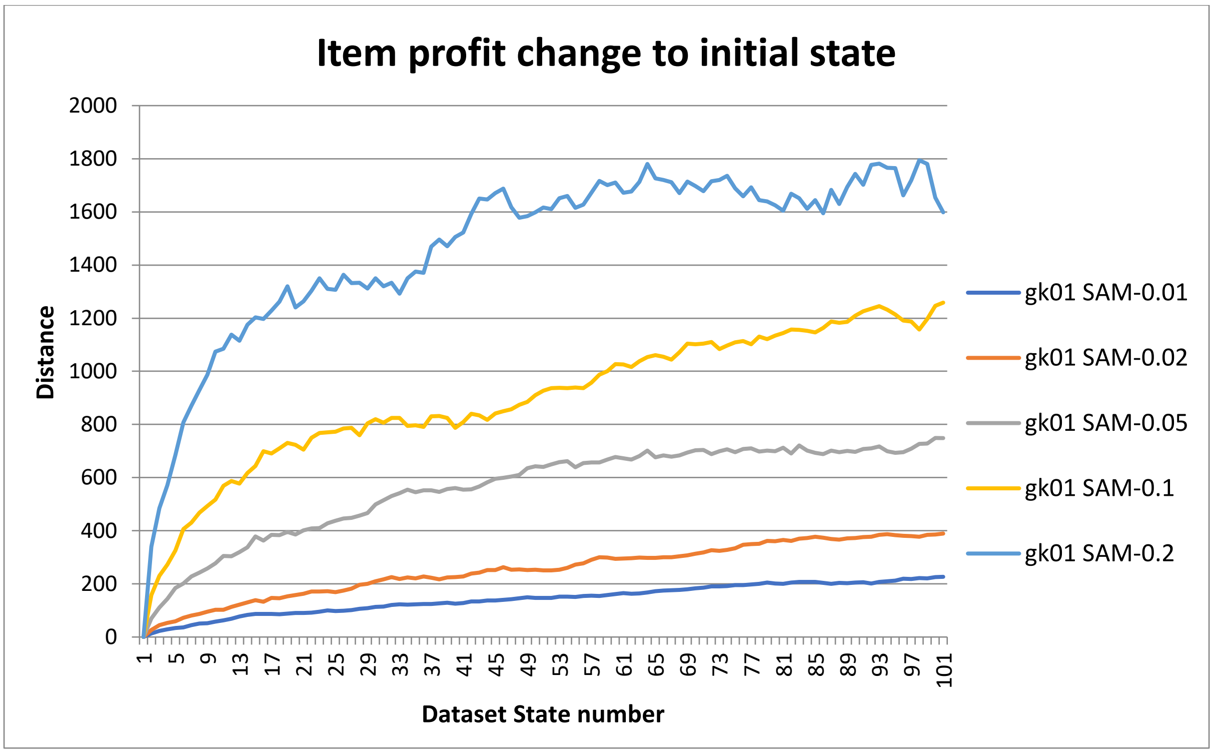

Figure 4-2: Item profit change to initial state for datasets generated from GK01

The graph in Figure 4-3 shows how much all items' average weight differs from the initial state. For the GK01 dataset, the theoretical average weight distance is $\mathbb{E}\overline{WD} = 2382$. The dynamic dataset generated the highest *Δ* SAM - 0.2 has the average weight distance far away from the possible theoretical average weight distance. Reaching the theoretical average weight distance would take significantly more states. This is because the dataset generation method has to simultaneously follow the limit of the weight of the items for each knapsack and item's value, which is the ratio of average weight over profit. This makes item weight distribution significantly slower.

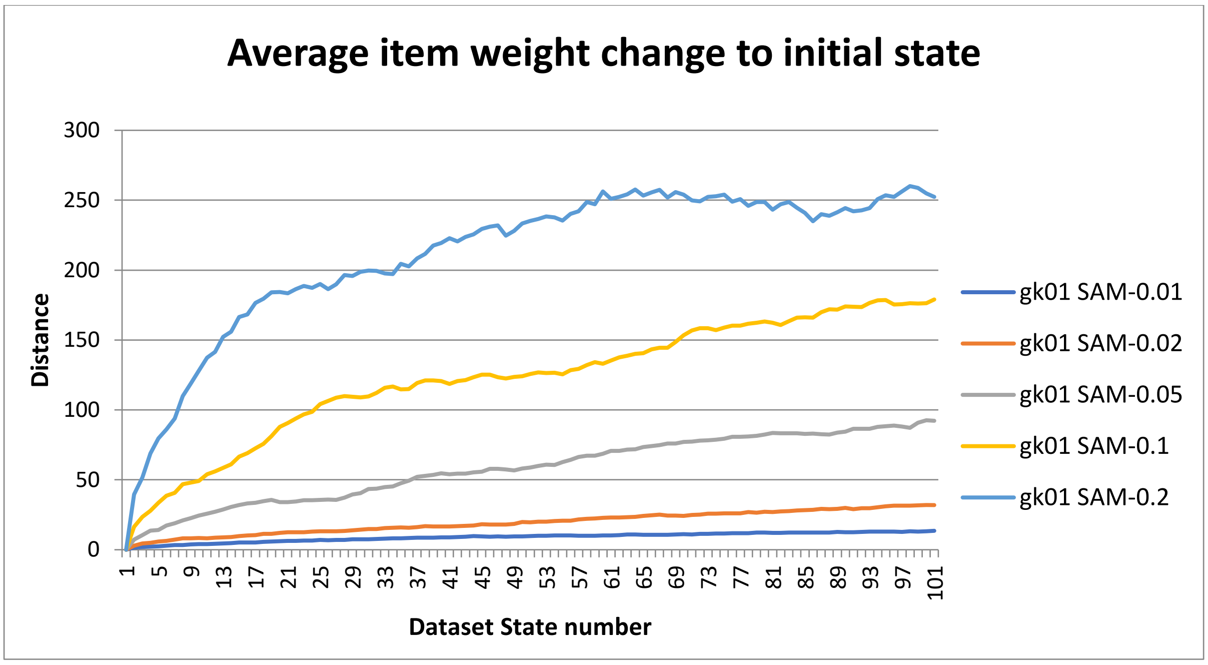

Figure 4-3: Average item weight change to initial state for datasets generated from GK01

The graph in Figure 4-4 shows how much all items' absolute weight of all knapsacks is different from the initial state. For the GK01 dataset, theoretical absolute weight distance $\mathbb{E}|WD| = 9225$. Similar to theoretical average weight distance dynamic dataset generated *Δ* SAM - 0.2 has the average weight distance to the initial state far away from the possible theoretical absolute weight distance. These values follow the same set of constraints and are expressed using different calculations method.

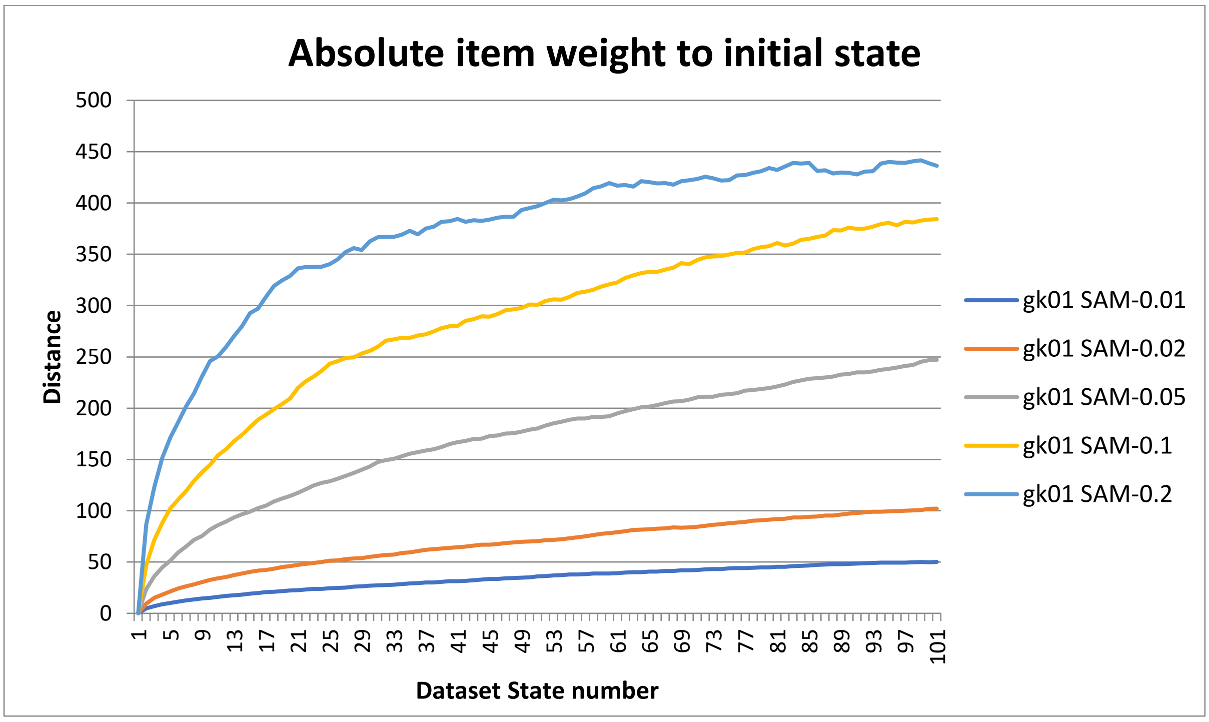

Figure 4-4: Absolute item weight to initial state for datasets generated from GK01

The graph in Figure 4-5 shows how much each states' item profits are, on average, different from the previous state. This profit change value is relatively constant throughout all states of the dataset, as the measurement is not compounding over multiple states. Also, this value is far below the theoretical profit distance. Having a high distance from state to state would make a not useful dynamic dataset because of high disturbances, making the dataset not have any relation among the states. Having reasonably low profit change from state to state enables dynamic optimization algorithms to reuse information in previous states to solve the next state.

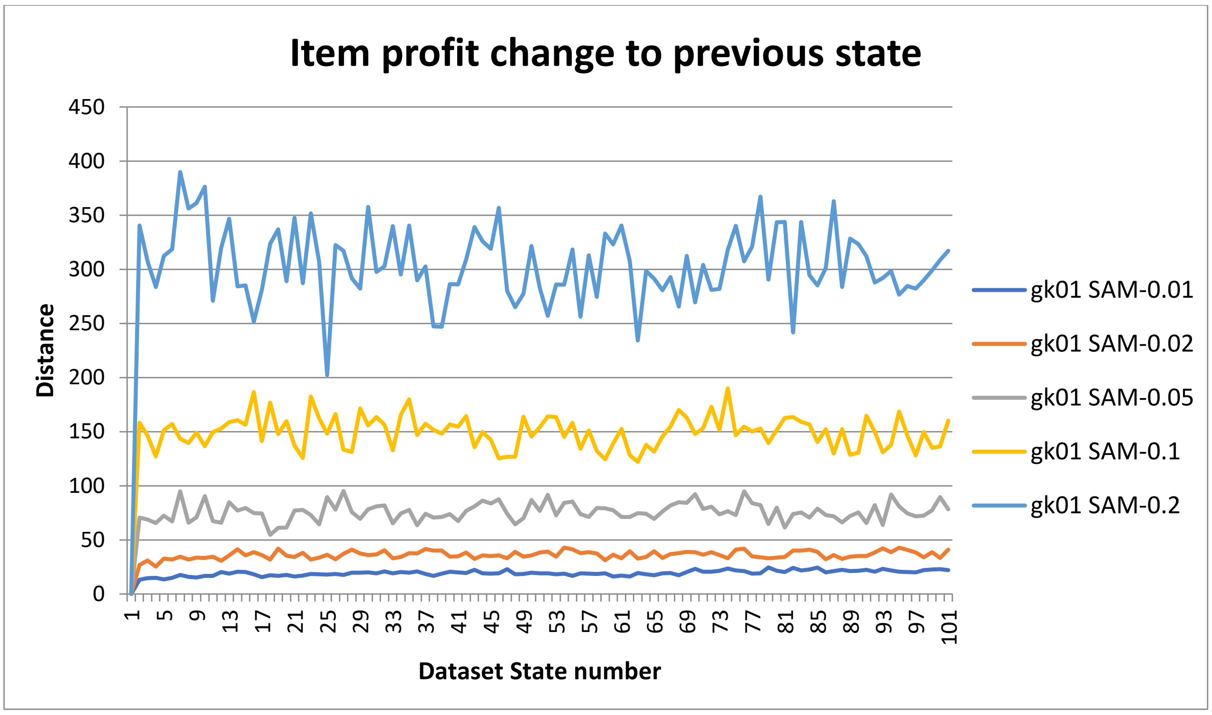

Figure 4-5: Item profit change to the previous state for datasets generated from GK01

The graph in Figure 4-6 shows how much all items' average weight of all knapsacks is different from the previous state. Similarly to item profit, the average weight distance is relatively constant, as the measurement is not compounding over multiple states.

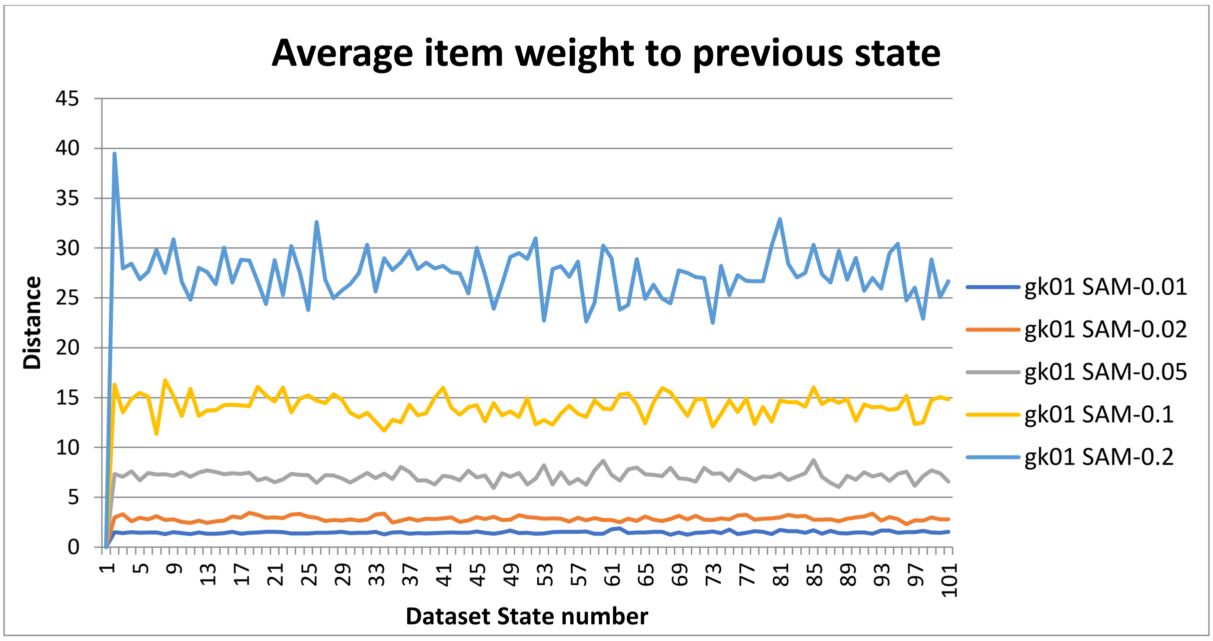

Figure 4-6: Average item weight to the previous state for datasets generated from GK01

The graph in Figure 4-7 shows how much all items' absolute weight of all knapsacks is different from the previous state. Similarly to item profit, the average weight distance is relatively constant, as the measurement is not compounding over multiple states and is far from the theoretical absolute weight distance.

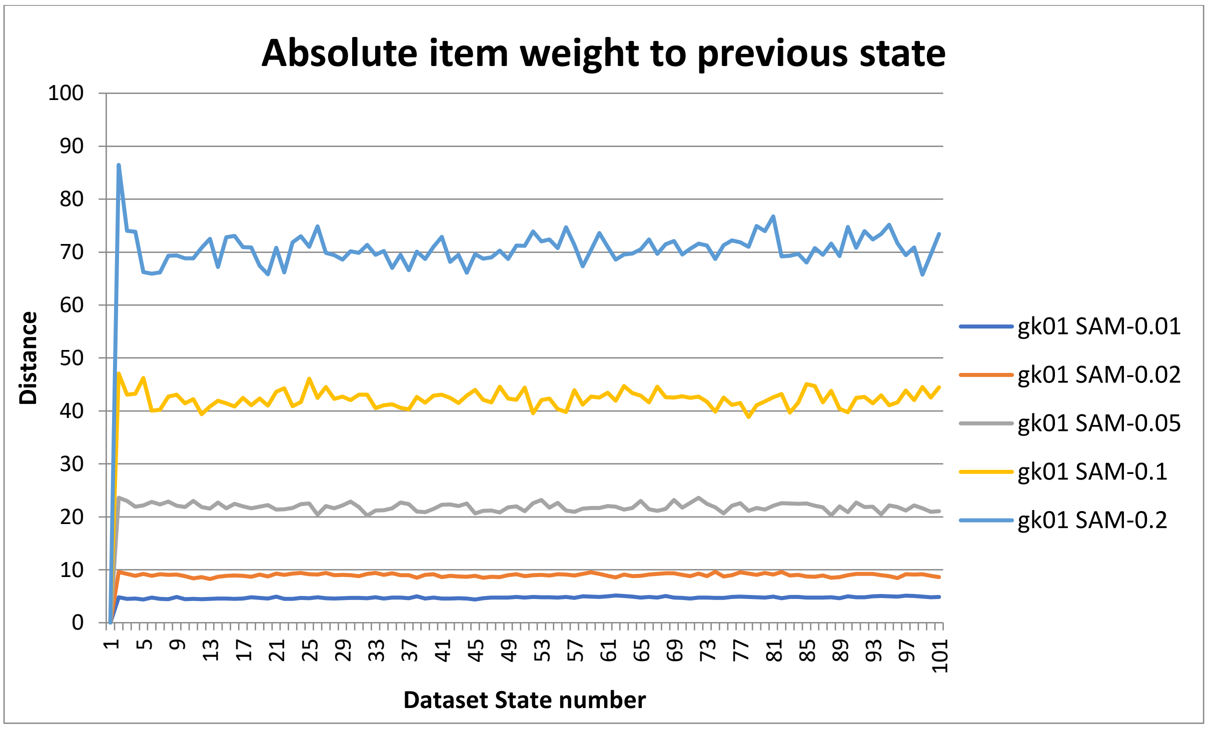

Figure 4-7: Absolute item weight to the previous state for datasets generated from GK01

4.3. Dynamic MKP dataset result analysis

Constrained optimization problems often have sparse solutions for ranked near-optimal solutions. This is due to a large portion of search space being infeasible and the remaining feasible space containing lots of close to optimal solutions distributed far apart in the search space. Multidimensional Knapsack Problem's solutions are incredibly sparse. This is due to the sparse nature of packing problems and made sparser by doing such packing in multiple dimensions. To analyse the results of the dynamic datasets, a small selection of lower combinatorial complexity problem instances in each state has been solved to the optimal solution using Google Or-tools integer linear programming [212].

4.3.1. Example GK01 dynamic dataset result analysis

The graph in Figure 4-8 shows the progression of states' optimal result distance to the initial state S_0 dataset. Where 0 all items in state's optimal result are same as in initial state's optimal result items, and 1 all items in state's optimal result are opposite of initial state's optimal result items. A higher Δ value makes result distance to the initial state's result higher. GK01 Knapsack tightness is exactly 0.5, and the resulting tightness is often very close to knapsack tightness. Therefore, the theoretical solution distance is also $\mathbb{E}(SD) = 0.5$. Over 100 states SAM - 0.01 and SAM - 0.02 are growing, but SAM - 0.05, SAM - 0.1 and SAM - 0.2 do reach theoretical distance and stops growing.

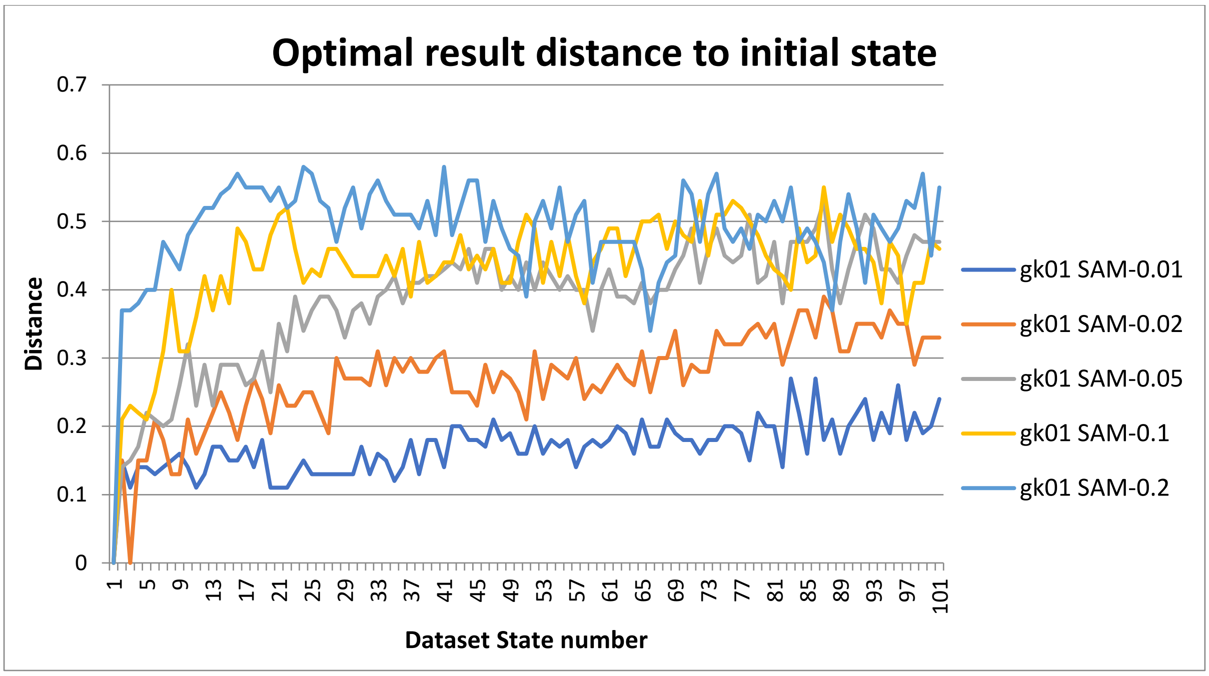

Figure 4-8: Optimal result distance to initial state for datasets generated from GK01

The graph in Figure 4-9 shows the progression of states' optimal result distance to the previous state. It shows how much difference is in items taken to the optimal solution in comparison to the previous state S_{i-1}. For datasets with lower *Δ* value, some states solution distance is zero compared to the state before. Even with slight profit and item weight changes, the optimal solution can still have the same items fit in the knapsack for maximum profit. However, the final result profit will be different and such information not reflected in this graph. Furthermore, with SAM - 0.2, the solution distance is around 0.3 to the previous state, which is quite close to the theoretical solution distance. This dataset characteristic might appear to be very challenging for dynamic optimization algorithms to tackle since there are many changes in the optimal result. This dataset is still valid, regardless of how challenging it is, to test how quickly algorithms can adapt to significant change and find good results improved on previous state's results and not necessarily find the optimal result.

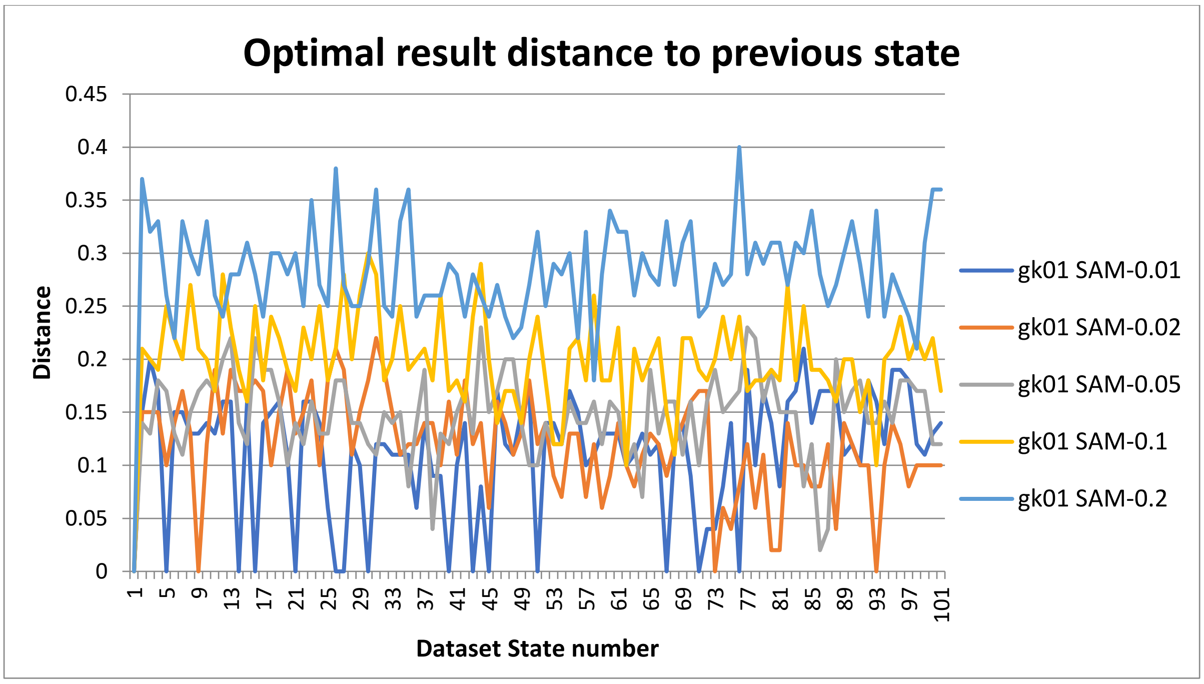

Figure 4-9: Optimal result distance to the previous state for datasets generated from GK01

4.3.2. Dynamic datasets' optimal result scores

The performance of the MKP result is measured with the total profit of items in the knapsack. The optimal MKP result score is the maximum possible profit. Then for dynamic MKP, result performance is measured with a sum of each dataset state result profit. When each dynamic MKP dataset state result is found optimal, then the overall dynamic dataset score is optimal.

Following is the table with the optimal result scores of the dynamic datasets. In Table 4-1, optimal result scores are shown for 50 datasets. There are optimal result scores shown of partial dynamic dataset and full dynamic dataset for each of these datasets. The result of 0 states which is only the initial state's result, 10 states which is optimal result summed up to 10th state, 25 states which is optimal result summed up to 25th state, 50 states which is optimal result summed up to 50th state, 75 states which is optimal result summed up to 75th state, and 100 states which is optimal result of a full dataset.

Table 4-1: Dynamic datasets optimal result scores of selected datasets. Optimal result scores are the sum of 0 states, 10 states, 25 states, 50 states, 75 states, and 100 states.

Dataset	0 states	10 states	25 states	50 states	75 states	100 states
gk01 SAM-0.01	463218	5089903	12043838	23653772	35281859	46897705
gk01 SAM-0.02	463218	5092327	12059177	23693904	35383395	47113463
gk01 SAM-0.05	463218	5113768	12097901	23892037	35778881	47610395
gk01 SAM-0.1	463218	5121272	12169486	24044573	35925950	47924151
gk01 SAM-0.2	463218	5177484	12293033	24189553	36116201	48166300
OR10x100-0.25_1 SAM-0.01	2836872	31205045	73830256	145018245	216448557	288212783
OR10x100-0.25_1 SAM-0.02	2836872	31176234	73779210	145071138	216174187	286840590
OR10x100-0.25_1 SAM-0.05	2836872	31123728	73635411	144442790	214350962	284811984
OR10x100-0.25_1 SAM-0.1	2836872	30805648	71709699	140660402	208541345	276564063
OR10x100-0.25_1 SAM-0.2	2836872	29648279	69427770	133693742	197679341	262428557
OR10x100-0.50_1 SAM-0.01	5091585	55953867	132336914	259552759	386384459	513376968
OR10x100-0.50_1 SAM-0.02	5091585	55866397	132033525	260305360	388821856	517324762
OR10x100-0.50_1 SAM-0.05	5091585	56065926	133118084	262589997	393619307	524410687
OR10x100-0.50_1 SAM-0.1	5091585	56629927	134571121	266159634	395126427	526352982
OR10x100-0.50_1 SAM-0.2	5091585	56542194	135328561	268860774	403977829	540243294
OR10x100-0.75_1 SAM-0.01	7057125	77643072	183396152	359231095	535672559	712133307
OR10x100-0.75_1 SAM-0.02	7057125	77369672	182892248	357984295	532064332	706249480
OR10x100-0.75_1 SAM-0.05	7057125	77602511	183329109	359336913	534961784	709195281
OR10x100-0.75_1 SAM-0.1	7057125	76740410	181074663	349629918	514015441	679144481
OR10x100-0.75_1 SAM-0.2	7057125	74281645	171926494	331859849	492799464	658580578
OR30x100-0.25_1 SAM-0.01	2699358	29687005	70191529	137774216	205487725	273471842
OR30x100-0.25_1 SAM-0.02	2699358	29702444	70268015	137971147	205397065	272700290
OR30x100-0.25_1 SAM-0.05	2699358	29502620	69475293	135491089	201383026	266403778
OR30x100-0.25_1 SAM-0.1	2699358	28900744	68183405	133807293	197430527	262757855
OR30x100-0.25_1 SAM-0.2	2699358	27977949	66054651	130252336	195671607	261708983
OR30x100-0.50_1 SAM-0.01	5014341	55059043	129891484	254621859	379817047	505328790
OR30x100-0.50_1 SAM-0.02	5014341	55025745	130037468	255079511	379914843	505524676
OR30x100-0.50_1 SAM-0.05	5014341	55200665	130045508	254327181	378085224	501037881
OR30x100-0.50_1 SAM-0.1	5014341	54810897	128082790	247397275	366761689	486607011
OR30x100-0.50_1 SAM-0.2	5014341	53103991	124497893	244395075	363760632	483039836
OR30x100-0.75_1 SAM-0.01	7071762	77864424	184162853	361344304	538697991	716206864
OR30x100-0.75_1 SAM-0.02	7071762	77780937	183865316	359936517	536422253	713362025
OR30x100-0.75_1 SAM-0.05	7071762	77932653	184544801	362035765	538160553	714949227
OR30x100-0.75_1 SAM-0.1	7071762	77557884	183180055	361505598	542978957	725238465
OR30x100-0.75_1 SAM-0.2	7071762	77189609	183136611	365412793	547980898	727926255
OR5x100-0.25_1 SAM-0.01	2998863	32939171	77757416	152752166	227713579	302850314
OR5x100-0.25_1 SAM-0.02	2998863	32821440	77541184	152413231	227296574	302426478
OR5x100-0.25_1 SAM-0.05	2998863	33098483	78857803	155459878	233754591	311477600
OR5x100-0.25_1 SAM-0.1	2998863	32819219	79103918	159026997	238008299	318890576
OR5x100-0.25_1 SAM-0.2	2998863	31792534	75667725	150669330	224083685	298265765
OR5x100-0.50_1 SAM-0.01	5259111	57816931	136753398	268534896	400626535	532774988
OR5x100-0.50_1 SAM-0.02	5259111	57739290	136877350	268314426	400103228	532502998
OR5x100-0.50_1 SAM-0.05	5259111	57570616	136465452	269864698	403784094	536018915
OR5x100-0.50_1 SAM-0.1	5259111	58845857	142707776	289653226	440496588	591692649
OR5x100-0.50_1 SAM-0.2	5259111	62153210	151213344	304160231	460660467	610787683
OR5x100-0.75_1 SAM-0.01	7358106	81027232	191732938	376103226	559814098	742782860
OR5x100-0.75_1 SAM-0.02	7358106	81164628	191864001	375883541	559920437	744012008
OR5x100-0.75_1 SAM-0.05	7358106	81539986	193322737	378920251	562125666	744462914
OR5x100-0.75_1 SAM-0.1	7358106	81354829	191189919	372271347	550102686	726936473
OR5x100-0.75_1 SAM-0.2	7358106	80952636	187322886	367257155	546541749	724836393

4.4. Comparative performance analysis

In addition to dataset statistical and optimal result analysis, comparative algorithm performance is tested. A high-performance baseline ACO algorithm implementation called ACO with Dynamic Impact [1] introduced in Chapter 3 has been adapted to solve the Dynamic MKP benchmark. The algorithm has been configured to perform two popular dynamic optimization strategies: Full-Restart and Pheromone-Sharing. Full-Restart strategy is a standard optimization strategy, where each state is considered independently, and after each state change, the optimization is restarted from the beginning. The pheromone-Sharing strategy is a simple yet very effective dynamic optimization strategy, where after each state change, the pheromone is reused [157]. All tests have been executed on the AMD Threadripper 2990WX system with the clock running at 2.9Ghz, with execution parallelism set to 32 threads on the first NUMA node.

To cross-compare highly efficient dynamic optimization algorithm result scores of the Dynamic MKP, each result profit has to be expressed as a profit gap to the best-known profit, or the "result gap" for short. The result gap score is calculated for each state, which is the percentage of the state's result profit difference to the best-known profit. The best-known results are submitted to a verified public repository [213]. Using the result gap allows comparing algorithm performance quantitatively across all benchmark instances.

The dynamic optimization results of the ACO algorithm are displayed in Figure 4-10 and Figure 4-11. The ACO algorithm solved all Dynamic MKP benchmark dataset instances generated from the GK library in both instances. For each SAM Δ value there are 11 GK benchmark datasets run 10 times. Each dynamic dataset state run time has been limited to 1 second per 100 items in the problem. For example, GK01 has 100 items therefore, each state is limited to 1 second runtime, GK03 with 150 items limited to 1.5s, and GK11 with 2500 items limited to 25s. In Figure 4-10, the ACO algorithm with a Full-Restart strategy is configured to solve each dynamic state independently, from the start, without any share of the learned knowledge from previous state optimization. Every state for this optimization strategy appears as a new optimization problem therefore, the convergence of every state is similar throughout the whole optimization. Also, the *Δ* value does not have an impact on the optimization quality for ACO with Full-Restart strategy. In Figure 4-11 ACO algorithm with a Pheromone-Sharing strategy continues to use the same pheromone after dynamic state change and therefore has a significant head start to improve the solution further. ACO with Pheromone-Sharing strategy can take a significant advantage when the *Δ* value is low because each dynamic change is small and the optimal solution is not significantly different compared to the state before the change.

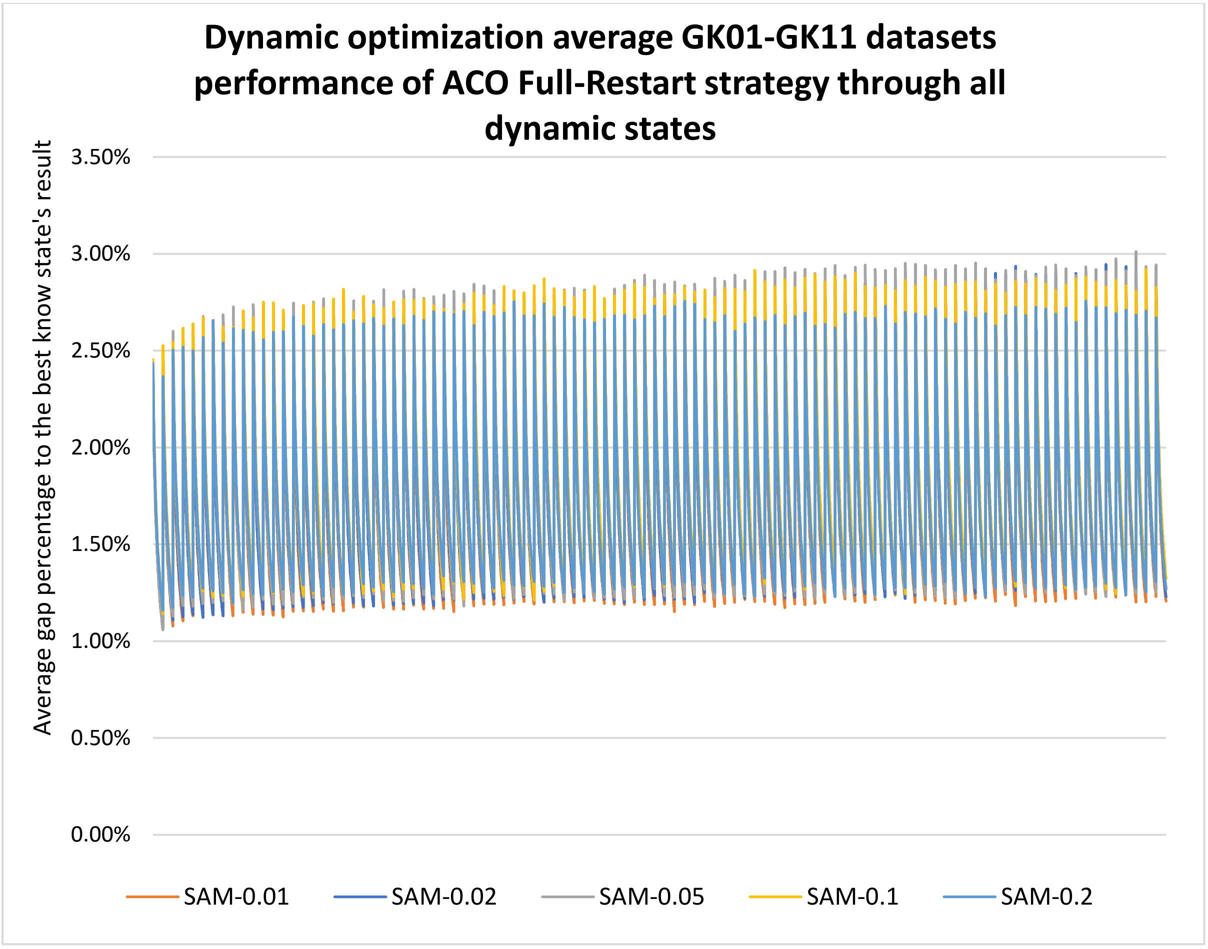

Figure 4-10: Dynamic optimization performance of ACO Full-restart strategy for all SAM levels. Each line shows the average gap convergence of GK01-GK11 dynamic datasets group run 10 times each, totalling 110 runs.

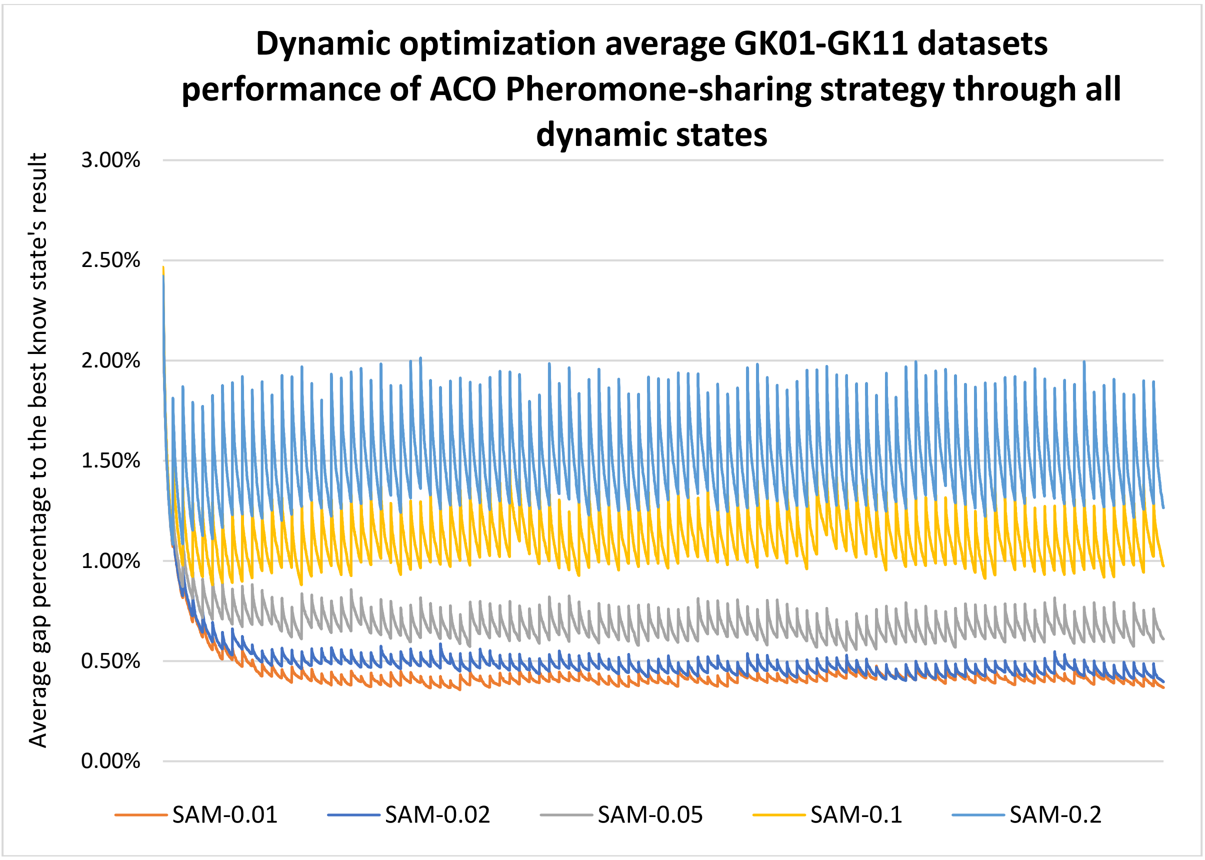

Figure 4-11: Dynamic optimization performance of ACO Pheromone-sharing strategy for all SAM levels. Each line shows the average gap convergence of GK01-GK11 dynamic datasets group run 10 times each, totalling 110 runs.

4.5. Further dynamic dataset analysis

Dynamic datasets are numerically heavy, and static on-the-paper visualizations such as graphs, diagrams, or tables cannot show a complete picture and give the reader an intuitive understanding of the dataset and its dynamics. For this reason, further dataset analysis demonstration is developed. This analysis is not possible to be printed out, therefore the analysis is published on GitHub with complete data of all dynamic datasets [5].

4.5.1. Profit and weight distance effect

The profit and weight distance effect demonstration is a dynamic scatter plot where each item is represented by a dot on a value over a size plot. All items are divided into four groups by their weight and profit values. Groups are chosen considering two factors. First, whether item profit is higher or lower than median item profit, and second, whether item weight is higher or lower than median item weight. Since profit and weight are independent variables, this divides all items into four equally sized groups. Each item is marked for the initial state dataset and remains constant in all states of the dynamic dataset.

For example, the distance effect is displayed of dataset GK01 SAM-0.05 for the initial state and the last state in Figure 4-12 and Figure 4-13. At first, for the initial state, the plot appears evenly divided into four quadrants. Series 1 is initially low weight and low value items; Series 2 is initially low weight and high value; Series 3 is initially high weight and low value items; Series 4 is initially high weight and high value items. Then, all groups become increasingly mixed up by advancing graphs through each state until each series can appear to have low and high value and weight items scattered. When the dataset profit and weights reach theoretically expected distance values, the groups should look mixed entirely up. In the example of GK01 SAM-0.05 last state, the groups do not appear to be completely mixed up. Most of the large weight items remained on the heavy side, and most of the low weight items remained on the light side.

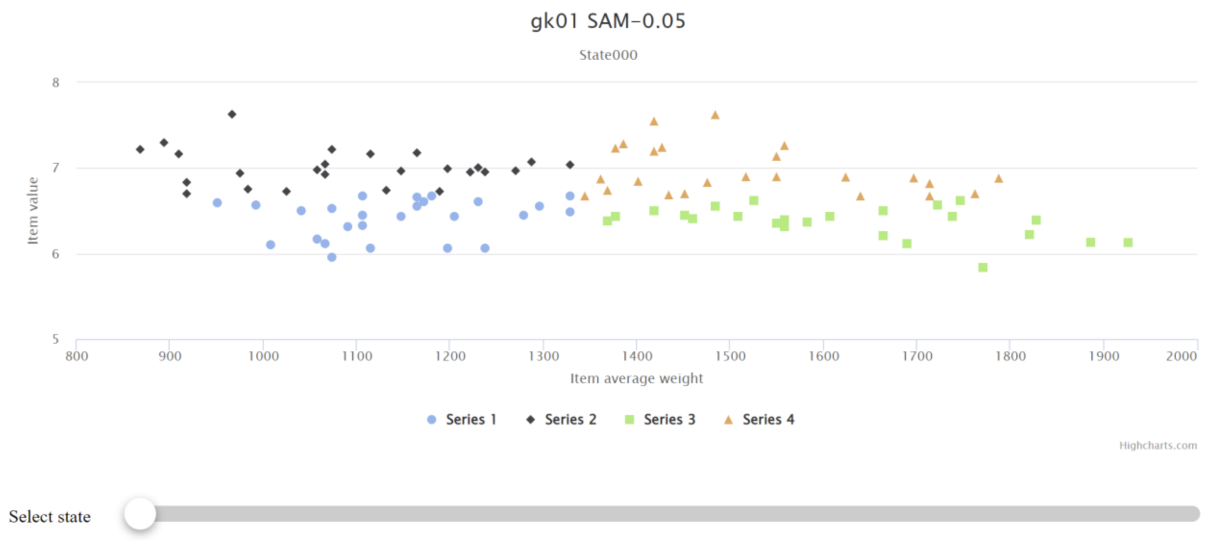

Figure 4-12: Profit and weight distance effect for GK01 SAM-0.05 dataset initial state. Each series represent the division of each item's value and average weight into a quadrant based on the initial state. On the initial state, the division is clearly visible.

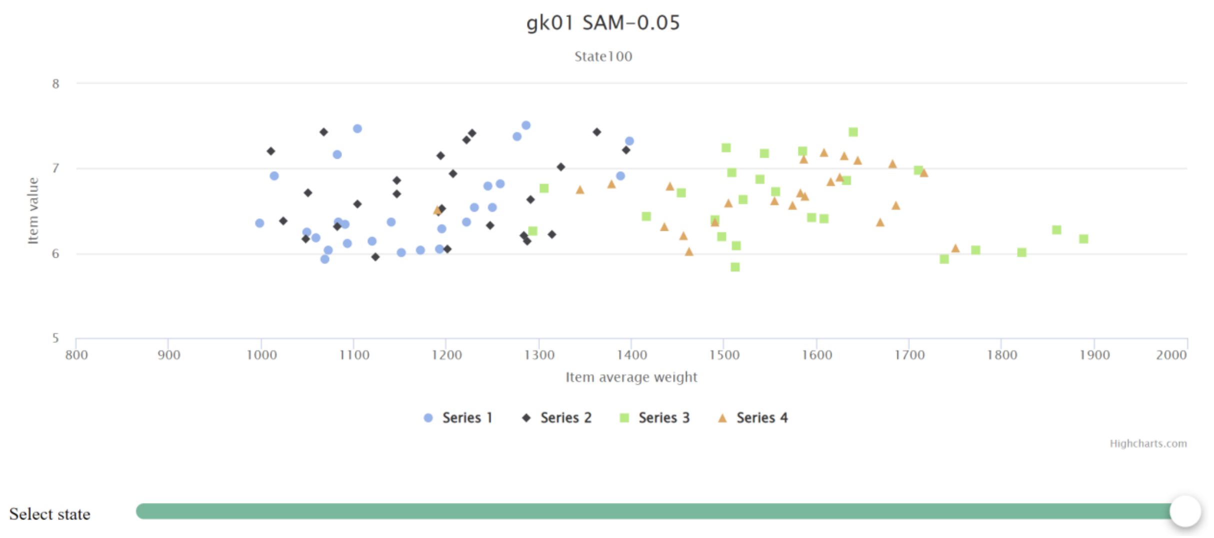

Figure 4-13: Profit and weight distance effect for GK01 SAM-0.05 dataset last state. Each series represent the division of each item's value and average weight into a quadrant based on the initial state. On the last state, items are significantly mixed up.

4.5.2. Optimal result effect

Similarly to profit and weight distance effect demonstration, an optimal result effect demonstration is a dynamic scatter plot where each item is represented by a dot on a value over a size plot. However, in this dynamic plot, each item belongs in a group according to the optimal result decision vector obtained from the linear solver solution for each state. The item is either part of the optimal set or not. From one state to another, the item may change the group to reflect a new optimal solution for that given state.

For example, the optimal result is displayed of dataset GK01 SAM-0.05 for the initial state and the last state in Figure 4-14 and Figure 4-15. In both example figures, the higher value items are significantly more likely to be included in the optimal result decision vector than lower value items. However, the item's size does not appear to impact the likelihood of being included in the optimal result decision vector.

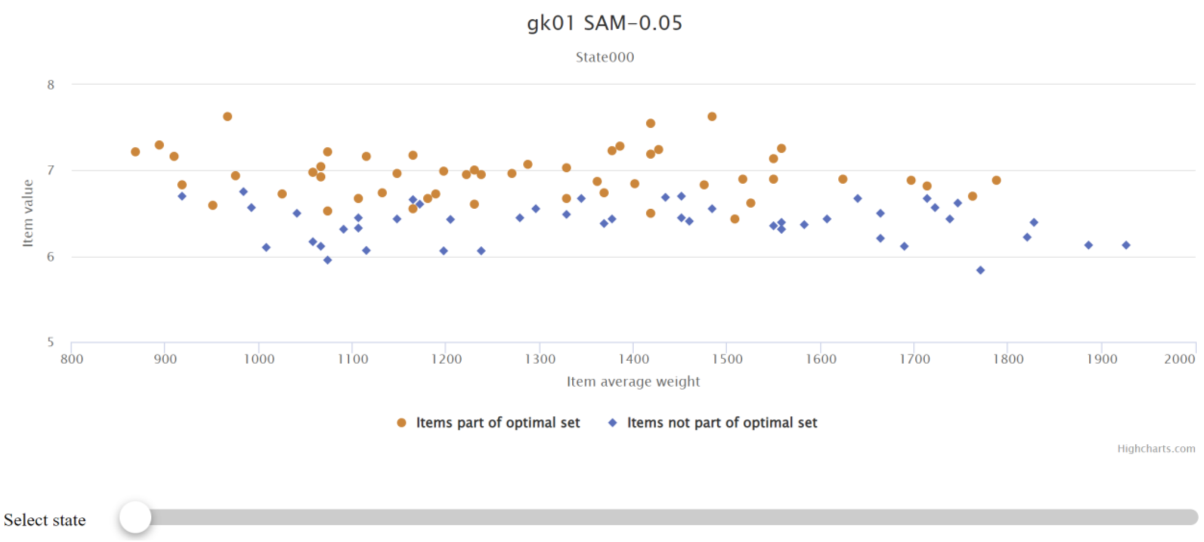

Figure 4-14: Optimal result effect GK01 SAM-0.05 dataset initial state

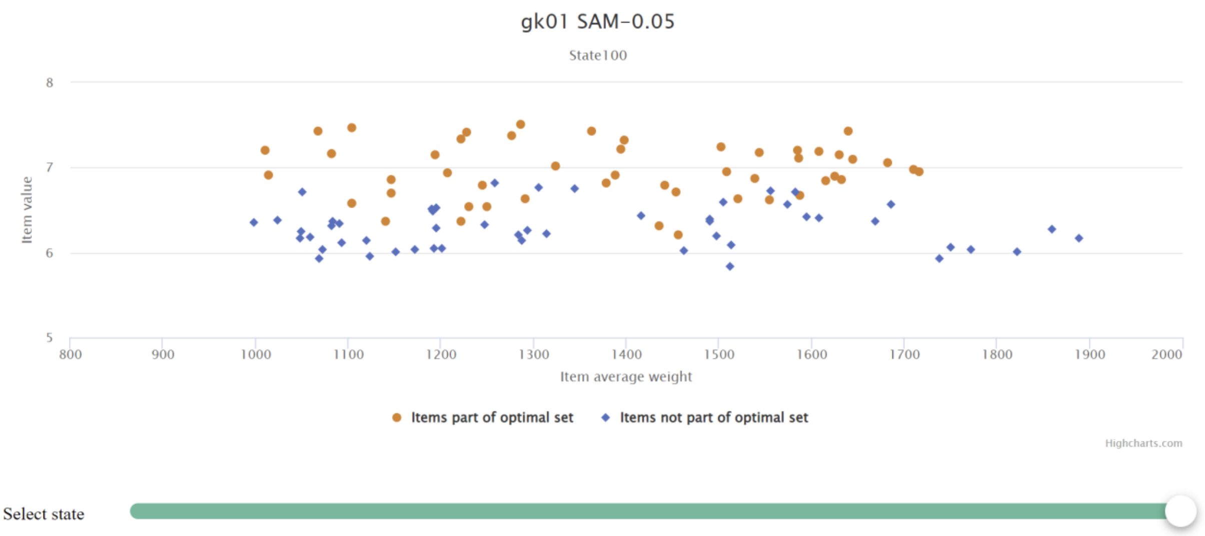

Figure 4-15: Optimal result effect, GK01 SAM-0.05 dataset last state.

4.5.3. Dynamic dataset constraint coverage effect

This visualization chart is a group of line graphs where each line represents an item of the dynamic dataset. The line shows the path of the item that has been moved through the dataset constraint space. The chart can display up to 20 items at once, and it can limit the number of dynamic states range for a more transparent comparison of each item's path.

For example, the dataset constraint coverage paths are displayed for items 13 to 21 inclusive and span through all states from 0 to 100 in Figure 4-16 and Figure 4-17. The GK01 dataset generated using SAM-0.02 has all items cover a smaller, more localized constraint space than GK01 SAM-0.05. Items in the GK01 SAM-0.05 dataset has a broader coverage and has more overlap in the constraint space among the items.

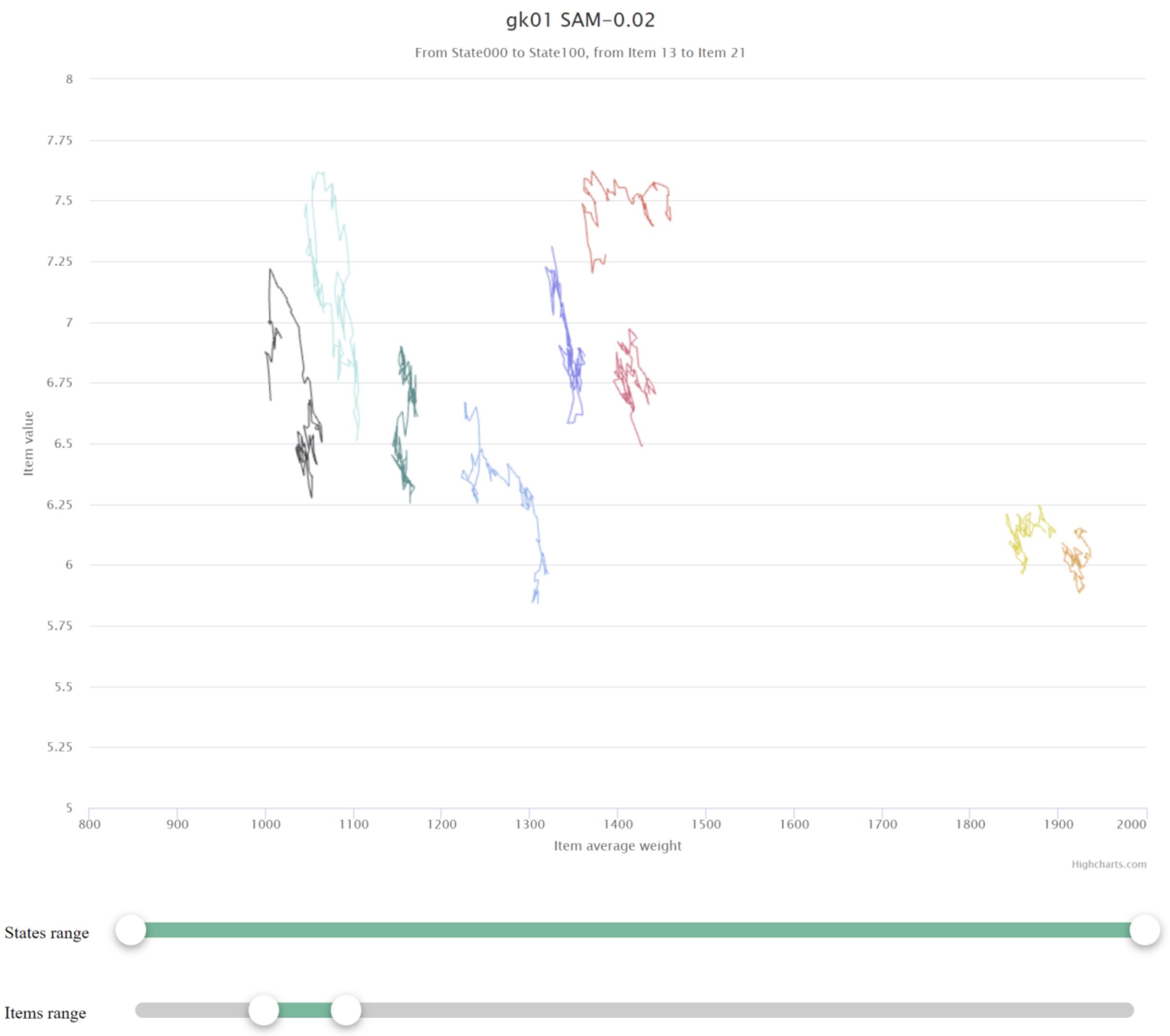

Figure 4-16: Dynamic dataset constraint coverage effect, GK01 SAM-0.02 dataset, items range 13-21 inclusive.

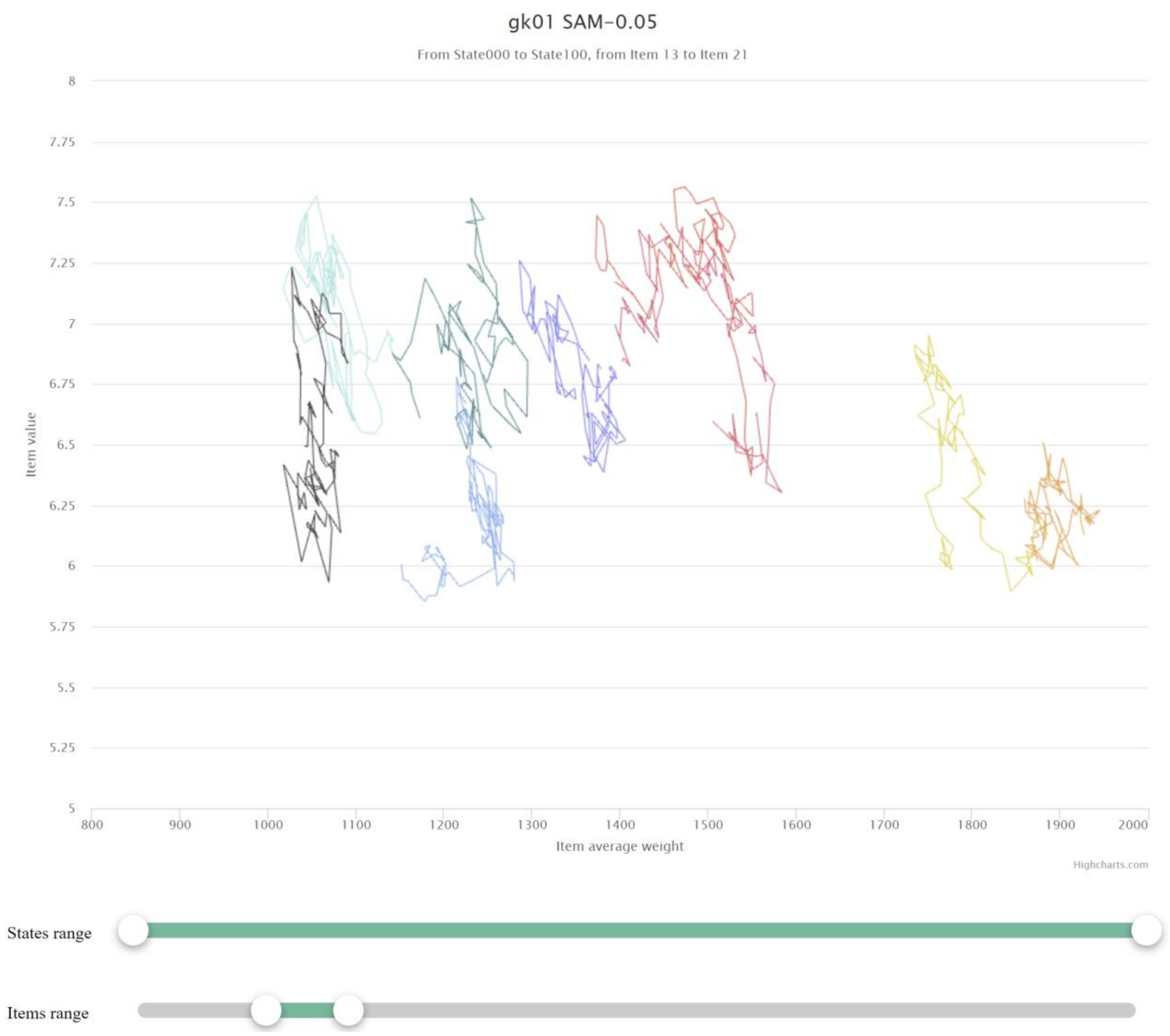

Figure 4-17: Dynamic dataset constraint coverage effect, GK01 SAM-0.05 dataset, items range 13-21 inclusive.

4.5.4. Dynamic dataset optimal result coverage effect

For the dynamic datasets with optimal results, the optimal result coverage of every state can be displayed. This chart displays for all states whether the item belongs in the decision vector of optimal solution or not. Orange colour represents an item in the optimal set and blue colour represents a not optimal item. The chart also differentiates the items always part of the optimal set with green colour series and items that are never part of the optimal set with black colour series.

For example, optimal item's decisions are displayed for items 7 to 21 inclusive and span through all states from 0 to 100 in Figure 4-18 and Figure 4-19. In dataset GK01 SAM-0.01, where each item has mutated the least, more items have remained always optimal or never optimal compared to dataset GK01 SAM-0.02, where items cover a larger constraint area and therefore larger changes in size and value have an effect on the optimal solution.

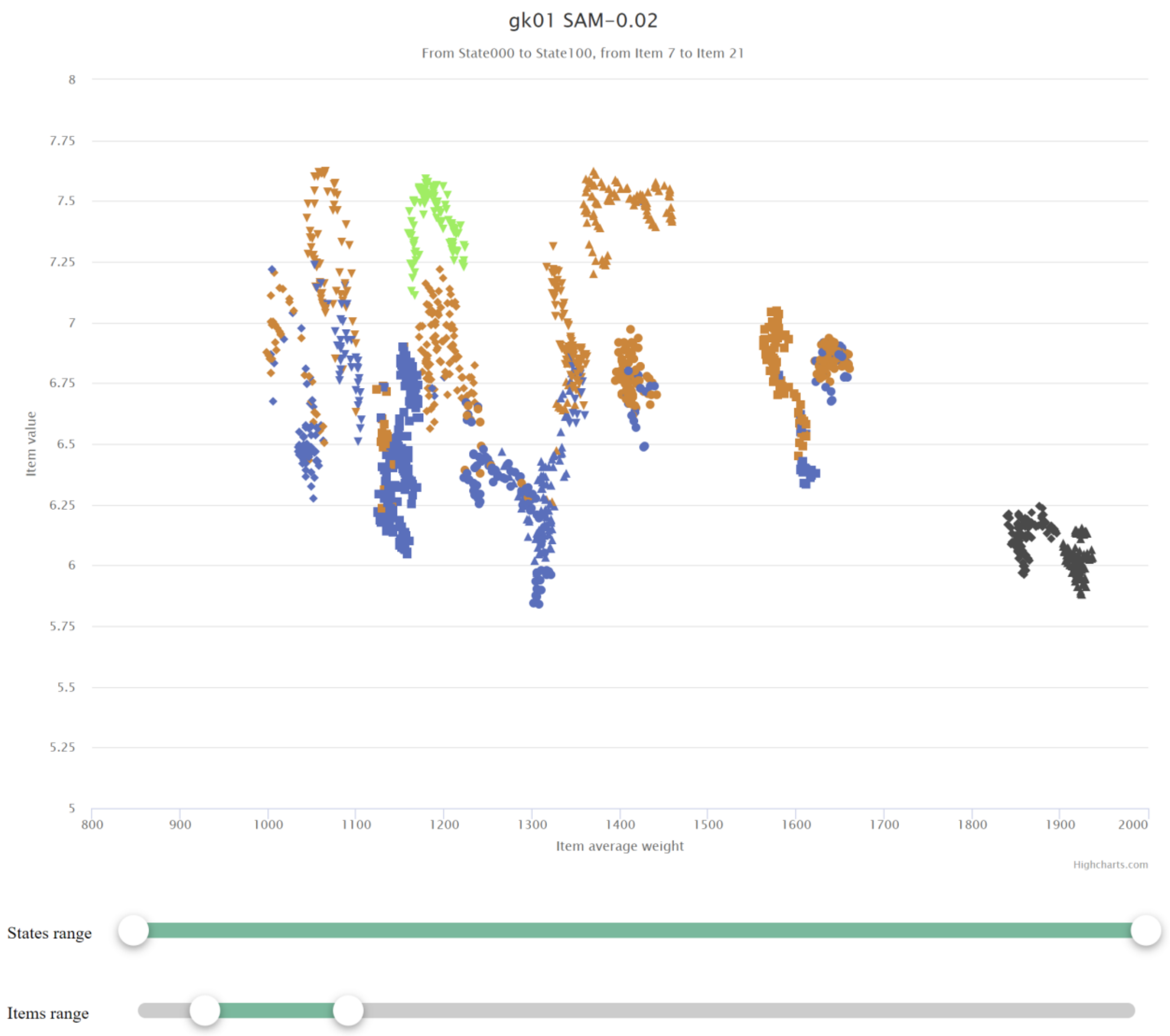

Figure 4-18: Dynamic dataset optimal result coverage effect, GK01 SAM-0.02 dataset, items range 7-21 inclusive.

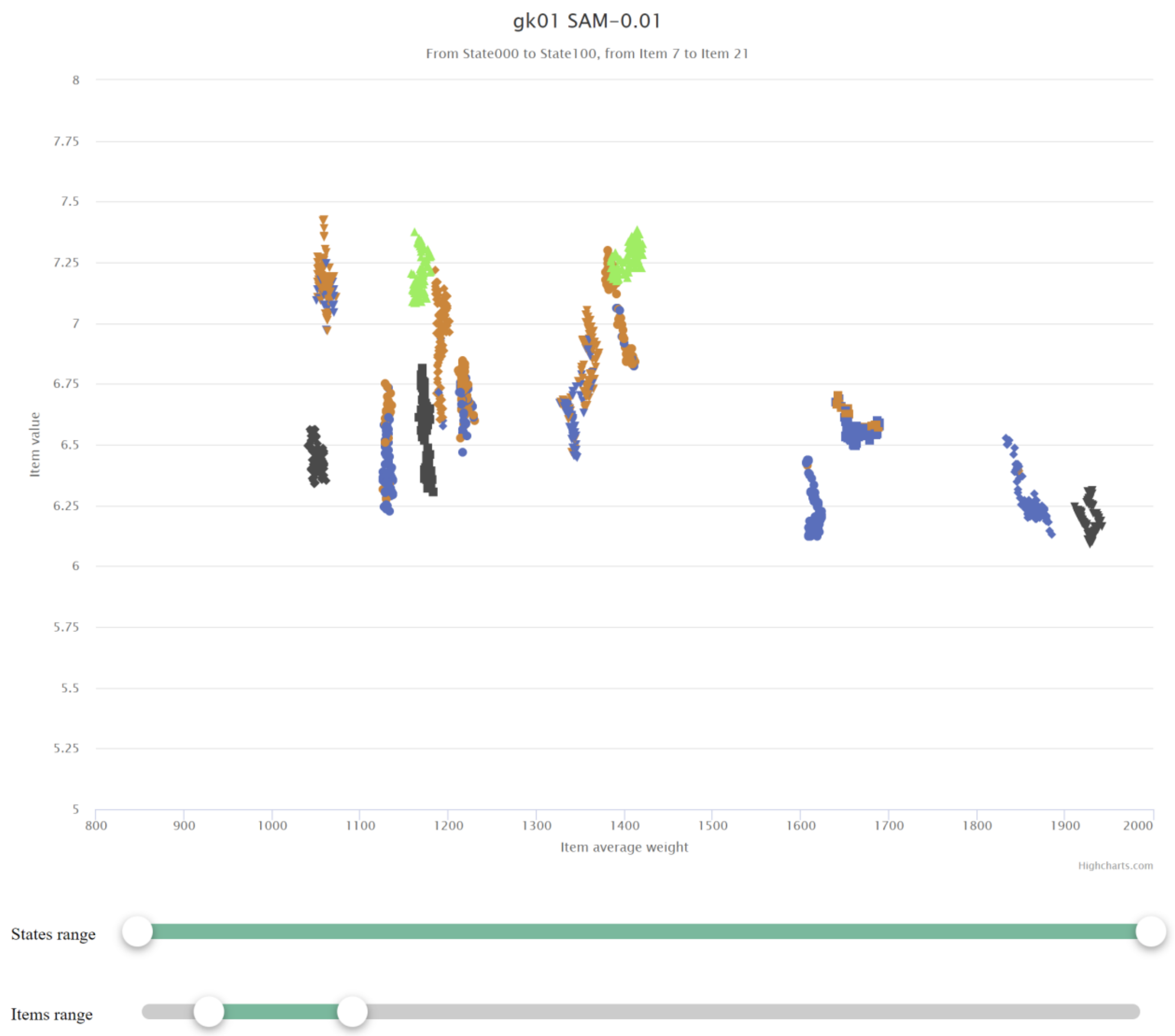

Figure 4-19: Dynamic dataset optimal result coverage effect, GK01 SAM-0.01 dataset, items range 7-21 inclusive.

4.6. Chapter Summary

This research chapter has resolved a critical gap in discrete Dynamic Optimization Problem (DOP) research. There were no fully defined DOP datasets upon which the research could be based. Previous works have used stochastic generation methods and have not preserved the optimization states or random operator seed values to directly compare the optimization results. Therefore, it was impossible to evaluate dynamic optimization algorithms fairly or conduct a repeatability study.

Introduced a non-stochastic dynamic dataset generation method that can consistently generate the next state of dynamic MKP based on nothing but input dataset and *Δ* value. The generated dataset will always be identical based on the input dataset.

Therefore, dynamic optimization algorithms can be cross-compared in future research by any research work.

Using this dynamic dataset generation method, 1405 fully defined Dynamic MKP benchmark instances were generated from the existing static MKP benchmark dataset library. Then those dynamic datasets were published to be used as a Dynamic MKP benchmark.

This chapter also provided a Dynamic MKP benchmark datasets analysis. The datasets were analysed quantitatively for the range of dynamism of all dataset parameters. Also, optimal result dynamics analysis was performed on 455 datasets with low combinatorial complexity of 100 items, where all states were solved to optimal result using a linear solver. Then developed an interactive tool for an additional dynamic demonstration which helps to develop an intuitive understanding of the dynamics of the datasets.

Finally, the new Dynamic MKP benchmark will be used in Chapter 5 to measure the performance of the new nature-inspired dynamic optimization strategy developed for the ACO algorithm. The non-stochastic nature of the benchmark makes it easy to compare the results of the baseline optimization and the proposed algorithm.

The contributions of this research chapter to science are:

- Introduced a deterministic dynamic dataset generation method that takes a static instance of the MKP dataset and generates a dynamic dataset consistently. Dataset Generator is published on GitHub. [2]
- Generated new, fully defined Dynamic MKP benchmark instances from existing static MKP benchmarks for consistent and repeatable cross research reference. The benchmark datasets are published on GitHub [3] and IEEE Dataport [4].
- The generated benchmarks are qualitatively and quantitatively analysed and proven as valid Dynamic MKP datasets. The visualization tool is published on GitHub. [5]
- The work presented in this chapter has been published in the in peer-reviewed journal Systems and Soft Computing, Elsevier [6].

Chapter 5. Herder Ants: Ant Colony Optimization with Aphids for Discrete Event-Triggered Dynamic Optimization Problems

As mentioned in Chapter 2 literature review, currently applied dynamic optimization strategies for ACO algorithm are rudimentary and not explicitly designed with dynamic optimization in mind. This research chapter introduces a discrete dynamic optimization strategy called Ant Colony Optimization (ACO) with Aphids, modelled after a real-world symbiotic relationship between ants and aphids. ACO with Aphids strategy is designed to improve solution quality of discrete domain Dynamic Optimization Problems (DOPs) with event-triggered discrete dynamism.

Up to recently, there was no existing Discrete event-triggered DOP benchmark to evaluate the efficiency of proposed algorithms, especially for a limited time per state of the dynamic optimization. This work uses fully-defined DMKP benchmark datasets proposed in Chapter 4 and evaluates the efficiency of several different ACO dynamic optimization strategies. Also, for maximum optimization performance, all implemented dynamic optimization strategies are based on the best performing ACO with Dynamic Impact configuration presented in Chapter 3.

5.1. Ant Colony Optimization with Aphids

The original Ant Colony Optimization (ACO) algorithm was described by Dorigo in his doctoral thesis [113], solving Traveling Salesman Problem in 1992. ACO algorithm has been modelled to mimic real ants' behaviour. While navigating, ants deposit pheromone on their path, and then other ants sense it and are drawn to it. A stronger pheromone trail attracts more ants compared to a weaker pheromone trail. When an ant travels a long distance from the food source to the nest, the pheromone trail is naturally spread out over that distance, and the pheromone evaporates faster, making the trail unattractive. On the other hand, if the path is short, each ant's round trip time is shorter, allowing the ant to deposit more pheromone on a short path and attract even more ants. Such ant behaviour is fundamentally iterative and allows ants to explore all the available areas for food sources and exploit already found food sources using the shortest travel path.

However, this optimization algorithm was not modelled with dynamic optimization problems in mind. But luckily, as mentioned in the literature review, some ants manifest another behaviour, herding aphids. In nature, those ants tend to aphids and collect their produced honeydew, providing an additional source of nutrition along with their usual scavenged food. Within ants' pheromone, aphids behave differently. They move less and produce more honeydew, while ants protect them from predators. Also, aphids rely on ants to relocate them when environmental conditions change. For example, ants move them onto new fresher plants when the plant is no longer fresh. Ants can prey on aphids once aphid's honeydew production decreases or aphids' population is too large.

5.1.1. ACO with Aphids design

This research proposes to use aphids in the ACO algorithm to increase the performance of discrete dynamic optimization. In this algorithm, the aphids are modelled as immobile food producers for ants to pick up. Ants must pick up this food at each dynamic change of the problem, creating a baseline pheromone for the new dynamic environment proportional to aphids' distribution around search space. Then ants continue to explore the current state of the environment by optimizing the combinatorial optimization problem. Then after optimization is finished, ants kill a portion of aphids and lay some new aphids on the best edges of the current state of the environment. Aphids do not move independently, but after each dynamic change, ants relocate a portion of aphids to a better location according to precalculated heuristic information.

5.1.2. Optimization system

For discrete dynamic optimization, the ACO algorithm runs within the optimization system to separate concerns of the optimization process and the data. The optimization system handles fully defined benchmark data. The optimization system mimics real-world dynamic optimization scenarios where future dynamic change is unknown. Each state of the dynamic optimization problem is dispatched with a constant time interval for the ACO to solve. Then ACO solves the optimization problem at its current state until the next state is dispatched. The system also records the fitness score and results of the ACO optimization for every state.

Generally, the degree of dynamism in the literature is defined as the frequency and the magnitude of dynamic change [8]. However, for benchmark dynamic combinatorial optimization problems, the main focus is only on the magnitude of dynamic change. The frequency of dynamic change will be set to a constant time window such that a fair comparison of dynamic optimization strategies is performed.

5.1.3. ACO with Aphids algorithm

ACO with Aphids design extends high performance ACO with Dynamic Impact algorithm introduced in Chapter 3. Aphids represent learned information mediators across all states of the dynamic optimization problem. For each dynamic optimization problem's state, the algorithm initializes search space with all feasible edges, sets each edge's pheromone to the default pheromone level τ_0, and precalculates heuristic information. Before the ant search starts, ants perform two additional steps unique to the ACO with Aphids algorithm. Firstly, they relocate aphids based on new search space heuristic information and aphids' relocation parameter. Then, ants collect honeydew produced by aphids and set the initial pheromone level to base the search in the new environment. Then ants perform iterative search normally for a given time span. Once the search is terminated, ants kill a portion of aphids according to the aphids' kill parameter. And finally, ants lay down new aphids based on the best solution found for the current state and aphids' lay down parameter. The algorithm is further summarised in the pseudo-code below.

Table 5-1: ACO with Aphids pseudo-code

```
Algorithm 1: ACO with Aphids
Input: Dynamic optimization problem dataset
Output: Solutions to the dynamic optimization problem
1.   Initialize Aphids to default aphids' level
2.   FOR state IN dataset states DO
3.     Load state data to memory
```

```
4.      Prepare search space subroutine
5.      Initialize Ants' pheromone
4.      Relocate Aphids
7.      Collect honeydew by ants
8.      **WHILE** no events **AND** no termination **DO**
9.        Build Ant solution subroutine
10.       Update ants' pheromone
11.     **END WHILE**
12.     Record best state's solution
13.     Kill portion of aphids
14.     Lay new Aphids based on the best solution
15.   **END FOR**
```

5.1.3.1. Initialize Aphids to default aphids' level

Aphids are initialized only once for dynamic optimization, and then aphids evolve together with the dynamic environment. At the very start, aphids are initialized uniformly for the whole search space. The parameter must be above zero for the optimization algorithm to work correctly. The default value is $A_0 = 1$.

$$A_{j,i} := A_0, \qquad \forall (j, i) \tag{5-1}$$

Aphids' level is assigned to the default aphids' level on all edges, where, $A_{j,i}$ is the aphids' level of j node and i edge, A_0 is the default aphids' level parameter.

5.1.3.2. Load data to memory and prepare search space

At any given point during dynamic optimization, only one dynamic optimization state is loaded into memory. Previous optimization states are out of date and no longer help in the optimization of the current state. Further states are not revealed to the optimization algorithm as they are technically in the future. Then once the state is revealed, search space has to be prepared for that state. In the prepared search space, each edge has precalculated heuristic information $\eta_{j,i}$. Every optimization problem will have different formulas to calculate heuristic information since this heuristic information is based on the expert knowledge of the optimization problem that provides a myopic guide to the ACO algorithm.

5.1.3.3. *Initialize ants' pheromone*

In the Min-Max Ant System, static optimization starts with an initial pheromone matrix of default value τ_0 set to each edge. The same principle is applied to dynamic optimization. ACO with Aphids algorithm initializes the pheromone matrix for each optimization state to default pheromone value.

$$\tau_{j,i} := \tau_0, \qquad \forall(j,i) \tag{5-2}$$

Ants' pheromone is assigned to the default ants' pheromone level on all edges, where, $\tau_{j,i}$ is the pheromone level of j node and i edge, τ_0 is the default pheromone level parameter.

5.1.3.4. *Relocate aphids*

When the optimization system introduces the new state, new heuristic information is precalculated that can roughly point to areas where aphids should be placed. Ants use precalculated heuristic information to relocate aphids from the search space areas with worse heuristic information to areas with a better heuristic. Aphids are only partially relocated based on heuristic information because the heuristic information is not a perfectly accurate measure of fitness. Relocation of aphids noted by aphids' relocation parameter A_r. The aphids' relocation parameter is a multiplier value and must not be negative $A_r > 0$. The default value of the aphids' relocation parameter $A_r = 1$. However, the parameter value can be higher if heuristic information variance is low, and vice versa.

$$A_{j,i} := A_{j,i} \times \left(1 + \left(\eta_{j,i} - \bar{\eta}\right) \times A_r\right), \qquad \forall(j,i) \tag{5-3}$$

Aphids' level is reassigned to new aphids' level based on heuristic information and aphids' relocation parameter, where, $A_{j,i}$ is the aphids' level of j node and i edge, A_r is the aphids' relocation parameter, $\eta_{j,i}$ is heuristic information of j node and i edge, and $\bar{\eta}$ is average heuristic information value across the entire search space.

5.1.3.5. *Collect honeydew*

Honeydew is a crucial product of aphids. Before the iterative search starts, ants pick up the honeydew produced by aphids and lay pheromone on those edges where honeydew is produced. The amount of pheromone laid down is proportional to the amount of honeydew and, in turn, proportional to the number of aphids living on the edge. The default value of the aphids' honeydew production parameter $A_h = 1$. Increasing or decreasing this parameter value increases or decreases aphids' total influence on the dynamic search. However, too much honeydew might affect ants' ability to evaporate excess pheromone created while collecting honeydew.

$$\tau_{j,i} := \tau_{j,i} + (A_{j,i} \times A_h), \qquad \forall (j,i) \tag{5-4}$$

Ants' pheromone is reassigned to a new level based on aphids' level and honeydew production rate, where, $\tau_{j,i}$ is the pheromone level of j node and i edge, $A_{j,i}$ is the number of aphids currently living on the j node and i edge, and A_h is the aphids' honeydew production rate.

This newly laid pheromone forms a solid starting point for ACO to find good initial solutions after the dynamic change has occurred. Aphids' impact on pheromone is applied only once, and its effect does not negatively impact the iterative search convergence. Ants' pheromone normally evaporates as the search progresses, allowing ants to explore search space efficiently without getting stuck in the local optima.

5.1.3.6. Build ant solution

ACO with Aphids performs iterative search normally as described in MMAS [115]. In the iterative search, a set of ants each builds a complete solution independently. Each ant starts the search with an empty partial solution $s_p = \emptyset$. Then the ant searches for a single edge to add to the partial solution. Ant stochastically adds edge to the solution based on the calculated edge's probability without exceeding the optimization problem constraints. The solution is complete once every node's objective is met or constraints are exhausted.

5.1.3.7. Update ants' pheromone

All completed ant solutions are evaluated against fitness function within a single iteration. The best solution is then passed to update the global pheromone. Pheromone update consists of two steps, first evaporation and second lay down. During the evaporation step, a portion of pheromone is reduced by evaporation parameter ρ as in the following equation (5-5). Then 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 (5-6):

$$\tau_{j,i} := \tau_{j,i} \times (1-\rho), \qquad \forall(j,i) \tag{5-5}$$

$$\tau_{j,i} := \tau_{j,i} + \rho \times \Delta\tau_0, \qquad \forall(j,i) \in s_p \tag{5-6}$$

where ρ is a constant parameter of the pheromone evaporation rate introduced by Dorigo and Stützle [123], $\Delta\tau_0$ is the pheromone update rate, s_p is the solution of the chosen ant to lay down the pheromone.

5.1.3.8. Termination criteria

The search runtime is terminated of a current dynamic state when either one of three conditions occurs, event-triggered, time-based or iteration based. The first condition is when the dynamic change event is triggered. The system mimics the real-world dynamic change in the optimization problem's instance and dispatches new and updated problem search space. The system treats the solutions to the old dynamic state as invalid to the new search space and restarts the search. The second condition is time-based. The optimization is considered complete when the optimization time reaches a predefined time limit allocated for the optimization. The third condition is iteration based. Similarly to time-based termination, the optimization is considered complete when the iterative search has performed a predefined number of iterations.

5.1.3.9. Kill portion of aphids

After the iterative search is finished, a portion of aphids is killed based on the aphids' kill rate parameter A_k. Killing aphids procedure ensures that aphids' population does not grow too much. Also, the killing aphids procedure further removes aphids from poorly performing edges throughout multiple optimization states. Like in nature, a portion of aphids must die such that aphids populations stay in equilibrium $A_r > 0$. The default value of the aphids' kill rate parameter $A_k = 0.5$, which kills half of the remaining aphids.

$$A_{j,i} := A_{j,i} \times (1-A_k), \qquad \forall(j,i) \tag{5-7}$$

Aphids' level is reassigned to a new aphids' level based on aphids' kill rate, where, $A_{j,i}$ is the number of aphids on j node and i edge, and A_k is the aphids' kill rate.

5.1.3.10. *Lay new aphids*

The best solution found by ACO for the state is representative of that state's environment. Ants lay down new aphids on the edges of the search space that are included in the best solution. Lay down aphids procedure increases the number of aphids on well-performing edges and strengthens honeydew production. The default value of the aphids' lay down parameter $A_l = 1$. However, this parameter can be higher if typical solutions contain a small portion of edges available in the search space and vice versa.

$$A_{j,i} := A_{j,i} + A_l, \qquad \forall(j,i) \in s_p \tag{5-8}$$

Aphids' level is reassigned to a new aphids' level based on aphids' lay down rate, where $A_{j,i}$ is the number of aphids, A_l is the aphids' lay down rate, and s_p is the best solution chosen for lying down the aphids.

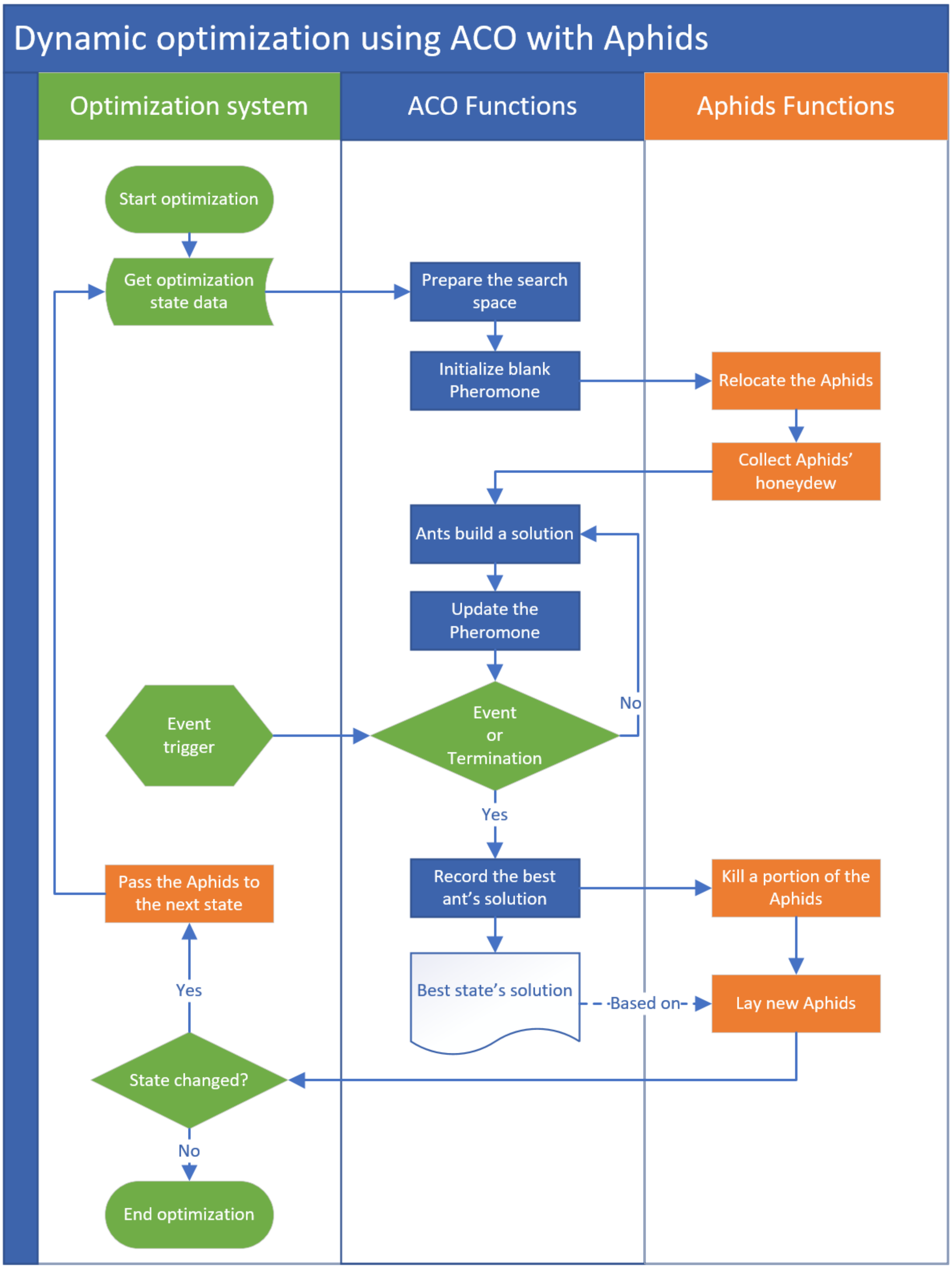

Figure 5-1: ACO with Aphids algorithm flowchart. The green colour represents Optimization system steps, the blue colour represents ACO algorithm steps, and the orange colour represents novel steps to ACO with Aphids algorithm.

5.2. Experimental setup

The experimental work presents the comparison of the proposed ACO with Aphids dynamic optimization strategy against the two most popular ACO dynamic optimization strategies: Full-Restart and Pheromone-Sharing [120], [214], [215].

5.2.1. Experimental dataset

A fully defined Dynamic Multidimensional Knapsack Problem (DMKP) benchmark, introduced in Chapter 4, is taken to test the proposed algorithm's performance. Each benchmark dataset includes complete information about each optimization problem instance called states [3], [4], [6]. Each benchmark dataset contains 101 states, where the first state is the initial state based on the static benchmark instance of the MKP and 100 deterministically generated states. This benchmark is perfect for reliably testing an event-triggered dynamic optimization system and comparing the results of dynamic optimization algorithms. Also, DMKP suits this aim well because algorithmic solutions solving MKP have a wide range of possible real-world applications.

From this benchmark library, 55 datasets of the GK group are used for the experimental work. Each GK dataset has a unique complexity and dynamism combination. There are eleven complexity levels in the range from the GK01 dynamic dataset group with 100 items and 15 knapsacks to the GK11 dynamic dataset group with 2,500 items and 100 knapsacks, and five dynamism levels indicated by the State Adjustment Magnitude Δ parameter: $\Delta = 0.01$, $\Delta = 0.02$, $\Delta = 0.05$, $\Delta = 0.1$, $\Delta = 0.2$. The Δ parameter is also called SAM in the code and charts with no special characters' support. For the first experiment of ACO with Aphids hyper-parameter tuning, only GK03 and GK08 complexities are used with all five dynamism levels. All 11 dataset groups from the GK group with all five dynamism levels are used for the second experiment of a full comparison of dynamic optimisation strategies.

5.2.2. Baseline ACO algorithm and optimization system

All three dynamic optimization strategies are implemented within the same baseline ACO core algorithm, solving static MKP benchmarks [1]. High-quality MKP solutions were possible to achieve by utilizing a dynamic impact evaluation method in the ACO edge's probability calculation. Also, this baseline ACO algorithm implementation efficiently utilizes modern multicore computer architectures by running multiple ant searches in parallel within one iteration and synchronising before the pheromone update.

A fair comparison of dynamic optimization algorithm performance is ensured by the optimization system dispatching an event-trigger at a precise interval based on the complexity of the optimization problem. Event-trigger dispatch based on time eliminates any variations in algorithm execution overheads related to strategy. The optimization system has the rule to dispatch events after a time period proportional to the number of items in the dataset. Each state has 1 second to execute optimization for every 200 items in the dataset. For example, any dataset in the GK01 dataset group has 100 items. Therefore, each state is allowed to execute for only 0.5 seconds. The GK11 is the largest dataset group with 2500 items per dataset. Therefore, each state is allowed to execute for 12.5 seconds.

ACO algorithm hyper-parameters have been tuned in Chapter 3 and used throughout all experimentation. The best combination of pheromone parameters is: $\tau_{max} = 1$, $\tau_{min} = 0.001$, $\rho = 0.1$. Configuration of probability parameters: $\alpha = 1$, $\beta = 0$, $\gamma = 8$, $q0 = 0.01$. Each iteration runs 512 ants in parallel.

The experimental environment is also carefully controlled to ensure the consistency of computational power among the tests. All tests have been executed on the AMD Threadripper 2990WX system with the clock running at 2.9Ghz. Only one experiment was allowed to run on the system simultaneously without any other background tasks. Execution parallelism is set to 32 threads on the first NUMA node. The ACO core algorithm and dynamic optimization strategies are implemented in C++ language and compiled with the MSVC compiler.

5.2.3. Experimental measurements

The absolute performance of the DMKP benchmark instance is the sum of each state's result profits. The total profit sum is a good overall performance indication because each state equally contributes to the final result. A profit improvement of any state's result directly improves the total profit. The result performance is only comparable against the same benchmark instance because every dataset instance will have a different optimal total profit value. However, if the optimal or best-know values are available, a comparable metric can be calculated as the result's percentage difference from the best-known result called the “result gap”. The goal is to minimize the result gap, ideally to zero, which means the best-known or the optimal solution is found. The result gap can be calculated for any single solution and plotted on a graph. Also, dynamic algorithm performance can be measured using the difference in the result gap that occurs after the dynamic change called the “gap slip”. Although gap slip is not a primary objective of the optimization, it indicates how well the optimization algorithm tackles a dynamic problem and should be minimized.

This research performs quantitative analysis across all benchmark instances using “result gap” and “gap slip” metrics. The average result gap and average gap slip values of each dynamic benchmark dataset are calculated from all dynamic states, as shown in Figure 5-2. The best-known results are taken from a verified public repository [213]. Furthermore, all presented experimental measurements are the averages taken from 10 algorithm runs, and the standard deviation of the result gap is calculated to prove statistical significance.

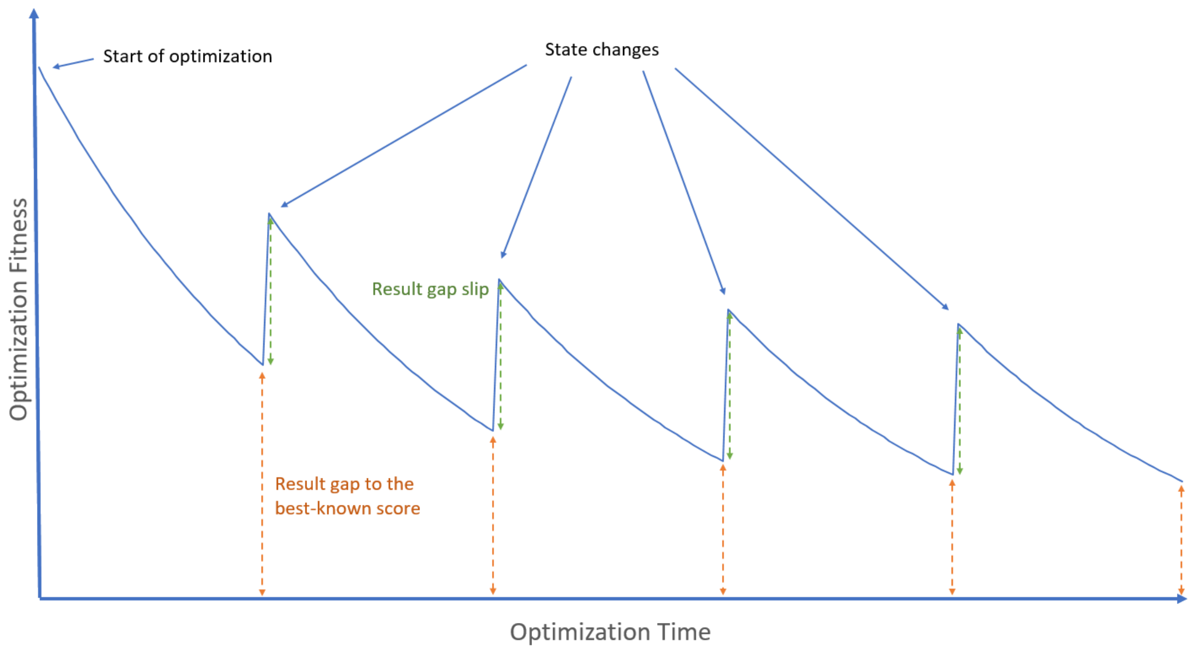

Figure 5-2: Experimental measurements visualization. The orange line represents a measurement of each state's result gap to the best know profit score. The green line represents the result gap slip after the dynamic change. The total dynamic optimization result gap is an average of all states' result gap, and the total dynamic optimization result gap is an average of all states' gap slip.

5.3. Experimental results

The experimental work is split into two parts. The first part is dedicated to an iterative tuning of ACO with Aphids strategy hyper-parameters using a reduced benchmark dataset sample. Then the second part is dedicated to comparing ACO with Aphids strategy with Full-Restart and Pheromone-Sharing strategies.

5.3.1. ACO with Aphids hyper-parameter tuning results

There are four new Aphids functions with tuneable parameters in the ACO with Aphids algorithm, as shown in Figure 5-1. Initially, all Aphids tuneable parameters are initialized to the default values, as shown in Table 5-2, and sweep the tested the algorithm's performance by varying one parameter value per test. The tests are performed incrementally. The first test starts with all default parameter values, and the following tests use the best parameter values found in the previous tests. Hyper-parameter tuning is general to the algorithm, and tests can be performed on a subset of datasets to reduce computational demand. GK03 and GK08 dataset groups with all five dynamism levels are selected for the hyper parameter tuning test to represent low and high complexity members of the benchmark.

Table 5-2: Aphids' tuneable parameters table. Each parameter has a default value, min-max value range used in tests, and test resolution.

Parameter	Default Value	Min Test Value	Max Test Value	Test Resolution
Aphids' relocation: A_r	1	0	2	0.25
Aphids' honeydew production: A_h	1	0	2	0.25
Aphids' lay down rate: A_l	1	0	2	0.25
Aphids' kill rate: A_k	0.5	0.01	1	0.2

The first tested parameter is Aphids' relocation A_r. This parameter partially allows Ants to relocate a portion of the Aphids based on the heuristic information. The heuristic information is not a perfect measure of fitness. However, it can prove helpful for dynamic optimization. The results show a clear optimization improvement with higher A_r values and best-tested configuration is with $A_r = 2$.

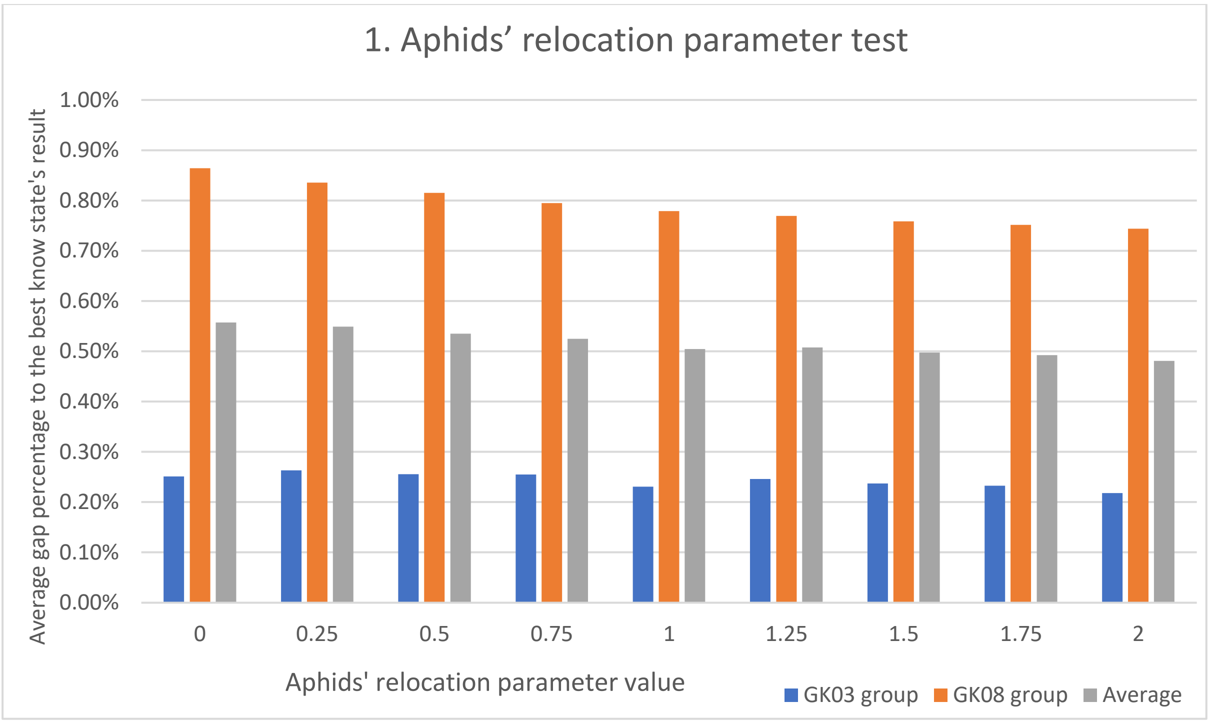

Figure 5-3: ACO with Aphids hyper-parameter tuning test number 1. Aphids’ relocation parameter test. The results show the best dynamic optimization performance is achieved using $A_r = 2$.

The second tested parameter is Aphids’ honeydew production A_h. This parameter controls how much honeydew each aphid produces at the start of the iterative search, where higher values result in stronger pheromone trails at the start of each dynamic optimization state. The results show a significant optimization improvement with higher A_h values up to $A_h = 1$ after which the results had no more improvement. With honeydew production above 1, Aphids saturate Ants’ pheromone trails with honeydew, and extra honeydew does not further benefit the dynamic search.

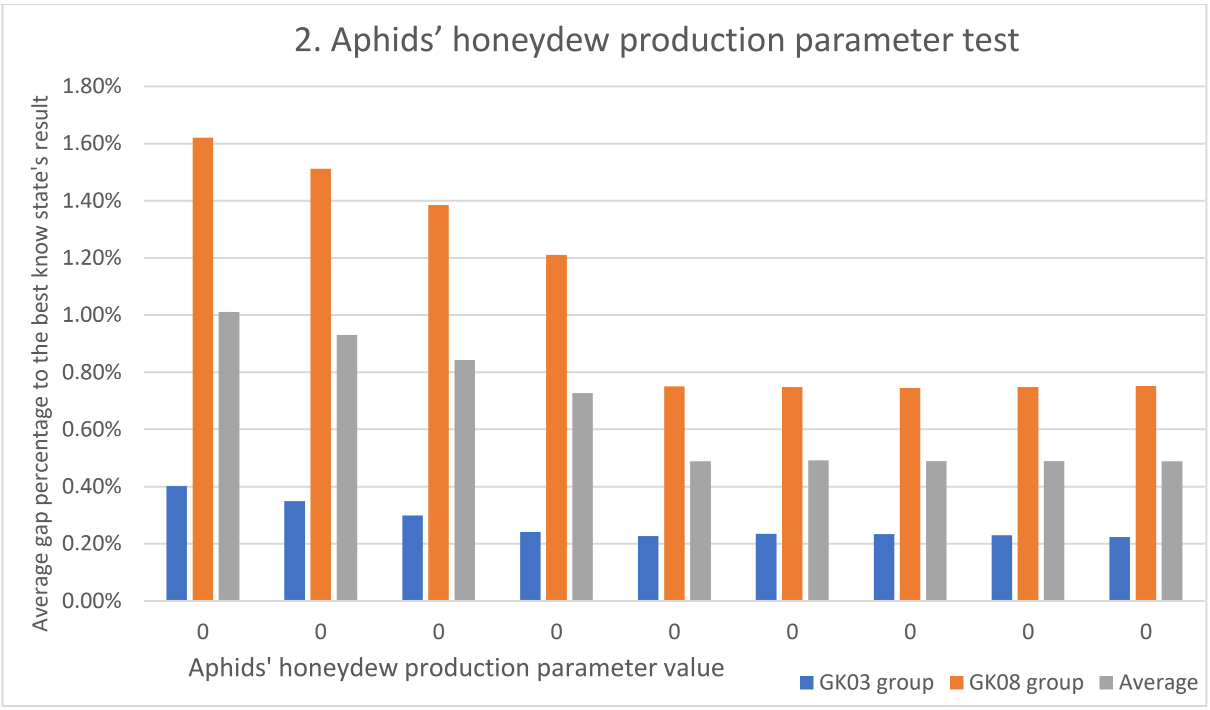

Figure 5-4: ACO with Aphids hyper-parameter tuning test number 2. Aphids’ honeydew production parameter test. The results show the best dynamic optimization performance is achieved using $A_h = 1$.

The third tested parameter is Aphids’ lay down rate A_l. The results of the GK03 complexity group are not significantly diminished, and the larger GK08 complexity group performs well too. This parameter controls how much Ants lay down new Aphids on the edges of the best solution obtained in the previous state’s optimization. The results show significant improvement with a lay down rate above 0, which is to be expected. When no aphids are laid, the information is no longer shared between states. When parameter $A_l = 1$, it appears to be a middle ground where the smaller

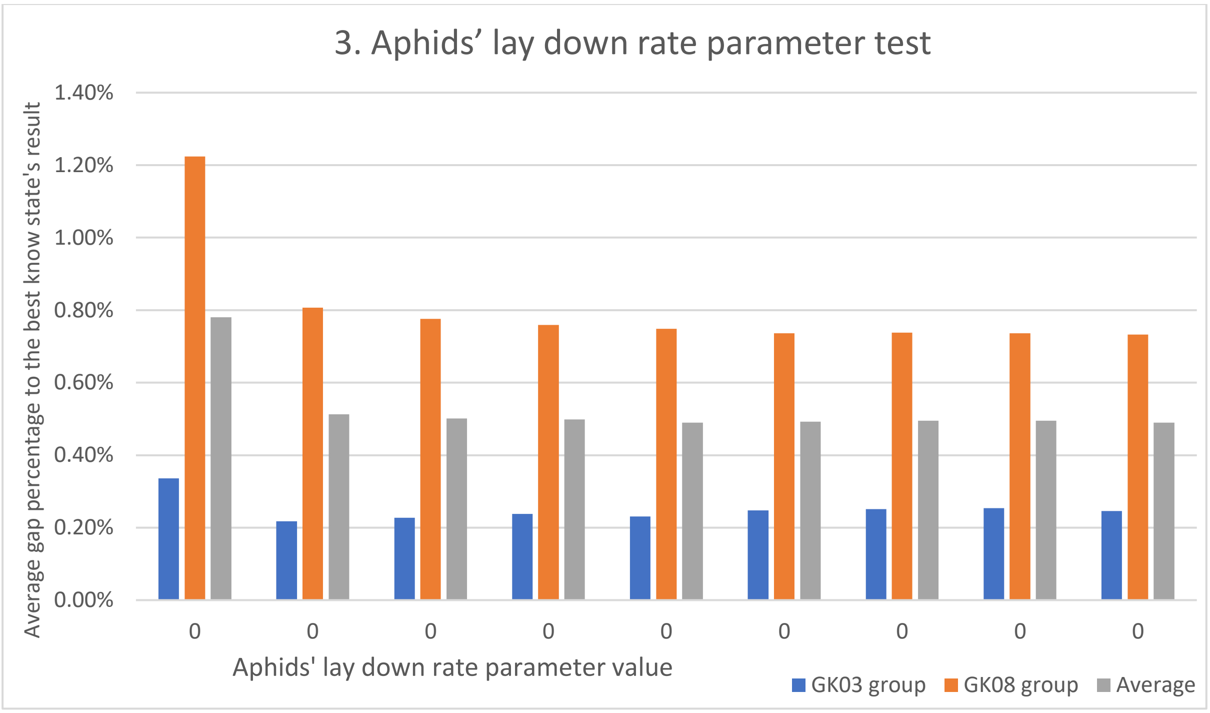

Figure 5-5: ACO with Aphids hyper-parameter tuning test number 3. Aphids’ lay down rate parameter test. The results show the best dynamic optimization performance is achieved using $A_l = 1$.

Finally, the last parameter tested is the Aphids’ kill rate A_k. This parameter controls which portion of the aphids is killed after completing the iterative search. This procedure ensures that the population does not grow too much and aphids are removed from poorly performing edges. The results show a preference for a higher kill rate $A_k = 0.8$, but not killing all aphids, which leaves a small portion of carry-over aphids from previous states.

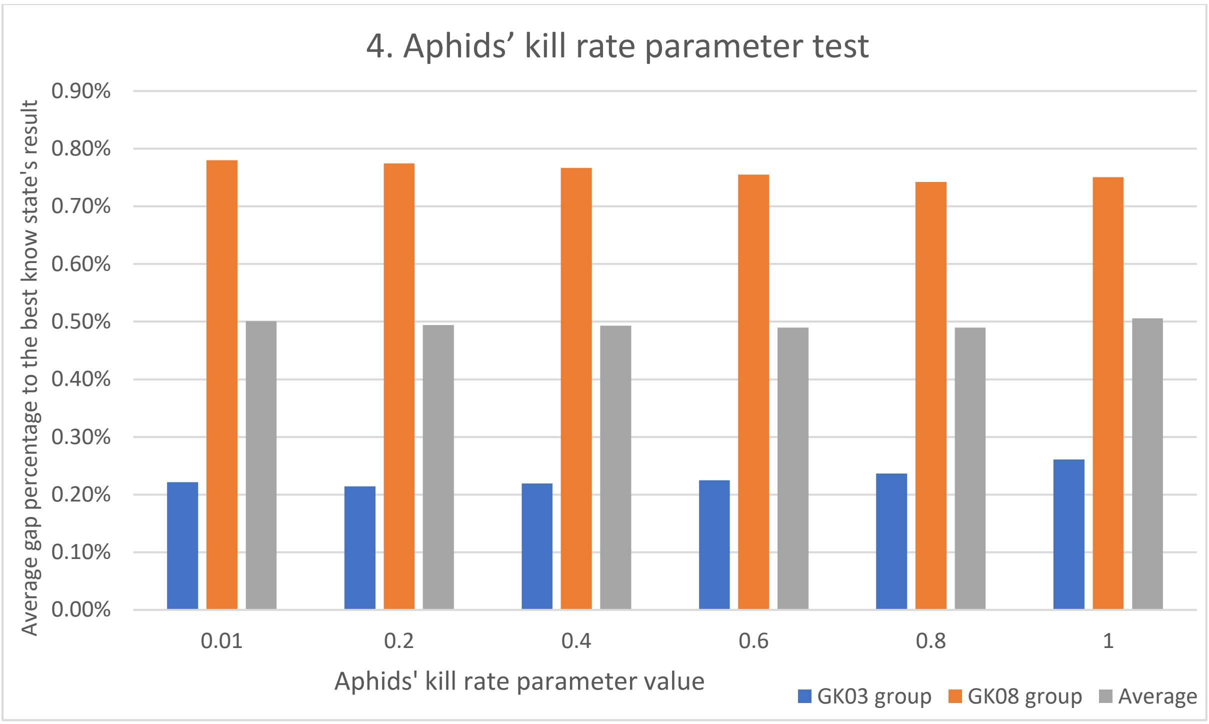

Figure 5-6: ACO with Aphids hyper-parameter tuning test number 4. Aphids' kill rate parameter test. The results show the best dynamic optimization performance is achieved using $\boldsymbol{A_k = 0.8}$.

In summary of ACO with Aphids hyper-parameter tuning, the Aphids specific hyper-parameters have been tested to find the best configuration to optimize the DMKP problem. The configuration: $A_r = 2,\ A_h = 1,\ A_l = 1,\ A_k = 0.8$ has been shown to perform the best. This ACO with Aphids hyper-parameter configuration will be used for the next experiment comparing ACO with aphids against the other two most popular ACO dynamic optimization strategies.

5.3.2. ACO with Aphids comparison with other ACO dynamic optimization strategies result

This experiment is dedicated to a full comparison of ACO with Aphids dynamic optimization algorithm strategy with the other two most common dynamic optimization strategies for ACO algorithm, Full-Restart and Pheromone-Sharing strategies. All strategies are tested using 55 fully-defined DMKP benchmark datasets with eleven dataset complexity groups from smallest GK01 to largest GK11 and five dynamism levels from SAM-0.01 to SAM-0.2. All three dynamic optimization strategies are implemented on the same ACO search core, and the experiments are conducted on the isolated test system. Therefore, the only test variables are the dynamic optimization strategies. The statistical results' significance is proven with acquired each data point by running the dynamic optimization ten times.

In this experiment, the DMKP benchmark datasets provide a wide range of complexity and dynamism levels for testing the dynamic optimization strategies. The complexity of the datasets is determined by the number of items and knapsacks involved, with GK01 representing the simplest scenario and GK11 representing the most complex. GK01 includes 100 items and 15 knapsacks, offering a relatively straightforward scenario for the strategies to handle. This represents a relatively small-scale problem scenario that might be encountered in practical applications such as small business inventory management or simple resource allocation tasks. On the other end of the spectrum, GK11 presents a more complicated scenario with 2500 items and 100 knapsacks. This scenario poses a significant challenge to the optimization strategies due to the larger search space and the increased complexity in finding optimal solutions. Situations mirroring this complexity could be found in larger-scale operations such as supply chain management for large corporations or large-scale project scheduling. The dynamism levels of the datasets range from SAM-0.01 to SAM-0.2. SAM-0.01 corresponds to a maximum interstate dynamism level of 1%. This implies that up to 1% of the problem parameters may change as the problem evolves, representing a scenario with relatively minor changes. This could mirror real-world situations where changes in the problem parameters are infrequent or subtle. SAM-0.2, on the other hand, corresponds to a maximum interstate dynamism level of 20%. This means that up to 20% of the problem parameters can change, representing a highly dynamic scenario. This could reflect real-world situations where the problem parameters can change significantly and frequently, such as in dynamic market conditions or unstable operational environments. By using these datasets, the experiment aims to assess the performance of the dynamic optimization strategies across a broad range of complexity and dynamism levels, mirroring the various conditions they might encounter in real-world applications. This comprehensive approach ensures the robustness and applicability of the experimental results.

In Table 5-3, the results show an average result gap of each dynamic optimization strategy for every dynamic benchmark dataset. The table also includes the average summary per dataset group and dynamism level, as well as the total average of the dynamic optimization strategy. Overall, the ACO with Aphids has shown the best performance, with an average result gap of 0.519%. The second best was the Pheromone-Sharing strategy, with an average gap of 0.733%. Lastly, the Full-Restart strategy achieved an average result gap of 1.092%. In relative terms, these results show that the ACO with Aphids strategy performed 110% better than the Full-Restart strategy and 41% better than the Pheromone-Sharing strategy.

Table 5-3: Dynamic optimization average result gap of all optimization strategies. Each data point is an average of all dynamic states' result gap over ten algorithm runs. (Lower is better)

Average results gap		Dynamism level	Dataset group											Average
			GK01	GK02	GK03	GK04	GK05	GK06	GK07	GK08	GK09	GK10	GK11	
Strategy	ACO with Aphids	SAM-0.01	0.158%	0.160%	0.117%	0.171%	0.124%	0.159%	0.339%	0.429%	0.694%	0.652%	0.563%	0.324%
		SAM-0.02	0.160%	0.191%	0.125%	0.160%	0.139%	0.189%	0.294%	0.437%	0.640%	0.696%	0.694%	0.339%
		SAM-0.05	0.215%	0.259%	0.173%	0.241%	0.177%	0.335%	0.355%	0.595%	0.756%	0.902%	0.927%	0.449%
		SAM-0.1	0.283%	0.347%	0.233%	0.454%	0.322%	0.539%	0.553%	0.951%	0.969%	1.208%	1.142%	0.636%
		SAM-0.2	0.282%	0.407%	0.281%	0.566%	0.391%	0.708%	0.900%	1.354%	1.413%	1.641%	1.371%	0.847%
		Average	0.220%	0.273%	0.186%	0.319%	0.231%	0.386%	0.488%	0.753%	0.894%	1.020%	0.939%	**0.519%**
	Full-Restart	SAM-0.01	0.142%	0.199%	0.481%	0.933%	0.867%	1.100%	1.622%	1.777%	2.231%	2.011%	1.661%	1.184%
		SAM-0.02	0.124%	0.191%	0.425%	0.876%	0.766%	1.040%	1.520%	1.822%	2.285%	2.266%	1.890%	1.200%
		SAM-0.05	0.149%	0.194%	0.373%	0.799%	0.630%	0.969%	1.399%	1.649%	2.234%	2.134%	1.584%	1.101%
		SAM-0.1	0.166%	0.177%	0.329%	0.735%	0.655%	0.916%	1.254%	1.504%	1.985%	1.896%	1.358%	0.998%
		SAM-0.2	0.127%	0.163%	0.325%	0.683%	0.590%	0.888%	1.198%	1.547%	1.876%	1.915%	1.433%	0.977%
		Average	0.142%	0.185%	0.387%	0.805%	0.701%	0.983%	1.399%	1.660%	2.122%	2.044%	1.585%	**1.092%**
	Pheromone-Sharing	SAM-0.01	0.180%	0.196%	0.211%	0.236%	0.184%	0.263%	0.227%	0.454%	0.750%	0.882%	1.137%	0.429%
		SAM-0.02	0.190%	0.219%	0.218%	0.260%	0.190%	0.364%	0.268%	0.529%	0.805%	1.050%	1.229%	0.484%
		SAM-0.05	0.276%	0.332%	0.283%	0.345%	0.225%	0.486%	0.420%	0.851%	1.082%	1.271%	1.300%	0.625%
		SAM-0.1	0.337%	0.453%	0.377%	0.612%	0.381%	0.799%	0.831%	1.419%	1.938%	1.720%	1.344%	0.928%
		SAM-0.2	0.340%	0.511%	0.471%	0.819%	0.497%	1.062%	1.565%	1.933%	2.490%	2.056%	1.457%	1.200%
		Average	0.265%	0.342%	0.312%	0.454%	0.295%	0.595%	0.662%	1.037%	1.413%	1.396%	1.293%	**0.733%**

Furthermore, the result gap standard deviation of each experiment is shown in Table 5-4 to disprove a null hypothesis. Out of all tested dynamic optimization strategies, the Full-Restart strategy had the lowest overall standard deviation of just 0.0104%, the ACO with Aphids strategy had a larger overall standard deviation of 0.0195%, and the Pheromone-Sharing strategy had the largest overall standard deviation of 0.0330%. Statistical significance and rejection of the null hypothesis can be proved using the two-sample unpaired t-test considering the result magnitude, standard deviation, and sample size [216]. ACO with Aphids and Full-Restart strategies sample separations result in a T-value of 81.9 and a P-value $< 10^{-6}$. ACO with Aphids and Pheromone-Sharing strategies sample separations result in a T-value of 17.7 and a P-value $<$

10^{-6}. Both t-test groups reject the null hypothesis and show exceptionally statistically significant sample separations.

Table 5-4: Dynamic optimization result gap standard deviation of all optimization strategies. Each data point is a standard deviation of the dynamic optimization result gap with a sample size of 10 runs.

Result gap standard deviation		Dynamism level	Dataset group											Average
			GK01	GK02	GK03	GK04	GK05	GK06	GK07	GK08	GK09	GK10	GK11	
Strategy	ACO with Aphids	SAM-0.01	0.032%	0.030%	0.022%	0.034%	0.020%	0.025%	0.012%	0.015%	0.046%	0.012%	0.010%	0.026%
		SAM-0.02	0.019%	0.027%	0.017%	0.018%	0.014%	0.021%	0.022%	0.013%	0.012%	0.014%	0.008%	0.018%
		SAM-0.05	0.020%	0.027%	0.012%	0.022%	0.011%	0.019%	0.012%	0.012%	0.022%	0.019%	0.012%	0.018%
		SAM-0.1	0.010%	0.020%	0.020%	0.012%	0.014%	0.029%	0.009%	0.012%	0.012%	0.011%	0.009%	0.016%
		SAM-0.2	0.017%	0.025%	0.015%	0.010%	0.011%	0.046%	0.014%	0.013%	0.007%	0.011%	0.007%	0.019%
		Average	0.021%	0.026%	0.017%	0.021%	0.015%	0.030%	0.014%	0.013%	0.024%	0.014%	0.009%	**0.020%**
	Full-Restart	SAM-0.01	0.008%	0.005%	0.011%	0.011%	0.017%	0.014%	0.007%	0.006%	0.007%	0.004%	0.003%	0.009%
		SAM-0.02	0.008%	0.009%	0.011%	0.012%	0.019%	0.008%	0.006%	0.005%	0.010%	0.006%	0.003%	0.010%
		SAM-0.05	0.007%	0.010%	0.010%	0.015%	0.026%	0.006%	0.006%	0.008%	0.009%	0.007%	0.002%	0.011%
		SAM-0.1	0.007%	0.008%	0.011%	0.016%	0.024%	0.014%	0.011%	0.010%	0.004%	0.005%	0.002%	0.012%
		SAM-0.2	0.010%	0.012%	0.008%	0.010%	0.019%	0.006%	0.008%	0.006%	0.007%	0.006%	0.004%	0.010%
		Average	0.008%	0.009%	0.010%	0.013%	0.021%	0.010%	0.008%	0.007%	0.008%	0.006%	0.003%	**0.010%**
	Pheromone-Sharing	SAM-0.01	0.036%	0.041%	0.035%	0.047%	0.019%	0.055%	0.014%	0.019%	0.102%	0.026%	0.022%	0.045%
		SAM-0.02	0.020%	0.026%	0.040%	0.040%	0.020%	0.041%	0.019%	0.028%	0.021%	0.042%	0.016%	0.030%
		SAM-0.05	0.023%	0.026%	0.016%	0.028%	0.012%	0.055%	0.021%	0.044%	0.018%	0.057%	0.012%	0.032%
		SAM-0.1	0.017%	0.030%	0.018%	0.035%	0.007%	0.019%	0.025%	0.029%	0.029%	0.054%	0.005%	0.028%
		SAM-0.2	0.014%	0.023%	0.025%	0.023%	0.013%	0.034%	0.029%	0.032%	0.023%	0.052%	0.006%	0.027%
		Average	0.023%	0.030%	0.028%	0.036%	0.015%	0.043%	0.022%	0.031%	0.050%	0.048%	0.014%	**0.033%**

The average gap slip results show how much performance decreases on average after each dynamic change. The gap slip is calculated using the first iteration's result of the current state minus the last iteration result before the dynamic change has occurred. The average gap slip measurements are shown in Table 5-5 of each dynamic optimization strategy for every dynamic benchmark dataset. Interestingly, the Pheromone-Sharing strategy achieved the best average gap slip of only 0.240%. Meanwhile, the ACO with Aphids strategy has achieved an average gap slip of 0.304%. Lastly, the Full-Restart strategy achieved the worst average gap slip of 1.420%.

Table 5-5: Dynamic optimization average gap slip of all optimization strategies. Each data point is an average of all dynamic states gap slip over ten algorithm runs. (Lower is better)

Average gap slip		Dynamism level	Dataset group											Average
			GK01	GK02	GK03	GK04	GK05	GK06	GK07	GK08	GK09	GK10	GK11	
Strategy	ACO with Aphids	SAM-0.01	0.050%	0.073%	0.103%	0.125%	0.155%	0.167%	0.128%	0.108%	0.013%	0.014%	0.005%	0.086%
		SAM-0.02	0.101%	0.134%	0.169%	0.177%	0.185%	0.192%	0.146%	0.121%	0.038%	0.027%	0.009%	0.118%
		SAM-0.05	0.263%	0.288%	0.339%	0.339%	0.292%	0.304%	0.197%	0.182%	0.073%	0.043%	0.026%	0.213%
		SAM-0.1	0.563%	0.587%	0.607%	0.553%	0.532%	0.506%	0.334%	0.307%	0.178%	0.121%	0.043%	0.394%
		SAM-0.2	1.062%	1.090%	1.116%	0.877%	0.956%	0.759%	0.684%	0.475%	0.410%	0.273%	0.074%	0.707%
		Average	0.408%	0.434%	0.467%	0.414%	0.424%	0.386%	0.298%	0.239%	0.142%	0.096%	0.031%	**0.304%**
	Full-Restart	SAM-0.01	2.730%	2.607%	2.400%	1.785%	2.049%	1.465%	1.333%	0.685%	0.474%	0.232%	0.060%	1.438%
		SAM-0.02	2.645%	2.566%	2.390%	1.805%	2.220%	1.486%	1.474%	0.827%	0.542%	0.260%	0.078%	1.481%
		SAM-0.05	2.518%	2.528%	2.337%	1.704%	2.101%	1.407%	1.700%	0.904%	0.709%	0.349%	0.071%	1.484%
		SAM-0.1	2.355%	2.369%	2.174%	1.550%	1.916%	1.290%	1.608%	0.816%	0.698%	0.318%	0.061%	1.378%
		SAM-0.2	2.242%	2.248%	2.085%	1.552%	1.861%	1.254%	1.467%	0.786%	0.641%	0.308%	0.061%	1.319%
		Average	2.498%	2.464%	2.277%	1.679%	2.029%	1.380%	1.516%	0.804%	0.613%	0.293%	0.066%	**1.420%**
	Pheromone-Sharing	SAM-0.01	0.041%	0.052%	0.044%	0.047%	0.054%	0.043%	0.063%	0.077%	0.042%	0.043%	0.027%	0.048%
		SAM-0.02	0.090%	0.096%	0.080%	0.084%	0.083%	0.079%	0.067%	0.082%	0.048%	0.047%	0.032%	0.072%
		SAM-0.05	0.204%	0.203%	0.199%	0.208%	0.211%	0.181%	0.117%	0.133%	0.077%	0.071%	0.039%	0.149%
		SAM-0.1	0.477%	0.476%	0.466%	0.434%	0.461%	0.365%	0.263%	0.226%	0.146%	0.119%	0.043%	0.316%
		SAM-0.2	1.179%	1.044%	0.995%	0.707%	0.895%	0.555%	0.565%	0.314%	0.247%	0.178%	0.054%	0.612%
		Average	0.398%	0.374%	0.357%	0.296%	0.341%	0.244%	0.215%	0.166%	0.112%	0.092%	0.039%	**0.240%**

The performance of dynamic dataset group results is visually compared in Figure 5-7. For small benchmark dataset group instances GK01 and GK02, the Full-Restart strategy performs the best with an average result gap of 0.14% and 0.18%, respectively. This strategy solves each optimization state from the start, and information learned from previous states does not negatively impact algorithm convergence. The ACO with Aphids strategy demonstrated superior performance in comparison to Full-Restart and Pheromone-Sharing strategies for all larger instances GK03 through GK11. Additionally, the ACO with Aphids strategy outperforms the Pheromone-Sharing strategy for every dynamic dataset group.

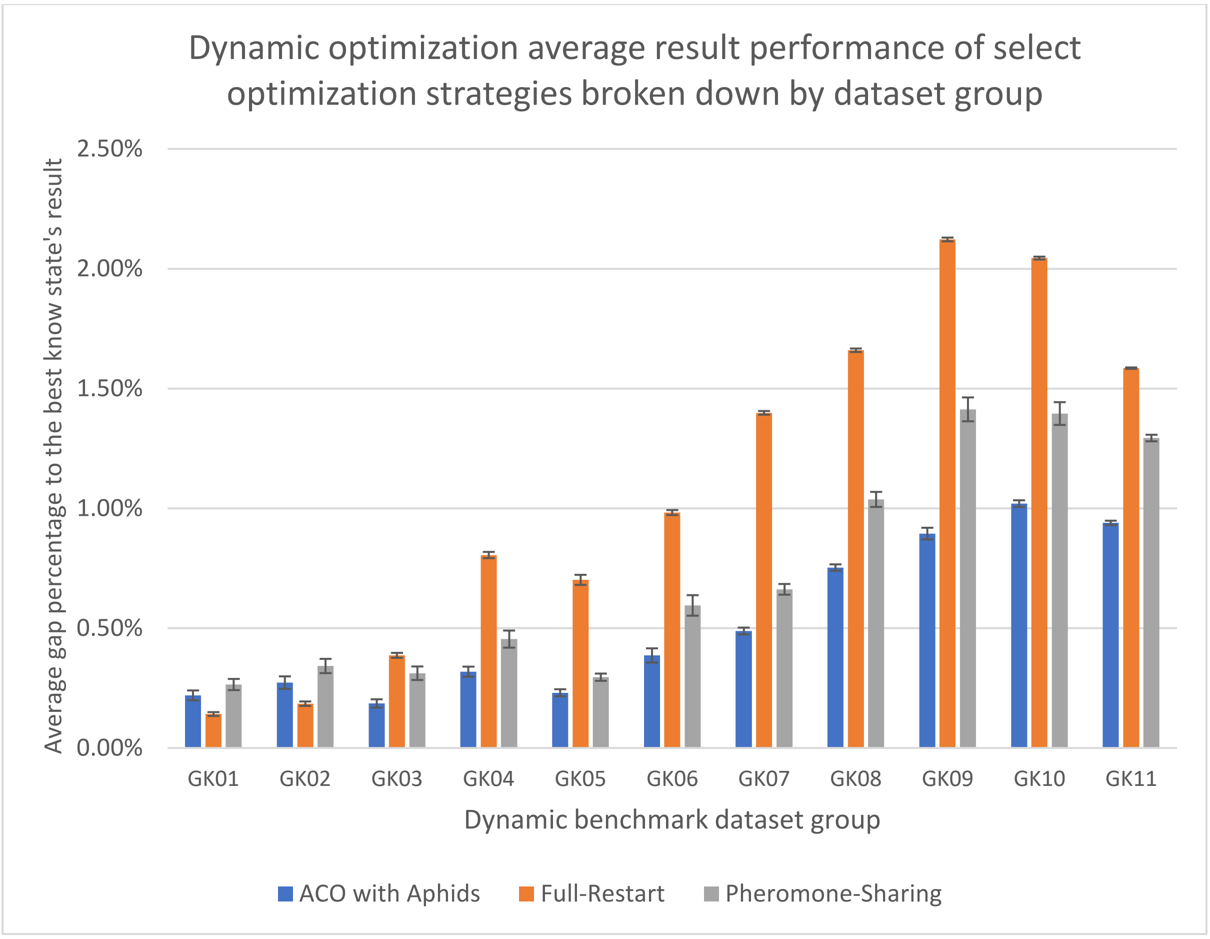

Figure 5-7: Dynamic optimization average performance of each dynamic optimization strategy averaged per dataset group. Each data point is an average result gap of all five dynamism levels run ten times. Error bars indicate the standard deviation of experiment results.

The average performance of the state's first and last iterations across all dynamic dataset groups are compared in Figure 5-8. The upper mark indicates the average result gap achieved within the first iteration after the dynamic change, and the lower mark is the average final result achieved before the dynamic change occurs. The first iteration of the Full-Restart strategy for all dataset groups has the worst result gap of 2.51%. This behaviour is expected because the Full-Restart strategy represents the worst-case scenario where the information is not carried from one dynamic state to the next, and each state has to converge independently. However, the Full-Restart strategy also improves the result gap the most by an average of 1.42%. The first iteration result gap of the ACO with Aphids strategy is, on average, 0.83%, which is only slightly better than the result gap of the Pheromone-Sharing strategy of 0.99%. However, the improvement of the ACO with Aphids strategy from the first iteration to the last by 0.31% is more significant than the improvement of the Pheromone-Sharing strategy by 0.25% for every dataset group. This indicates that aphids help ants adapt to a new dynamic environment quicker, and the starting pheromone is less localized to an outdated solution. The benefit of using aphids is compounded for especially large benchmark dataset groups, like GK08 and larger.

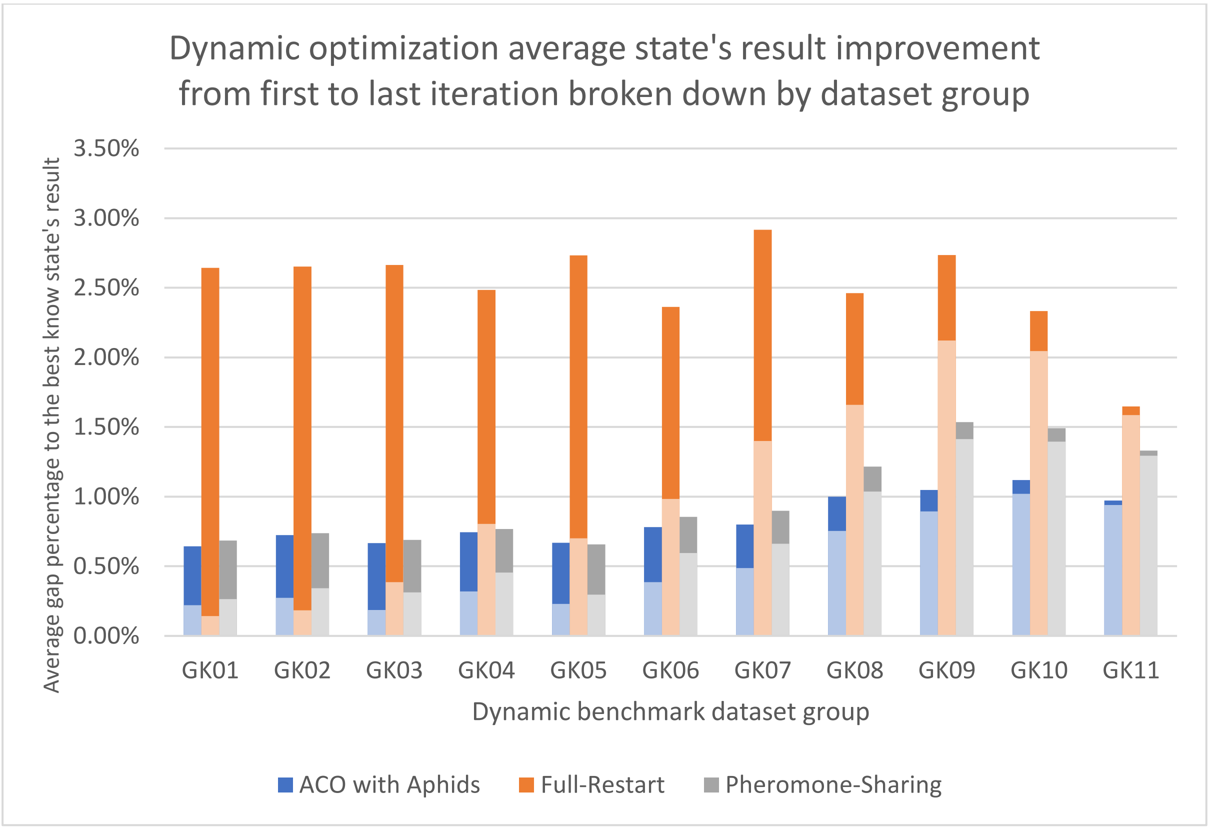

Figure 5-8: Dynamic optimization average state's result improvement from the first to the last iteration of select optimization strategies broken down by dataset group. Each result data point is an average result gap of all five dynamism levels run ten times.

Selected optimization strategies are also compared by performance for all dynamism levels in Figure 5-9. The Full-Restart strategy performs almost equally well for all dynamism levels, with an average result gap of 1.09%. This behaviour is expected as the Full-Restart strategy solves each dynamic state independently, and the dynamism level has no impact on algorithm performance. Then for both ACO with Aphids and Pheromone-Sharing strategies, a lower dynamism level allows for better performance because previously found solutions are changed to a lower degree and are more up-to-date. On the flip side, the larger dynamism level hurts the dynamic optimization performance of ACO with Aphids and Pheromone-Sharing strategies. At the highest dynamism level, $\Delta = 0.2$ Pheromone-Sharing strategy is no longer beneficial over the Full-Restart strategy. Furthermore, the ACO with Aphids strategy consistently outperforms the Pheromone-Sharing strategy across all dynamism levels, on average 29.2% lower result gap, the lowest reduction for SAM-0.01 dynamism level is 24.4%, and the highest reduction for SAM-0.1 dynamism level is 31.4%.

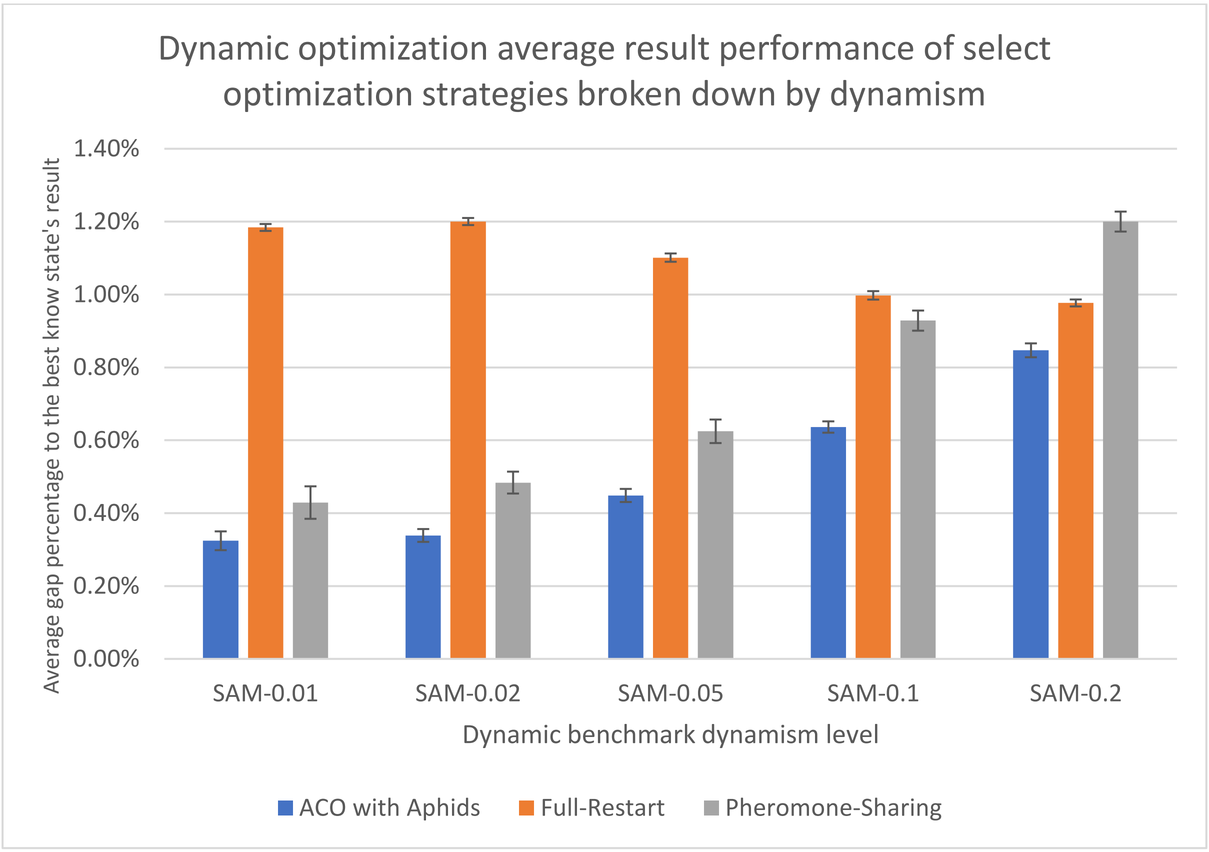

Figure 5-9: Dynamic optimization average performance of each dynamic optimization strategy averaged per dynamism. Each data point is an average result gap of all 11 dataset groups run ten times. Error bars indicate the standard deviation of experiment results.

The average performance of the first and last iteration of the state across all dynamism levels is compared in Figure 5-10. Similarly to Figure 5-8, the upper mark indicates the average result gap achieved within the first iteration after the dynamic change has occurred, and the lower mark is the average final result achieved before the dynamic change occurs. As expected for the Full-Restart strategy, dynamism has almost no impact on the first iteration’s performance after the dynamic change because each state is solved independently. ACO with Aphids and Pheromone-Sharing strategies, higher dynamism causes more prominent performance degradation after the dynamic change. However, on average, the overall first iteration performance of the ACO with Aphids strategy is better by 15.6% than the Pheromone-Sharing strategy.

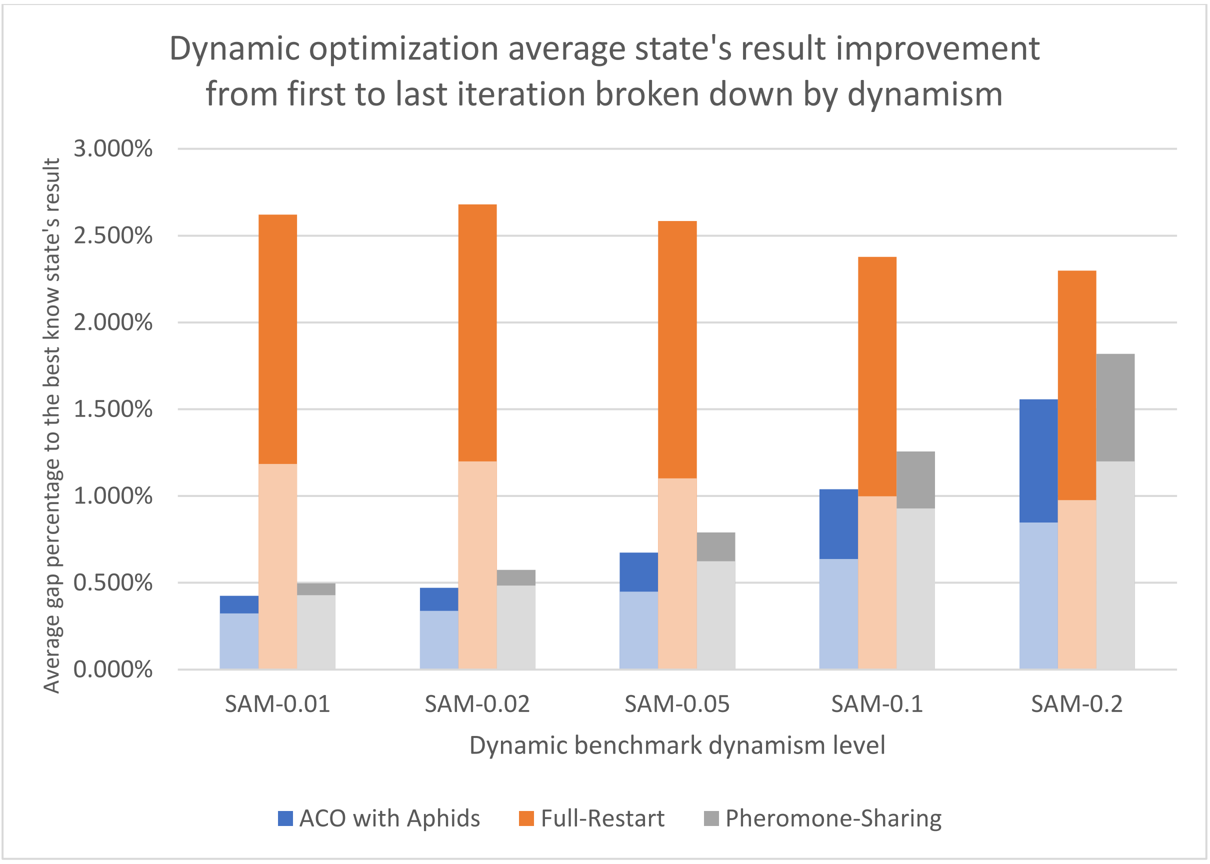

Figure 5-10: Dynamic optimization average state's result improvement from the first to the last iteration of select optimization strategies broken down by dynamism. Each result data point is an average result gap of all 11 dataset groups run ten times.

Finally, the aggregate convergence through all states is compared in Figure 5-11. Each convergence line represents the average convergence of all tested benchmark datasets. The average convergence is calculated from the optimization result with a normalized optimization time to account for differences in the time given for each state's optimization based on benchmark dataset complexity. The Pheromone-Sharing strategy and ACO with Aphids strategy reach equilibrium over the first few states where subsequent state solutions results are no closer to best-know solution results than previous states' solutions. Meanwhile, the Full-Restart strategy does not show any inter-state convergence.

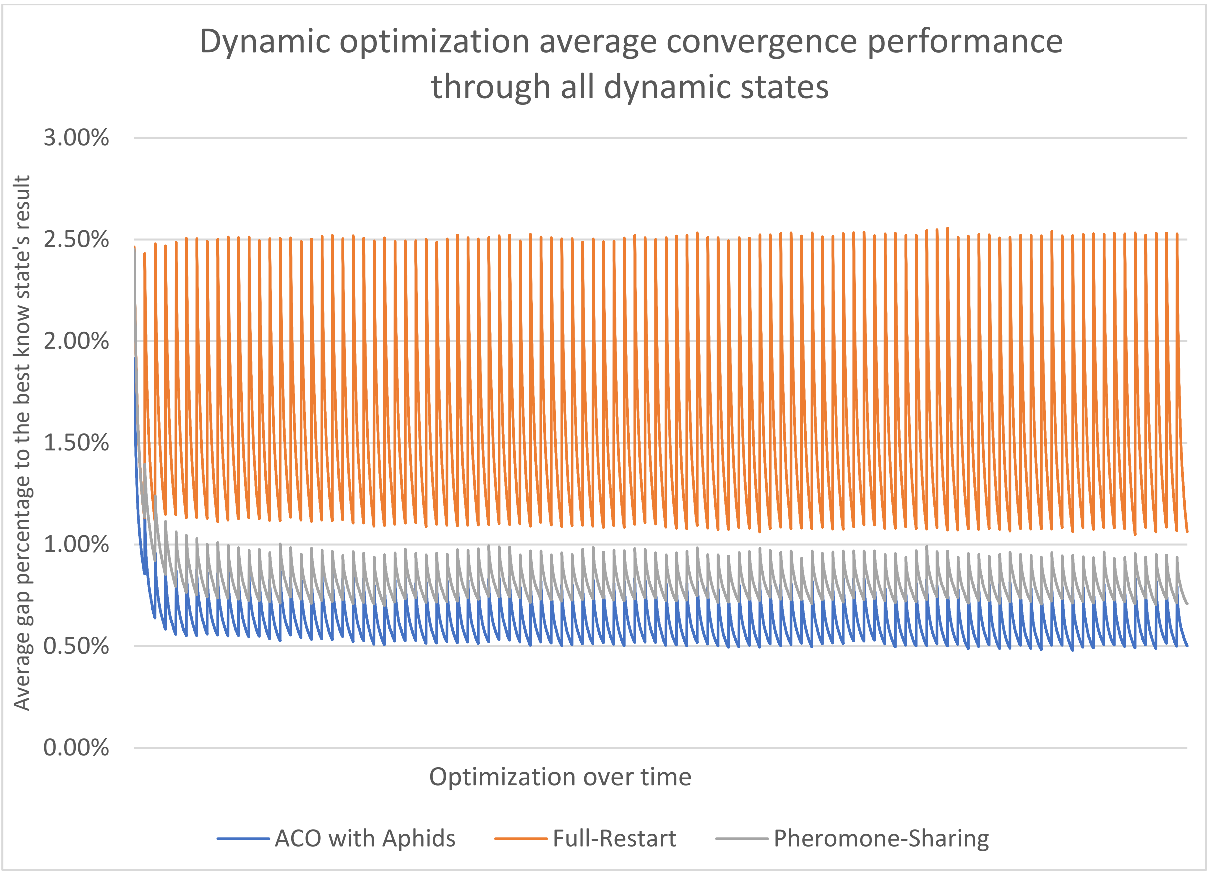

Figure 5-11: Dynamic optimization average convergence performance through all dynamic states. Each convergence performance result is an average convergence of all 55 benchmark datasets run ten times.

For a clearer view, Figure 5-12 shows the same aggregate convergence as in Figure 5-11, but only through the first ten states. The ACO with Aphids strategy starts with significantly better first state optimization results than Full-Restart and Pheromone-Sharing strategies. This first state's convergence improvement is caused by a high aphids' relocation rate, which occurs in every state, including the first. Then as predicted in Figure 2-6, the Full-Restart strategy converges almost equally for every state. The Pheromone-Sharing strategy has a minimal result gap slip and a reduced convergence slope. Finally, ACO with Aphids strategy has reasonably good result gap slip while maintaining an excellent convergence slope after each dynamic change.

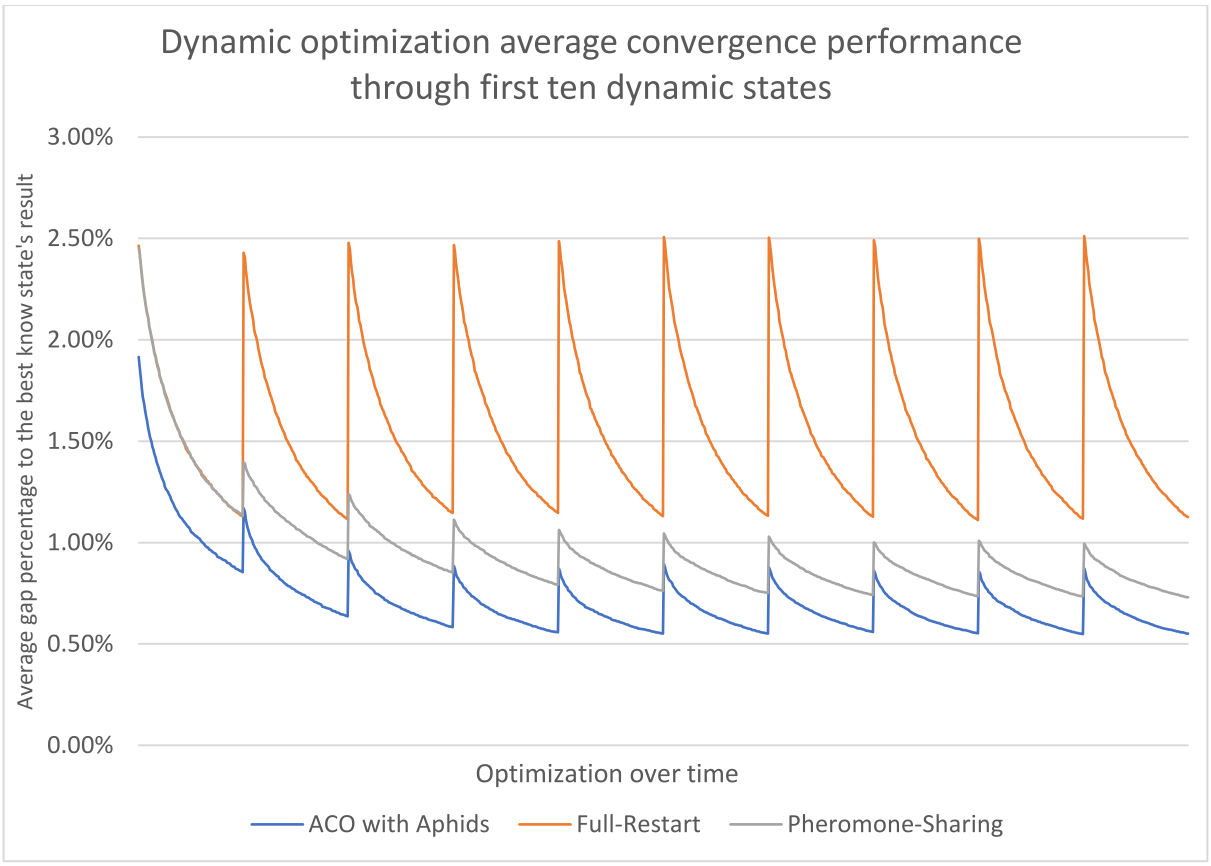

Figure 5-12: Dynamic optimization average convergence performance through the first ten dynamic states. Each convergence performance result is an average convergence of all 55 benchmark datasets run ten times.

In summary, ACO with Aphids has outperformed the Pheromone-Sharing strategy and has proven especially beneficial for large optimization problems. This demonstrates how adding aphids to the ACO algorithm improves the dynamic performance of large optimization problems with limited time allowed to converge. Also, for the smallest optimization problems Full-Restart strategy has performed better than dedicated strategies for dynamic optimization, as it was accurately predicted by previous research [157].

5.4. Chapter Summary

This chapter aimed to solve the trade-off problem offered by currently used rudimentary ACO dynamic optimization strategies. The full-Restart strategy shows a significant penalty to solution quality after each dynamic change, and the Pheromone-Sharing strategy has a reduced slope of convergence.

A nature-inspired addition to the Ant Colony Optimization algorithm was introduced to improve its performance for discrete dynamic optimization problems. The proposed method modelled ants' interaction with aphids in the dynamic environment. In the real world, Aphids produce honeydew which is nutritious to ants, and under ants' influence, aphids give up their mobility to increase honeydew production. This nature-inspired interaction between ants and aphids is beneficial for dynamic optimization, where ants control the population of the aphids and placed aphids mediate the information sharing across dynamic states of the optimization problem.

Then ACO with Aphids algorithm has been tested against the two most popular dynamic optimization strategies, Full-Restart and Pheromone-Sharing on Dynamic Multidimensional Knapsack Problem (DMKP). ACO with Aphids has significantly outperformed the Full-Restart strategy for large dataset groups and a limited amount of time to solve each state. On average, the result gap was reduced by 52.5%, which is a 110.5% better performance. Also, ACO with Aphids has outperformed the Pheromone-Sharing strategy in every optimization scenario for all dynamism levels and dataset group sizes. On average, the result gap was reduced by 29.2%, which is a 41.2% better performance. The test results have proved ACO with Aphids superior performance over both Full-Restart and Pheromone-Sharing strategies with rejected null hypothesis, P-value less than 10^{-6}.

Overall, the proposed ACO with Aphids algorithm proved to be a well-rounded, dynamic optimization strategy with a strong ability to adapt to dynamic change and maintain quick convergence. This strong adaptability further compounds through several dynamic states for especially large optimization problems, where positive convergence occurs over multiple dynamic optimization states.

The contributions of this research chapter to science are as follows:

- Introduced a new nature-inspired dynamic optimization strategy for ACO algorithm with improved interstate convergence, called ACO with Aphids. The strategy is modelled by mimicking the real-life symbiotic relationship between ants and aphids.
- Provided the description of ACO with Aphids algorithm with enough detail to make this strategy possible to apply to any dynamic optimization problem.

- Tested and proved the superior performance of ACO with Aphids algorithm solving event-triggered Dynamic Multidimensional Knapsack Problem (DMKP) against two most popular competing strategies: Pheromone-Sharing and Full-Restart.
- The work presented in this chapter has been submitted to the peer-reviewed journal Swarm and Evolutionary Computation, Elsevier [7].

Chapter 6. Conclusions and future work

6.1. Conclusions

This thesis studied Ant Colony Optimization (ACO) algorithm to maximize the efficiency of solving Dynamic Optimization Problems (DOPs). In particular, solving combinatorial DOPs which are much more representative of optimization problems that occur in the real world. ACO algorithm is excellent at solving such problems due to ants' highly adaptable behaviour. The main contribution of this thesis is a High Efficiency Dynamic Combinatorial Optimization System (HEDCOS), which combines improvements made to the ACO algorithm.

This thesis HEDCOS addresses three identified important research gaps in the literature on ACO solving combinatorial DOPs: One, dynamic optimization methods for ACO inter-state convergence are rudimentary. Two, sub-heuristic ACO search methods lack a generalized methodology to improve algorithm convergence. Three, combinatorial dynamic optimization research lacks replicable qualities.

First, this thesis introduced a sub-heuristic ACO search method called Dynamic Impact to improve algorithm convergence for constrained optimization problems. Dynamic Impact is an additional component to the ACO algorithm probability calculation besides pheromone and heuristic information. Similar to heuristic information, Dynamic Impact is a myopic search component, but the value of Dynamic Impact depends on the state of partial solution, which considers non-linear problem fitness and resource consumption. ACO with Dynamic Impact was tested on real-world MMPPFO problem and theoretical MKP problem. For the GK benchmark datasets, ACO with Dynamic Impact results were 4.26 times closer to the best-known or optimal result within the same search efforts. Dynamic Impact methodology is compatible with both static and dynamic optimization problems.

Second, this thesis has resolved a critical gap in the replicability of combinatorial dynamic optimization research and introduced a non-stochastic dynamic dataset generator. Previously, researchers used either private real-world optimization datasets or stochastically generated datasets from existing benchmarks without publishing optimization states or stochastic generator seeds. This dataset generator removed all randomness and used only deterministic methods to generate DMKP states in sequential order. Then, created 1405 fully defined DMKP benchmark instances with 5 dynamism levels using this dataset generation method.

Third, this thesis introduced ACO with Aphids nature-inspired dynamic optimization algorithm. The ACO with Aphids mimics the naturally occurring symbiotic relationship between ants and aphids to improve the performance of discrete dynamic optimization. Aphids help mediate information throughout dynamic optimization and improve overall inter-state convergence. For an accurate comparison, ACO with Aphids, Full-Restart and Pheromone-Sharing strategies were implemented on the ACO with Dynamic Impact algorithm introduced earlier and solved the DMKP benchmark. ACO with Aphids showed superior performance. On average, the result gap was reduced by 52.5% compared to the Full-Restart strategy and 29.2% compared to the Pheromone-Sharing strategy.

Overall, by developing HEDCOS, ACO algorithm combinatorial dynamic optimization performance was improved by a total of 8.99 times compared to the baseline high-performance parallel ACO solving MKP. Moreover, all optimization results are fully replicable, which allows for transparent comparison with future research solutions.

However, while the High Efficiency Dynamic Combinatorial Optimization System (HEDCOS) developed in this thesis showed significant performance improvements in combinatorial dynamic optimization with ACO, the research has certain limitations. The study was primarily validated using the theoretical DMKP benchmark and only partially tested with real-world MMPPFO problem in a static environment. The comparisons of the ACO with Aphids algorithm were limited to only two other ACO dynamic optimization strategies.

6.2. Future work

HEDCOS was researched with real-world combinatorial DOPs in mind. However, it was thoroughly tested with the theoretical DMKP benchmark and only partially with real-world MMPPFO problem in a static environment. Naturally, the next step is to test ACO with Aphids solving a dynamic variant of the MMPPFO problem.

ACO with Aphids algorithm has only been compared to the other two ACO dynamic optimization strategies. In the future, it would be useful to perform dynamic optimization tests to compare ACO with Aphids algorithm to other well-known dynamic optimization algorithms based on GA and PSO.

The baseline ACO algorithm used in this research implements the pheromone logic described by Stützle and Hoos in Min-Max Ant System. However, there is another profoundly different pheromone mediation logic of Population-based ACO described by Guntsch and Middendorf. Theoretically, aphids are compatible with both pheromone mediation types. A competing P-ACO with Aphids algorithm is worth investigating for both real-world applications and solving the DMKP benchmark.

This research has focused on single-objective optimization. However, many real-world problems are multi-objective in nature, involving trade-offs between competing objectives. Developing ACO methods that can handle these multi-objective problems in dynamic environments represents an important future direction.

As the complexity and size of problems in industry continue to grow, the developed algorithms need to keep pace. Future research should focus on scaling these algorithms, testing them on larger and more complex problems. This exploration will not only test the algorithms' limits but also provide valuable insight into their behaviour and performance under high stress and large-scale conditions.

Integrating machine learning, particularly reinforcement learning, could offer a way to enhance ACO algorithm performance. For instance, machine learning could be used to learn the optimal parameters or strategies for the algorithm over time, reducing the need for manual tuning and potentially leading to better results.

Many real-world optimization problems are influenced by real-time data streams such as market prices, sensor data, or user behaviour. Future work could explore how to incorporate such dynamic data into the optimization process, providing solutions that are not just optimal but also timely and context-aware.

Lastly, DMKP benchmark datasets were created using a deterministic state generation methodology where states are created in sequential order based on information within the initial state. The same deterministic dataset generation principles can also be adapted to other types of optimization problems, like the Traveling Salesman Problem or Job-shop Scheduling Problem.

References

[1] J. Skackauskas, T. Kalganova, I. Dear and M. Janakiram, “Dynamic impact for ant colony optimization algorithm,” *Swarm and Evolutionary Computation,* vol. 69, 2022.

[2] J. Skackauskas, “GitHub - Dynamic MKP Datasets Generator,” 2021. [Online]. Available: https://github.com/jonasska/Dynamic-MKP-Datasets-Generator. [Accessed 24 2 2021].

[3] J. Skackauskas, “GitHub - Dynamic MKP Benchmark Datasets,” 2021. [Online]. Available: https://github.com/jonasska/Dynamic-MKP-Benchmark-Datasets. [Accessed 14 4 2021].

[4] J. Skackauskas, “Dynamic-MKP Benchmark Datasets,” *IEEE Dataport,* 2022.

[5] J. Skackauskas, “GitHub - Dynamic MKP Datasets Visualization,” 2021. [Online]. Available: https://github.com/jonasska/Dynamic-MKP-Datasets-Visualization. [Accessed 1 1 2021].

[6] J. Skackauskas and T. Kalganova, “Dynamic Multidimensional Knapsack Problem benchmark datasets,” *Systems and Soft Computing,* vol. 4, 2022.

[7] J. Skackauskas and T. Kalganova, “Herder Ants: Ant Colony Optimization with Aphids for Discrete Event-Triggered Dynamic Optimization Problems,” *Submitted to Swarm and Evolutionary Computation,* 2022.

[8] M. Mavrovouniotis, C. Li and S. Yang, “A survey of swarm intelligence for dynamic optimization: Algorithms and applications,” *Swarm and Evolutionary Computation,* vol. 33, pp. 1 - 17, 2017.

[9] D. Wolpert and W. Macready, “No free lunch theorems for optimization,” *IEEE Transactions on Evolutionary Computation,* vol. 1, no. 1, pp. 67 - 82, 1997.

[10] G. B. McMahon and P. G. Burton, “Flow-Shop Scheduling with the Branch-and-Bound Method,” *Operations research,* vol. 15, no. 3, pp. 473 - 481, 1967.

[11] D. Bertsimas and J. N. Tsitsiklis, “Introduction to linear optimization,” *Athena Scientific,* vol. 7, 1997.

[12] R. Bellman, “Dynamic programming,” *Science,* 1966.

[13] G. Steiner, “On the complexity of dynamic programming for sequencing problems with precedence constraints,” *Annals of operations research,* vol. 26, no. 1-4, pp. 103 - 123, 1990.

[14] J. Branke and H. Schmeck, “Designing Evolutionary Algorithms for Dynamic Optimization Problems,” *Advances in Evolutionary Computing,* pp. 239 - 262, 2003.

[15] C. Arango, P. Cortés, L. Onieva and A. Escudero, “Simulation‐Optimization Models for the Dynamic Berth Allocation Problem,” *Computer‐Aided Civil and Infrastructure Engineering,* vol. 28, no. 10, pp. 769 - 779, 2013.

[16] Y. Liu, X. Ling, Z. Shi, M. LV, J. Fang and L. Zhang, “A Survey on Particle Swarm Optimization Algorithms for Multimodal Function Optimization,” *Journal of Software,* vol. 6, no. 12, 2011.

[17] T. Stützle and H. H. Hoos, Stochastic Local Search, The Morgan Kaufmann Series in Artificial Intelligence, 2005.

[18] M. Mavrovouniotis and S. Yang, “Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors,” *Applied Soft Computing,* vol. 13, no. 10, pp. 4023 - 4037, 2013.

[19] D. Yazdani, J. Branke, M. N. Omidvar, X. Li, C. Li, M. Mavrovouniotis, T. T. Nguyen, S. Yang and X. Yao, “IEEE CEC 2022 Competition on Dynamic Optimization Problems Generated by Generalized Moving Peaks Benchmark,” *arXiv: 2106.06174,* 2021.

[20] X. Li, M. G. Epitropakis, K. Deb and A. Engelbrecht, “Seeking Multiple Solutions: An Updated Survey on Niching Methods and Their Applications,”

IEEE Transactions on Evolutionary Computation, vol. 21, no. 4, pp. 518 - 538, 2017.

[21] Q. Li, J. Zou, S. Yang, J. Zheng and G. Ruan, “A predictive strategy based on special points for evolutionary dynamic multi-objective optimization,” *Soft Computing,* vol. 23, no. 11, pp. 3723 - 3739, 2018.

[22] C. Raquel and X. Yao, “Dynamic Multi-objective Optimization: A Survey of the State-of-the-Art,” *Evolutionary Computation for Dynamic Optimization Problems,* pp. 85 - 106, 2013.

[23] T. T. Nguyen, S. Yang and J. Branke, “Evolutionary dynamic optimization: A survey of the state of the art,” *Swarm and Evolutionary Computation,* vol. 6, pp. 1-24, 2012.

[24] X.-S. Yang, “Nature-inspired optimization algorithms: Challenges and open problems,” *Journal of computational science,* 2020.

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

[26] M. K. Y. Shambour, A. A. Abusnaina and A. I. Alsalibi, “Modified Global Flower Pollination Algorithm and its Application for Optimization Problems,” *Interdisciplinary sciences : computational life sciences,* vol. 11, no. 3, pp. 496 - 507, 2018.

[27] Y. Zheng, X. Lu, M. Zhang and S. Chen, “Optimization Problems and Algorithms,” *Biogeography-Based Optimization: Algorithms and Applications,* pp. 1 - 25, 2018.

[28] Q. Xu, Z. Xu and T. Ma, “A Short Survey and Challenges for Multiobjective Evolutionary Algorithms Based on Decomposition,” *2019 International Conference on Computer, Information and Telecommunication Systems (CITS),* pp. 1 - 5, 2019.

[29] Z. Yao, Y. Jiang, B. Zhao, X. Luo and B. Peng, “A dynamic optimization method for adaptive signal control in a connected vehicle environment,”

Journal of Intelligent Transportation Systems, vol. 24, no. 2, pp. 184 - 200, 2020.

[30] Y. Zhou and X. Liu, “Control Parameterization-Based Adaptive Particle Swarm Approach for Solving Chemical Dynamic Optimization Problems,” *Chemical Engineering & Technology,* vol. 37, no. 4, pp. 692 - 702, 2014.

[31] A. A. Ateya, A. Muthanna, A. Vybornova, A. D. Algarni, A. Abuarqoub, Y. Koucheryavy and A. Koucheryavy, “Chaotic salp swarm algorithm for SDN multi-controller networks,” *Engineering Science and Technology, an International Journal,* vol. 22, no. 4, pp. 1001 - 1012, 2019.

[32] G.-G. Wang, C.-L. Wei, Y. Wang and W. Pedrycz, “Improving distributed anti-flocking algorithm for dynamic coverage of mobile wireless networks with obstacle avoidance,” *Knowledge-based systems,* vol. 225, p. 107133, 2021.

[33] J. Eaton, S. Yang and M. Gongora, “Ant Colony Optimization for Simulated Dynamic Multi-Objective Railway Junction Rescheduling,” *IEEE Transactions on Intelligent Transportation Systems,* vol. 18, no. 11, pp. 2980 - 2992, 2017.

[34] S. Ali, K.-u.-R. Khoumbati and D. N. Hakro, “Multi-Hop Optimization in Wireless Sensor Networks using Genetic Algorithm,” *University of Sindh Journal of Information and Communication Technology,* vol. 3, no. 4, pp. 193 - 197, 2019.

[35] M. H. Afshar, “A new transition rule for ant colony optimization algorithms: application to pipe network optimization problems,” *Engineering optimization,* vol. 37, no. 5, pp. 525 - 540, 2005.

[36] 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.

[37] Y. Marinakis and M. Marinaki, “A hybrid genetic – Particle Swarm Optimization Algorithm for the vehicle routing problem,” *Expert systems with applications,* vol. 37, no. 2, pp. 1446 - 1455, 2010.

[38] F. T. Chan, R. Bhagwat, N. Kumar, M. Tiwari and P. Lam, “Development of a decision support system for air-cargo pallets loading problem: A case study,” *Expert systems with applications,* vol. 31, no. 3, pp. 472 - 485, 2006.

[39] 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.

[40] R. Z. Farahani and M. Elahipanah, “A genetic algorithm to optimize the total cost and service level for just-in-time distribution in a supply chain,” *International journal of production economics,* vol. 111, no. 2, pp. 229 - 243, 2008.

[41] X.-b. Liu and X.-m. Sun, “A multi-improved genetic algorithm for facility layout optimisation based on slicing tree,” *International journal of production research,* vol. 50, no. 18, pp. 5173 - 5180, 2012.

[42] A. Sadrzadeh, “A genetic algorithm with the heuristic procedure to solve the multi-line layout problem,” *Computers & industrial engineering,* vol. 62, no. 4, pp. 1055 - 1064, 2012.

[43] J. M. Palomo-Romero, L. Salas-Morera and L. García-Hernández, “An island model genetic algorithm for unequal area facility layout problems,” *Expert systems with applications,* vol. 68, pp. 151 - 162, 2017.

[44] Y.-H. Perng, Y.-K. Juan and H.-S. Hsu, “Genetic algorithm-based decision support for the restoration budget allocation of historical buildings,” *Building and environment,* vol. 42, no. 2, pp. 770 - 778, 2007.

[45] T.-c. Fu, C.-p. Chung and F.-l. Chung, “Adopting genetic algorithms for technical analysis and portfolio management,” *Computers & mathematics with applications,* vol. 66, no. 10, pp. 1743 - 1757, 2013.

[46] P. C. Gilmore and R. E. Gomory, “The Theory and Computation of Knapsack Functions,” *Operations research,* vol. 14, no. 6, pp. 1045 - 1074, 1966.

[47] D. Yska, Y. Mei and M. Zhang, "Genetic Programming Hyper-Heuristic with Cooperative Coevolution for Dynamic Flexible Job Shop Scheduling," *Genetic Programming,* pp. 306 - 321, 2018.

[48] Y. Fang, C. Peng, P. Lou, Z. Zhou, J. Hu and J. Yan, "Digital-Twin-Based Job Shop Scheduling Toward Smart Manufacturing," *IEEE transactions on industrial informatics,* vol. 15, no. 12, pp. 6425 - 6435, 2019.

[49] T. Jamrus, C.-F. Chien, M. Gen and K. Sethanan, "Hybrid Particle Swarm Optimization Combined With Genetic Operators for Flexible Job-Shop Scheduling Under Uncertain Processing Time for Semiconductor Manufacturing," *IEEE transactions on semiconductor manufacturing,* vol. 31, no. 1, pp. 32 - 41, 2018.

[50] F. Zhang, Y. Mei, S. Nguyen and M. Zhang, "Evolving Scheduling Heuristics via Genetic Programming With Feature Selection in Dynamic Flexible Job-Shop Scheduling," *IEEE transactions on cybernetics,* vol. 51, no. 4, pp. 1797 - 1811, 2021.

[51] M. Fera, F. Fruggiero, A. Lambiase, R. Macchiaroli and V. Todisco, "A modified genetic algorithm for time and cost optimization of an additive manufacturing single-machine scheduling," *International journal of industrial engineering computations,* vol. 9, no. 4, pp. 423 - 438, 2018.

[52] R. Liu, X. Xie, K. Yu and Q. Hu, "A survey on simulation optimization for the manufacturing system operation," *International journal of modelling & simulation,* vol. 38, no. 2, pp. 116 - 127, 2018.

[53] P. Udhayakumar and S. Kumanan, "Sequencing and scheduling of job and tool in a flexible manufacturing system using ant colony optimization algorithm," *International journal of advanced manufacturing technology,* vol. 50, no. 9-12, pp. 1075 - 1084, 2010.

[54] Z. Wang, M. Rahman, Y. Wong and J. Sun, "Optimization of multi-pass milling using parallel genetic algorithm and parallel genetic simulated annealing,"

International journal of machine tools & manufacture, vol. 45, no. 15, pp. 1726 - 1734, 2005.

[55] P. Pongcharoen, W. Promtet, P. Yenradee and C. Hicks, “Stochastic Optimisation Timetabling Tool for university course scheduling,” *International journal of production economics,* vol. 112, no. 2, pp. 903 - 918, 2008.

[56] A. A. Gozali and S. Fujimura, “Solving University Course Timetabling Problem Using Multi-Depth Genetic Algorithm,” *SHS web of conferences,* vol. 77, 2020.

[57] N. N. H. Almaalei and N. A. M. R. Siti, “REVIEW OF ACO ALGORITHM ON NETWORK AND SCHEDULING PROBLEM,” *Compusoft,* vol. 8, no. 7, pp. 3250-3260, 2019.

[58] G. Pinto, I. Ainbinder and G. Rabinowitz, “A genetic algorithm-based approach for solving the resource-sharing and scheduling problem,” *Computers & industrial engineering,* vol. 57, no. 3, pp. 1131 - 1143, 2009.

[59] A. R. Kavitha and C. Chellamuthu, “Brain tumour segmentation from MRI image using genetic algorithm with fuzzy initialisation and seeded modified region growing (GFSMRG) method,” *The imaging science journal,* vol. 64, no. 5, pp. 285 - 297, 2016.

[60] H.-L. Yang and Q.-F. Lin, “Opinion mining for multiple types of emotion-embedded products/services through evolutionary strategy,” *Expert systems with applications,* vol. 99, pp. 44 - 55, 2018.

[61] M. Dragoni, “An Evolutionary Strategy For Concept-Based Multi-Domain Sentiment Analysis,” *IEEE computational intelligence magazine,* vol. 14, no. 2, pp. 18 - 27, 2019.

[62] S. Ghosh and S. Bhattacharya, “A data-driven understanding of COVID-19 dynamics using sequential genetic algorithm based probabilistic cellular automata,” *Applied soft computing,* vol. 96, p. 106692, 2020.

[63] D. J. Munk, G. A. Vio and G. P. Steven, “Topology and shape optimization methods using evolutionary algorithms: a review,” *Structural and multidisciplinary optimization,* vol. 52, no. 3, pp. 613 - 631, 2015.

[64] M. H. Abolbashari and S. Keshavarzmanesh, “On various aspects of application of the evolutionary structural optimization method for 2d and 3d continuum structures,” *Finite elements in analysis and design,* vol. 42, no. 6, pp. 478 - 491, 2006.

[65] M. Castelli, L. Trujillo, L. Vanneschi and A. Popovič, “Prediction of energy performance of residential buildings: A genetic programming approach,” *Energy and buildings,* vol. 102, pp. 67 - 74, 2015.

[66] A. Zameer, J. Arshad, A. Khan and M. A. Z. Raja, “Intelligent and robust prediction of short term wind power using genetic programming based ensemble of neural networks,” *Energy conversion and management,* vol. 134, pp. 361 - 372, 2017.

[67] G. Mauša and T. Galinac Grbac, “Co-evolutionary multi-population genetic programming for classification in software defect prediction: An empirical case study,” *Applied soft computing,* vol. 55, pp. 331 - 351, 2017.

[68] B. Tran, B. Xue and M. Zhang, “Genetic programming for feature construction and selection in classification on high-dimensional data,” *Memetic computing,* vol. 8, no. 1, pp. 3 - 15, 2015.

[69] M. Suganuma, S. Shirakawa and T. Nagao, “A genetic programming approach to designing convolutional neural network architectures,” *Proceedings of the Genetic and Evolutionary Computation Conference,* pp. 497 - 504, 2017.

[70] D. Hein, S. Udluft and T. A. Runkler, “Interpretable policies for reinforcement learning by genetic programming,” *Engineering applications of artificial intelligence,* vol. 76, pp. 158 - 169, 2018.

[71] T. Uhlig and O. Rose, “Simulation-based optimization for groups of cluster tools in semiconductor manufacturing using simulated annealing,”

Proceedings of the 2011 Winter Simulation Conference (WSC), pp. 1852 - 1863, 2011.

[72] S. L. Rosen and C. M. Harmonosky, “An improved simulated annealing simulation optimization method for discrete parameter stochastic systems,” *Computers & operations research,* vol. 32, no. 2, pp. 343 - 358, 2005.

[73] S.-W. Lin, C.-Y. Cheng, P. Pourhejazy and K.-C. Ying, “Multi-temperature simulated annealing for optimizing mixed-blocking permutation flowshop scheduling problems,” *Expert systems with applications,* vol. 165, 2021.

[74] W. Bożejko, Z. Hejducki and M. Wodecki, “Applying metaheuristic strategies in construction projects management,” *Journal of civil engineering and management,* vol. 18, no. 5, pp. 621 - 630, 2012.

[75] C. Lee, “A review of applications of genetic algorithms in operations management,” *Engineering applications of artificial intelligence,* vol. 76, pp. 1 - 12, 2018.

[76] M. Savelsbergh and T. Volgenant, “Edge exchanges in the degree-constrained minimum spanning tree problem,” *Computers & operations research,* vol. 12, no. 4, pp. 341 - 348, 1985.

[77] W. Rand and R. Riolo, “Measurements for understanding the behavior of the genetic algorithm in dynamic environments: a case study using the shaky ladder hyperplane-defined functions,” *Proceedings of the 7th annual workshop on Genetic and evolutionary computation,* pp. 32-38, 2005.

[78] 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.

[79] M. Mavrovouniotis and S. Yang, “A memetic ant colony optimization algorithm for the dynamic travelling salesman problem,” *Soft Computing,* vol. 15, no. 7, pp. 1405 - 1425, 2011.

[80] J. Branke, “Evolutionary optimization in dynamic environments,” *Springer Science & Business Media,* vol. 3, 2012.

[81] R. Chen, K. Li and X. Yao, “Dynamic Multiobjectives Optimization With a Changing Number of Objectives,” *IEEE Transactions on Evolutionary Computation,* vol. 22, no. 1, pp. 157 - 171, 2018.

[82] M. E. Breaban and A. Iftene, “Dynamic Objective Sampling in Many-objective Optimization,” *Procedia Computer Science,* vol. 60, pp. 178 - 187, 2015.

[83] S. Guan, Q. Chen and W. Mo, “Evolving dynamic multiple-objective optimization problems with objective replacement,” *Springer,* 2005.

[84] H. Lu and W. Chen, “Dynamic-objective particle swarm optimization for constrained optimization problems,” *Journal of Combinatorial Optimization,* vol. 12, no. 4, pp. 409 - 419, 2006.

[85] Q. Jiang, L. Wang, Y. Lin, X. Hei, G. Yu and X. Lu, “An efficient multi-objective artificial raindrop algorithm and its application to dynamic optimization problems in chemical processes,” *Applied Soft Computing,* vol. 58, pp. 354 - 377, 2017.

[86] L. Huang, I. Suh and A. Abraham, “ Dynamic multi-objective optimization based on membrane computing for control of time-varying unstable plants.,” *Information Sciences,* vol. 181, no. 11, pp. 2370-2391, 2011.

[87] F. B. Ozsoydan and A. Baykasoğlu, “Quantum firefly swarms for multimodal dynamic optimization problems,” *Expert Systems With Applications,* vol. 115, pp. 189 - 199, 2019.

[88] R. Liu, X. Song, L. Fang and More..., “An r-dominance-based preference multi-objective optimization for many-objective optimization,” *Soft Computing,* vol. 21, no. 17, pp. 5003 - 5024, 09/2017.

[89] A. Sharifi, J. Kazemi Kordestani, M. Mahdaviani and M. R. Meybodi, “A novel hybrid adaptive collaborative approach based on particle swarm optimization

and local search for dynamic optimization problems," *Applied Soft Computing,* vol. 32, pp. 432 - 448, 2015.

[90] T. Friedrich, T. Kroeger and F. Neumann, "Weighted preferences in evolutionary multi-objective optimization," *Machine Learning and Cybernetics,* vol. 4, no. 2, pp. 139 - 148, 04/2013.

[91] H. Zhang and G.-G. Wang, "Improved NSGA-III using transfer learning and centroid distance for dynamic multi-objective optimization," *Complex & intelligent systems,* 2021.

[92] S. Rostami and A. Shenfield, "A multi-tier adaptive grid algorithm for the evolutionary multi-objective optimisation of complex problems," *Soft Computing,* vol. 21, no. 17, pp. 4963 - 4979, 09/2017.

[93] G. Li, G.-G. Wang, J. Dong, W.-C. Yeh and K. Li, "DLEA: A dynamic learning evolution algorithm for many-objective optimization," *Information sciences,* vol. 574, pp. 567 - 589, 2021.

[94] H. Abbass, R. Sarker and C. Newton, "PDE: a Pareto-frontier differential evolution approach for multi-objective optimization problems," *Evolutionary Computation,* vol. 2, pp. 971-978, 2001.

[95] D. Yazdani, M. N. Omidvar, J. Branke, T. T. Nguyen and X. Yao, "Scaling Up Dynamic Optimization Problems: A Divide-and-Conquer Approach," *IEEE Transactions on Evolutionary Computation,* vol. 24, no. 1, pp. 1 - 15, 2020.

[96] H. Fu, B. Sendhoff, K. Tang and X. Yao, "Robust Optimization Over Time: Problem Difficulties and Benchmark Problems," *IEEE Transactions on Evolutionary Computation,* vol. 19, no. 5, pp. 731 - 745, 2015.

[97] A. Ş. Uyar, "Experimental Comparison of Replacement Strategies in Steady State Genetic Algorithms for the Dynamic MKP," *Applications of Evolutinary Computing,* vol. 4448, pp. 647 - 656, 2007.

[98] Ł. Strąk, R. Skinderowicz and U. Boryczka, “Adjustability of a discrete particle swarm optimization for the dynamic TSP,” *Soft Computing,* vol. 22, no. 22, pp. 7633 - 7648, 2018.

[99] Ł. Strąk, R. Skinderowicz, U. Boryczka and A. Nowakowski, “A Self-Adaptive Discrete PSO Algorithm with Heterogeneous Parameter Values for Dynamic TSP,” *Entropy (Basel, Switzerland),* vol. 21, no. 8, 2019.

[100] M. N. Omidvar, X. Li and K. Tang, “Designing benchmark problems for large-scale continuous optimization,” *Information sciences,* pp. 419 - 436, 2015.

[101] A. Ahrari, S. Elsayed, R. Sarker and D. Essam, “A New Prediction Approach for Dynamic Multiobjective Optimization,” *IEEE Congress on Evolutionary Computation,* pp. 2268 - 2275, 2019.

[102] S. Das, S. Maity, B.-Y. Qu and P. Suganthan, “Real-parameter evolutionary multimodal optimization — A survey of the state-of-the-art,” *Swarm and evolutionary computation,* pp. 71 - 88, 2011.

[103] K. N. Kaipa and D. Ghose, “Multimodal Function Optimization,” *Glowworm Swarm Optimization. Studies in Computational Intelligence,* vol. 698, 2017.

[104] K. Deb, L. Thiele, M. Laumanns and E. Zitzler, “Scalable multi-objective optimization test problems,” *Proceedings of the 2002 Congress on Evolutionary Computation,* vol. 1, pp. 825 - 830, 2002.

[105] A. Ahrari, S. Elsayed, R. Sarker, D. Essam and C. A. Coello Coello, “Adaptive Multilevel Prediction Method for Dynamic Multimodal Optimization,” *IEEE Transactions on Evolutionary Computation,* vol. 25, no. 3, pp. 463 - 477, 2021.

[106] W. Luo, X. Lin, T. Zhu and P. Xu, “A clonal selection algorithm for dynamic multimodal function optimization,” *Swarm and Evolutionary Computation,* vol. 50, 2019.

[107] S. Cheng, H. Lu, Y.-n. Guo, X. Lei, J. Liang, J. Chen and Y. Shi, “Dynamic multimodal optimization: A preliminary study,” *IEEE Congress on Evolutionary Computation,* pp. 279 - 285, 2019.

[108] G. Reinelt, “TSPLIB--A Traveling Salesman Problem Library,” *INFORMS journal on computing,* vol. 3, no. 4, pp. 376 - 384, 1991.

[109] N. Ouertani, H. B. Ramdhan, S. Krichen, I. Nouaouri and H. Allaoui, “A New Evolutionary Method to Deal with the Dynamic Vehicle Routing Problem,” *2018 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD),* pp. 1 - 5, 2018.

[110] H. Fisher and G. Thompson, “Probabilistic learning combinations of local job-shop scheduling rules,” *Englewood Cliffs,* pp. 225-251, 1963.

[111] S. Lawrence, “Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques,” *Technical Report, GSIA, Carnegie Mellon University,* 1984.

[112] T. s. b. p. f. j.-s. scheduling, “Adams, Joseph; Balas, Egon; Zawack, Daniel,” *Management science,* vol. 34, no. 3, pp. 391 - 401, 1988.

[113] M. Dorigo, “Optimization, Learning and Natural Algorithms,” 1992.

[114] B. Bullnheimer, R. F. Hartl and C. Strauß, “A new rank based version of the Ant System. A computational study,” 1997.

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

[116] 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.

[117] M. Guntsch and M. Middendorf, “A Population Based Approach for ACO,” *Applications of Evolutionary Computing,* pp. 72 - 81, 2002.

[118] T. Stützle and H. H. Hoos, “MAX–MIN ant system,” *Future generation computer systems,* vol. 16, no. 8, pp. 889-914, 2000.

[119] M. Dorigo and L. M. Gambardella, “Ant colonies for the travelling salesman problem,” *BioSystems,* vol. 43, no. 2, pp. 73 - 81, 1997.

[120] M. Guntsch and M. Middendorf, “Applying population based ACO to dynamic optimization problems.,” *International Workshop on Ant Algorithms,* pp. 111-122, 2002.

[121] J. Eaton and S. Yang, “Dynamic railway junction rescheduling using population based ant colony optimisation,” *2014 14th UK Workshop on Computational Intelligence (UKCI),* pp. 1 - 8, 2014.

[122] 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.

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

[124] L. Gambardella and M. Dorigo, “Solving symmetric and asymmetric TSPs by ant colonies,” *Proceedings of IEEE International Conference on Evolutionary Computation,* pp. 622 - 627, 1996.

[125] T. Bui and B. Moon, “A new genetic approach for the traveling salesman problem,” *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence,* vol. 1, pp. 7 - 12, 1994.

[126] T. Kötzing, F. Neumann, H. Röglin and C. Witt, “Theoretical analysis of two ACO approaches,” *Swarm intelligence,* vol. 6, no. 1, pp. 1 - 21, 2011.

[127] 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.

[128] S. Tsutsui, “Ant colony optimisation for continuous domains with aggregation pheromones metaphor,” *Proceedings of the The 5th International Conference on Recent Advances in Soft Computing,* pp. 207-212, 2004.

[129] K. Socha and M. Dorigo, “Ant colony optimization for continuous domains,” *European journal of operational research,* vol. 185, no. 3, pp. 1155 - 1173, 2008.

[130] 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.

[131] 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.

[132] 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.

[133] 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.

[134] 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.

[135] Z.-G. Ren, Z.-R. Feng, L.-J. Ke and Z.-J. Zhang, “New ideas for applying ant colony optimization to the set covering problem,” *Computers & industrial engineering,* vol. 58, no. 4, pp. 774 - 784, 2010.

[136] 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.

[137] 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.

[138] S. J. SHYU, B. M. T. LIN and T.-S. HSIAO, "Ant colony optimization for the cell assignment problem in PCS networks," *Computers & operations research,* vol. 33, no. 6, pp. 1713 - 1740, 2006.

[139] S. Fidanova and P. Pop, "An improved hybrid ant-local search algorithm for the partition graph coloring problem," *Journal of computational and applied mathematics,* vol. 293, pp. 55 - 61, 2016.

[140] D. Coltorti and A. Rizzoli, "Ant colony optimization for real-world vehicle routing problems," *SIGEVOlution,* vol. 2, no. 2, pp. 2 - 9, 2007.

[141] G.-F. Deng and W.-T. Lin, "Ant colony optimization-based algorithm for airline crew scheduling problem," *Expert systems with applications,* vol. 38, no. 5, pp. 5787 - 5793, 2011.

[142] C. Colson, M. Nehrir and C. Wang, "Ant colony optimization for microgrid multi-objective power management," *IEEE/PES Power Systems Conference and Exposition,* pp. 1 - 7, 2009.

[143] M. Marzband, E. Yousefnejad, A. Sumper and J. L. Domínguez-García, "Real time experimental implementation of optimum energy management system in standalone Microgrid by using multi-layer ant colony optimization," *International journal of electrical power & energy systems,* vol. 75, pp. 265 - 274, 2016.

[144] G. Li, L. Boukhatem and J. Wu, "Adaptive Quality-of-Service-Based Routing for Vehicular Ad Hoc Networks With Ant Colony Optimization," *IEEE transactions on vehicular technology,* vol. 66, no. 4, pp. 3249 - 3264, 2017.

[145] Z. Sun, M. Wei, Z. Zhang and G. Qu, "Secure Routing Protocol based on Multi-objective Ant-colony-optimization for wireless sensor networks," *Applied soft computing,* vol. 77, p. 2019, 366 - 375.

[146] C. Silva, J. Sousa, T. Runkler and J. Sá da Costa, “Distributed supply chain management using ant colony optimization,” *European journal of operational research,* vol. 199, no. 2, pp. 349 - 358, 2009.

[147] 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.

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

[149] 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.

[150] 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.

[151] 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.

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

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

[154] S. Zhang and T. N. Wong, “Flexible job-shop scheduling/rescheduling in dynamic environment: a hybrid MAS/ACO approach,” *International journal of production research,* vol. 55, no. 11, pp. 3173 - 3196, 2017.

[155] A. Prakasam and N. Savarimuthu, “Novel local restart strategies with hyper-populated ant colonies for dynamic optimization problems,” *Neural Computing and Applications,* vol. 31, no. 1, pp. 63 - 76, 2018.

[156] H. Xu, P. Pu and F. Duan, “A Hybrid Ant Colony Optimization for Dynamic Multidepot Vehicle Routing Problem,” *Discrete Dynamics in Nature and Society,* vol. 2018, pp. 1 - 10, 2018.

[157] D. Angus and T. Hendtlass, “Dynamic Ant Colony Optimisation,” *Applied Intelligence,* vol. 23, no. 1, pp. 33 - 38, 2005.

[158] M. Helbig and A. P. Engelbrecht, “Performance measures for dynamic multi-objective optimisation algorithms,” *Information sciences,* vol. 250, pp. 61 - 81, 2013.

[159] A. E. Rizzoli, R. Montemanni, E. Lucibello and L. M. Gambardella, “Ant colony optimization for real-world vehicle routing problems: From theory to applications,” *Swarm Intelligence,* vol. 1, no. 2, pp. 135 - 151, 2007.

[160] A. Donati, R. Montemanni, N. Casagrande and A. Rizzoli, “Time dependent vehicle routing problem with a multi ant colony system.,” *European Journal of Operational Research,* vol. 183, no. 3, pp. 1174-1191, 2008.

[161] W. Xiang and H. Lee, “Ant colony intelligence in multi-agent dynamic manufacturing scheduling,” *Engineering Applications of Artificial Intelligence,* vol. 21, no. 1, pp. 73 - 85, 2008.

[162] M. Krishnan, S. Yun and Y. M. Jung, “Dynamic clustering approach with ACO-based mobile sink for data collection in WSNs,” *Wireless Networks,* vol. 25, no. 8, pp. 4859 - 4871, 2018.

[163] M. Samà, P. Pellegrini, A. D’Ariano, J. Rodriguez and D. Pacciarelli, “Ant colony optimization for the real-time train routing selection problem,” *Transportation Research Part B: Methodological,* vol. 85, pp. 89-108, 2016.

[164] W. Meilin, Z. Xiangwei, D. Qingyun and H. Jinbin, “A dynamic schedule methodology for discrete job shop problem based on Ant Colony

Optimization," *2010 2nd IEEE International Conference on Information Management and Engineering,* pp. 306 - 309, 2010.

[165] J. Eaton, S. Yang and M. Mavrovouniotis, "Ant colony optimization with immigrants schemes for the dynamic railway junction rescheduling problem with multiple delays," *Soft Computing,* vol. 20, no. 8, pp. 2951 - 2966, 2016.

[166] J. Offenberg, "Balancing between Mutualism and Exploitation: The Symbiotic Interaction between Lasius Ants and Aphids," *Behavioral Ecology and Sociobiology,* vol. 49, no. 4, pp. 304 - 310, 2001.

[167] M. K. Fischer and A. W. Shingleton, "Host Plant and Ants Influence the Honeydew Sugar Composition of Aphids," *Functional Ecology,* vol. 15, no. 4, pp. 544 - 550, 2001.

[168] C. Lang and F. Menzel, "Lasius niger ants discriminate aphids based on their cuticular hydrocarbons," *Animal Behaviour,* vol. 82, no. 6, pp. 1245 - 1254, 2011.

[169] H. Sakata, "How an ant decides to prey on or to attend aphids," *Population Ecology,* vol. 36, no. 1, pp. 45 - 51, 1994.

[170] N. Eslami, S. Yazdani, M. Mirzaei and E. Hadavandi, "Aphid–Ant Mutualism: A novel nature-inspired metaheuristic algorithm for solving optimization problems," *Mathematics and Computers in Simulation,* vol. 201, pp. 362 - 395, 2022.

[171] J.-i. Kushida, A. Hara, T. Takahama and N. Mimura, "Cartesian ant programming introducing symbiotic relationship between ants and aphids," *2017 IEEE 10th International Workshop on Computational Intelligence and Applications,* pp. 115 - 120, 2017.

[172] N. Hara, Y. Shirasaki, S. Shimomura, Y. Uwate and Y. Nishio, "Combinatorial Optimization by Cooperative Mechanism of Ant Colony and Aphid," *International Workshop on Nonlinear Circuits, Communications and Signal Processing,* 2012.

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

[174] P. Chu and J. Beasley, “A Genetic Algorithm for the Multidimensional Knapsack Problem,” *Journal of heuristics,* vol. 4, no. 1, pp. 63 - 86, 1998.

[175] J. Puchinger, G. R. Raidl and U. Pferschy, “The Multidimensional Knapsack Problem: Structure and Algorithms,” *INFORMS journal on computing,* vol. 22, no. 2, pp. 250 - 265, 2010.

[176] 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.

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

[178] 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].

[179] M. J. Varnamkhasti and L. S. Lee, “A Genetic Algorithm Based on Sexual Selection for the Multidimensional 0/1 Knapsack Problems.,” *International journal of modern physics. Conference series,* vol. 9, pp. 422 - 431, 2012.

[180] X. Liu, F. Xiang and J. Mao, “An improved method for solving the large-scale multidimensional 0-1 knapsack problem,” *2014 International Conference on Electronics and Communication Systems (ICECS),* pp. 1 - 6, 2014.

[181] L. Ke, Z. Feng, Z. Ren and X. Wei, “An ant colony optimization approach for the multidimensional knapsack problem,” *Journal of heuristics,* vol. 16, no. 1, pp. 65 - 83, 2010.

[182] R. T. Liu and X. J. Lv, “MapReduce-Based Ant Colony Optimization Algorithm for Multi-Dimensional Knapsack Problem,” *Applied mechanics and materials,* Vols. 380-384, pp. 1877 - 1880, 2013.

[183] H. Fingler, E. N. Cáceres, H. Mongelli and S. W. Song, “A CUDA based Solution to the Multidimensional Knapsack Problem Using the Ant Colony Optimization,” *Procedia computer science,* vol. 29, pp. 84 - 94, 2014.

[184] W.-N. Chen, J. Zhang, H. Chung, W.-L. Zhong, W.-G. Wu and Y.-h. Shi, “A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems,” *IEEE transactions on evolutionary computation,* vol. 14, no. 2, pp. 278 - 300, 2010.

[185] A. Gherboudj, S. Labed and S. Chikhi, “A New Hybrid Binary Particle Swarm Optimization Algorithm for Multidimensional Knapsack Problem,” *Advances in Computer Science, Engineering & Applications,* pp. 489 - 498, 2012.

[186] M. Chih, “Three pseudo-utility ratio-inspired particle swarm optimization with local search for multidimensional knapsack problem,” *Swarm and evolutionary computation,* vol. 39, pp. 279 - 296, 2018.

[187] 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.

[188] 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.

[189] 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.

[190] J. Skackauskas, “Microchip Manufacturing Plant Production Floor Optimization (MMPPFO) problem,” Figshare, 2020.

[191] 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.

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

[193] 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.

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

[195] 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.

[196] 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.

[197] 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.

[198] 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.

[199] 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.

[200] 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.

[201] 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.

[202] J. Branke, M. Orbayı and Ş. Uyar, "The Role of Representations in Dynamic Knapsack Problems," *Applications of Evolutionary Computing,* pp. 764 - 775, 2006.

[203] C. Groba, A. Sartal and X. H. Vázquez, "Solving the dynamic traveling salesman problem using a genetic algorithm with trajectory prediction: An application to fish aggregating devices," *Computers & Operations Research,* vol. 56, pp. 22 - 32, 2015.

[204] B. H. Nguyen, B. Xue, P. Andreae and M. Zhang, "A New Binary Particle Swarm Optimization Approach: Momentum and Dynamic Balance Between Exploration and Exploitation," *IEEE Transactions on Cybernetics,* vol. 51, no. 2, pp. 589 - 603, 2021.

[205] Y. Feng, G.-G. Wang and L. Wang, "Solving randomized time-varying knapsack problems by a novel global firefly algorithm," *Engineering with Computers,* vol. 34, no. 3, pp. 621 - 635, 2017.

[206] Y. Feng, G.-G. Wang, S. Deb, M. Lu and X.-J. Zhao, "Solving 0–1 knapsack problem by a novel binary monarch butterfly optimization," *Neural Computing and Applications,* vol. 28, no. 7, pp. 1619 - 1634, 2015.

[207] Y. Feng, G.-G. Wang and X.-Z. Gao, "A Novel Hybrid Cuckoo Search Algorithm with Global Harmony Search for 0-1 Knapsack Problems,"

International Journal of Computational Intelligence Systems, vol. 9, no. 6, p. 1174, 2016.

[208] J. Cao, B. Yin, X. Lu, Y. Kang and X. Chen, “A modified artificial bee colony approach for the 0-1 knapsack problem,” *Applied intelligence,* vol. 48, no. 6, pp. 1582 - 1595, 2017.

[209] Y. Feng and G.-G. Wang, “A binary moth search algorithm based on self-learning for multidimensional knapsack problems,” *Future Generation Computer Systems,* vol. 126, pp. 48 - 64, 2022.

[210] B. Abdollahzadeh, S. Barshandeh, H. Javadi and N. Epicoco, “An enhanced binary slime mould algorithm for solving the 0–1 knapsack problem,” *Engineering with Computers,* 2021.

[211] A. Baykasoğlu and F. B. Ozsoydan, “Evolutionary and population-based methods versus constructive search strategies in dynamic combinatorial optimization,” *Information Sciences,* vol. 420, pp. 159 - 183, 2017.

[212] Google, “GitHub - or-tools,” Google, 2015. [Online]. Available: https://github.com/google/or-tools. [Accessed 16 12 2021].

[213] J. Skackauskas, “Dynamic MKP Benchmark Best Known Results,” 6 12 2021. [Online]. Available: https://github.com/jonasska/Dynamic-MKP-Benchmark-Best-Known-results. [Accessed 7 12 2021].

[214] D. Angus and T. Hendtlass, “Ant Colony Optimisation Applied to a Dynamically Changing Problem,” *Developments in Applied Artificial Intelligence,* pp. 618 - 627, 2002.

[215] M. Mavrovouniotis, I. S. Bonilha, F. M. Muller, G. Ellinas and M. Polycarpou, “Effective ACO-Based Memetic Algorithms for Symmetric and Asymmetric Dynamic Changes,” *IEEE Congress on Evolutionary Computation,* pp. 2567 - 2574, 2019.

[216] N. Cressie and H. Whitford, “How to use the two sample t-test,” *Biometrical Journal,* vol. 28, no. 2, pp. 131-148, 1986.