Contents lists available at ScienceDirect

Systems and Soft Computing

journal homepage: www.journals.elsevier.com/soft-computing-letters

Dynamic Multidimensional Knapsack Problem benchmark datasets

Jonas Skackauskas [*], Tatiana Kalganova

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

A R T I C L E I N F O

Keywords:
Dynamic MKP
benchmark datasets
discrete dynamic optimization
non-stochastic dataset generation

A B S T R A C T

With increasing research on solving Dynamic Optimization Problems (DOPs), many metaheuristic algorithms and their adaptations have been proposed to solve them. However, from currently existing research results, it is hard to evaluate the algorithm performance in a repeatable way for combinatorial DOPs due to the fact that each research work has created its own version of a dynamic problem dataset using stochastic methods. Up to date, there are no combinatorial DOP benchmarks with replicable qualities. This work introduces a non-stochastic consistent Dynamic Multidimensional Knapsack Problem (Dynamic MKP) dataset generation method that is also extensible to solve the research replicability problem. Using this method, generated and published 1405 Dynamic MKP benchmark datasets using existing famous static MKP benchmark instances as the initial state. Then the datasets are quantitatively and qualitatively analyzed. Furthermore, 445 datasets have the optimal result found of each state using a linear solver. The optimal results and result scores are included with published datasets.

1. Introduction

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 development is dynamic optimization, where some parameters of the initial problem can be changed that does not entirely invalidate the existing solution. However, some adjustments or fixes are necessary. Dynamic optimization is crucial when some optimization problem details are unknown beforehand or are subject to unexpected changes. The most popular family of algorithms to solve such dynamic optimization problems is the Evolutionary Algorithms (EAs) family [1].

Most 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 therefore 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) [2]. Control parameter optimization using Particle Swarm Optimization (PSO) algorithm [3]. 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 [4]. Distributed Guidance Anti-flocking Algorithm (DGAA) has been applied to Mobile Wireless Network (MWN) optimization [5]. The Ant Colony Optimization (ACO) algorithm has been applied to the Railway Junction Rescheduling problem that aims to solve dynamic multi-objective optimization, minimize time table deviation and energy expenditure [6].

However, there remains one problem with research conducted to solve real-world dynamic optimization problems. 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.

On another front, academic research also is developing new solutions for theoretical dynamic optimization problems. Such work can be categorized into two categories, first solving continuous optimization problems like Moving Peaks Benchmark (MPB), second solving discrete optimization problems like the Traveling Salesman Problem (TSP) where randomness is applied to some aspect of the optimization problem instance.

MPB is a popular continuous domain Dynamic Multi-Objective Optimization Problem (DMOP) with well-defined benchmark suites, like DTLZ [7] and CEC [8]. Many algorithmic improvements have been proposed to tackle the MPB. For example, several combinations of PSO algorithm enhancements were tested to solve and track the optimum of MPB [9]. Several improved versions of EAs are used to solve these

* Corresponding author
E-mail address: 1418820@brunel.ac.uk (J. Skackauskas).

https://doi.org/10.1016/j.sasc.2022.200041
Received 19 September 2021; Received in revised form 5 March 2022; Accepted 26 May 2022
Available online 27 May 2022

problems, like Non-dominated Sorting Genetic Algorithm II (NSGA-II) [10,11], and NSGA-III [12], Pareto Archived Evolution Strategy (PAES) [13], Dynamic Learning Evolution Algorithm (DLEA) [14]. Other dedicated algorithms for continuous dynamic optimization are Differential evolution (DE) [15], Firefly Algorithm (FA) [16], Artificial Immune System (AIS) [17]. Overall scalability and performance of MPB has been explored on multiple algorithms with generated heterogeneous and multimodal benchmarks [18]. Robust Optimization Over Time (ROOT) methodology has been proposed to enable these algorithms solve real world optimization problems too [19].

In the discrete domain of dynamic optimization problems, [20] research work solves Dynamic MKP where item profits, item weights and knapsack capacities are changed with a normally distributed random operator. For a Dynamic TSP [21,22] research work used published TSPLIB library and randomly modified vertex's location, which led to a generation of new problems each time. Then solved the Dynamic TSP using the PSO algorithm. For the Dynamic Vehicle Routing Problem (VRP) [23], 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.

One apparent separation among academic research of 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.

This research attempts to solve the aforementioned issue and create a non-stochastic dataset generation method for a discrete optimization problem. This research provides the 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. 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 seed, 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.

This Dynamic MKP benchmark is a good test suite for testing and comparing the performance of combinatorial optimization algorithms in dynamic environment like Genetic Algorithm (GA) [24], Particle Swarm Optimization (PSO) [25], Ant Colony Optimization (ACO) [26], Firefly Algorithm (FA) [27], Monarch Butterfly Optimization (MBO) [28], Cuckoo Search (CS) [29], Artificial Bee Colony (ABC) [30], Moth Search (MS) [31], Slime Mould Algorithm (SMA) [32].

The contributions of this research 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 [33].
- 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 [34].
- The generated benchmarks are qualitatively and quantitatively analyzed and proven as valid Dynamic MKP datasets. The visualization tool is published on GitHub [35].

Dynamic MKP Datasets

1.1. Dynamic MKP

The multidimensional Knapsack problem is widely used for benchmarking combinatorial optimization algorithms [36]. 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 to develop algorithms to solve the MKP problem. Then, Dynamic Multi-dimensional Knapsack 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.

All existing attempts to solve MKP have involved using static MKP benchmarks as the initial setting and introducing the stochastic changes to some aspect of the optimization problem and process over the time domain. [20] uses normally distributed random operator to change item profits, item weights and knapsack capacities on OR library initial datasets. Meanwhile, the research [37] has dynamic changes in item profits, item weights and knapsack capacities, and all new randomly generated items.

1.2. Dynamic MKP definition

Multidimensional knapsack problem consists of items I and knapsacks K. The items have profit and N-dimensional weight that fills knapsacks. The goal is to choose a set of items with a maximum total profit without exceeding any of the knapsack capacities.

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

$$subject\ to \sum_{i=1}^{n} \left(x_i \times W_{i,k}\right) \leq C_k,\ \forall(k)\ where\ k \in (\mathbb{N} \leq m) \tag{2}$$

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 Dynamic Multidimensional Knapsack Problem 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 notes as $S_t,\ t \in \{0,1,2,\ \ldots\}$ 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.

1.3. Dynamic MKP dataset creation using deterministic state generation method

The approach of 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:

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

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

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

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

where $M(x)$ is a mantissa of the number x, $frac(x)$ is a fractional part of a number, $H3$ 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 "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 $\min P = \min_i P_{0,i}$
- Maximum Profit $\max P = \max_i P_{0,i}$
- Minimum Value $\min V = \min_i \frac{P_{0,i}}{\overline{W_{0,i}}}$
- Maximum Value $\max V = \max_i \frac{P_{0,i}}{\overline{W_{0,i}}}$
- Minimum Weight $\min W = \min_{i,k} W_{0,i,k}$
- Maximum Weight $\max W = \max_{i,k} W_{0,i,k}$
- 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 ith item in the initial state S_0, $\overline{W_{0,i}}$ is the average weight of the ith item in the initial state S_0, $W_{0,i,k}$ is the weight of the ith item for the kth knapsack in the initial state S_0, and finally $C_{0,k}$ is the knapsack capacity of the kth 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 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 Δ parameter. The Δ parameter is also called SAM in the code and charts with no special characters' support.

- Profit adjustment magnitude

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

- Weight adjustment magnitude

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

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 limits of the dataset characteristics.

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

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

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

where, $P_{t+1,i}$ is new item profit for the state S_{t+1} of the ith 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 items average weight and original value: $maxV * \overline{W_{t,i}}$ and $minV * \overline{W_{t,i}}$. The profit adjustment Px value is calculated using profit adjustment multiplier ΔP 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 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 limits of the dataset characteristics.

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

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

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

where, $W_{t+1,i,k}$is new item weight for state S_{t+1} of the ith 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 $\frac{P_{t,i}}{maxV} - \overline{W_{t,i}}$ and $\overline{W_{t,i}} - \frac{P_{t,i}}{minV}$ such that as a result of new weight, the value of the item does not go over the limits of the initial dataset. $\frac{P_{t,i}}{maxV} - \overline{W_{t,i}}$ is items profit over the maximum value that gives minimum weight, and removing average weight gives

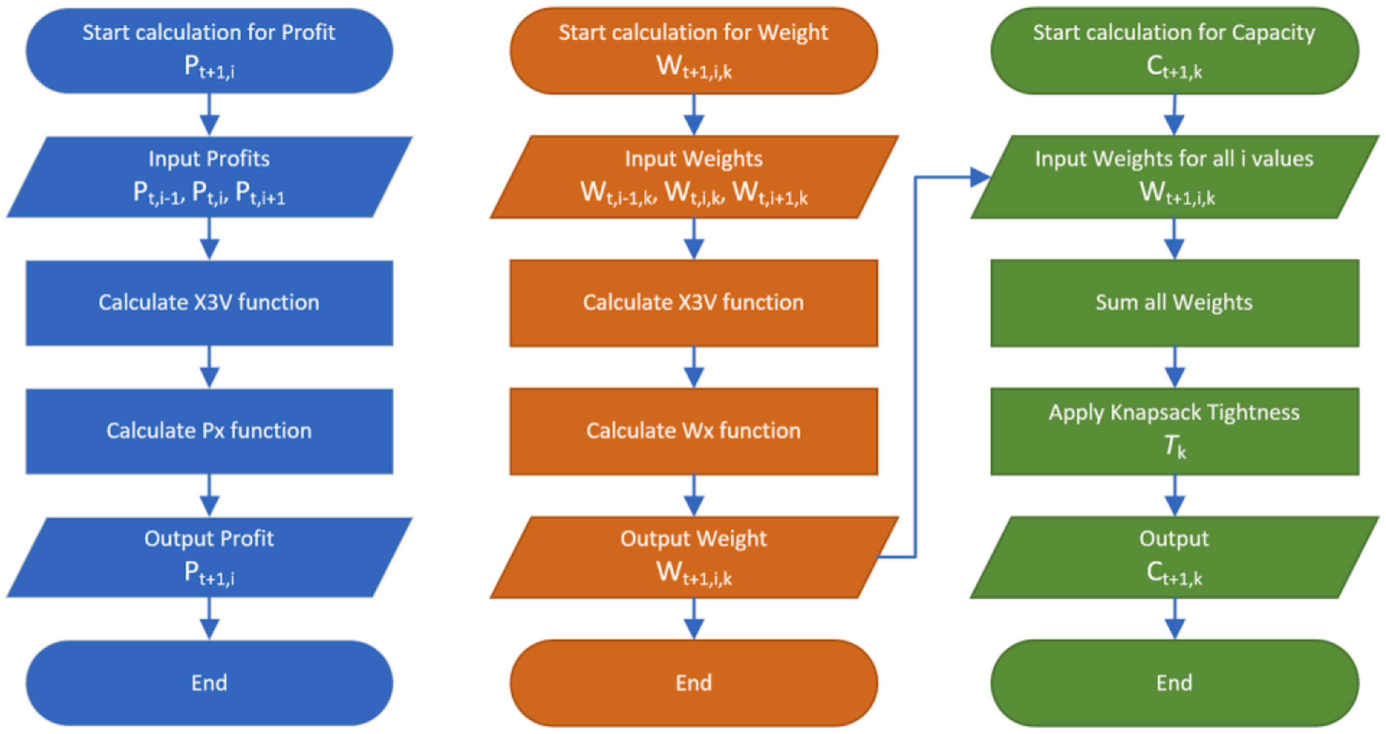
Start calculation for Profit $P_{t+1,i}$
Input Profits $P_{t,i-1}, P_{t,i}, P_{t,i+1}$
Calculate X3V function
Calculate Px function
Output Profit $P_{t+1,i}$
End
Start calculation for Weight $W_{t+1,i,k}$
Input Weights $W_{t,i-1,k}, W_{t,i,k}, W_{t,i+1,k}$
Calculate X3V function
Calculate Wx function
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

Fig. 1. Dataset state creation flowchart of Item Profits, Item Weights, and Knapsack Capacities. The flowchart shows the key dependencies of each value adjustment in the generated state.

maximum allowed weight increase to the item. Similarly, $\overline{W_{t,i}} - \frac{P_{t,i}}{minV}$ 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.

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

where, $C_{t+1,k}$ is 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 Fig. 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 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.

1.4. 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 [38]. 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 Δ SAM that set 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 Δ SAM is the maximum adjustment that will occur, it is recommended that minimum item weight $minW$ and minimum item profit $minP$ multiplied by Δ 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.

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

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 $\Delta = 0.2$
- 100 generated states with $\Delta = 0.1$
- 100 generated states with $\Delta = 0.05$

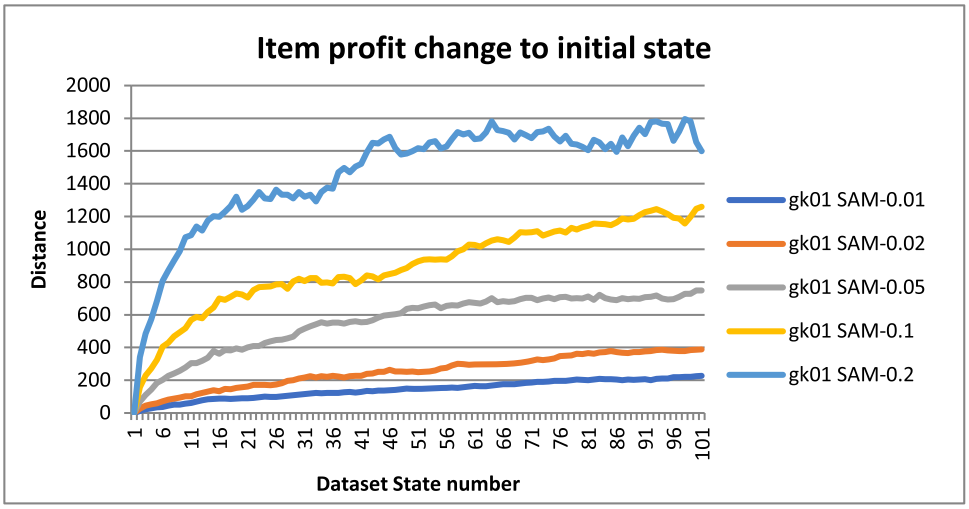

Fig. 2. Item profit change to initial state for datasets generated from GK01.

- 100 generated states with $\Delta = 0.02$
- 100 generated states with $\Delta = 0.01$

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 generated from 270 static instances in the OR library.

2. Dynamic MKP dataset analysis

Generated dynamic MKP datasets are analyzed 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 analyzed 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 solution distance. The solution distance is calculated by counting how many different items are between two state optimal result vectors.

Solution distance:

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

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:

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

Average weight distance:

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

Absolute weight distance:

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

2.1. *Statistical analysis metrics*

Each dataset has its unique properties and constraints. Therefore, to

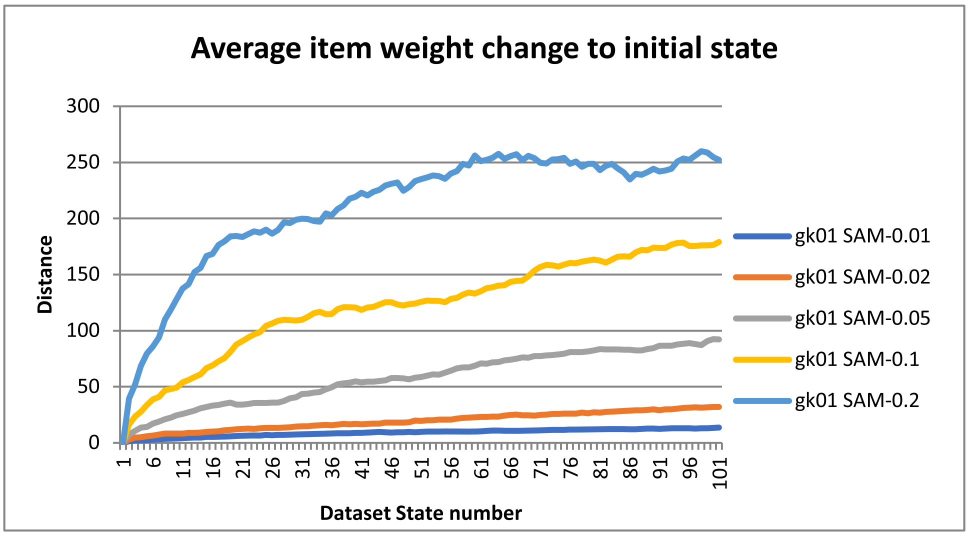

Fig. 3. Average item weight change to initial state for datasets generated from GK01.

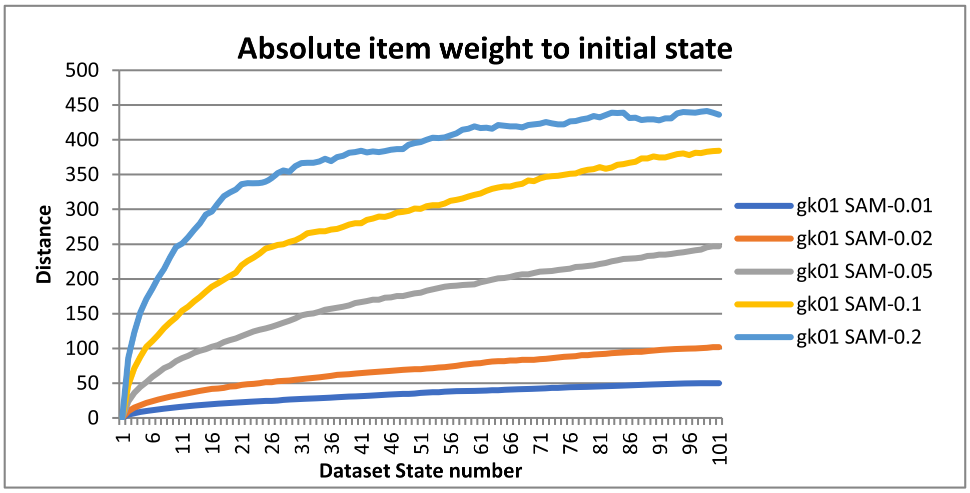

Fig. 4. Absolute item weight to initial state for datasets generated from GK01.

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.

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

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.

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

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.

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

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.

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

2.2. Example GK01 dynamic dataset statistical analysis

The graph in Fig. 2 shows how much each state's all items' profits is 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.

The graph in Fig. 3 shows how much all items' average weight is different 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 possible theoretical average weight distance. To reach 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.

The graph in Fig. 4 shows how much all items' absolute weight of all

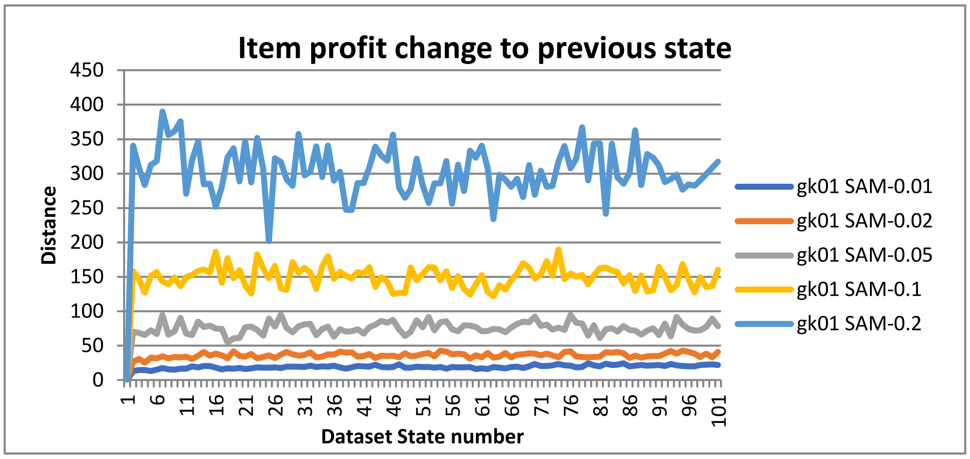

Fig. 5. Item profit change to the previous state for datasets generated from GK01.

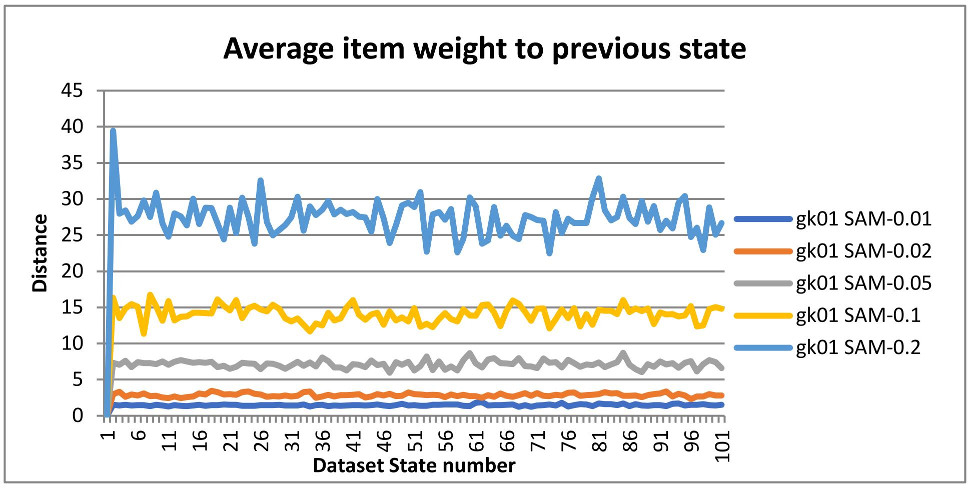

Fig. 6. Average item weight to the previous state for datasets generated from GK01.

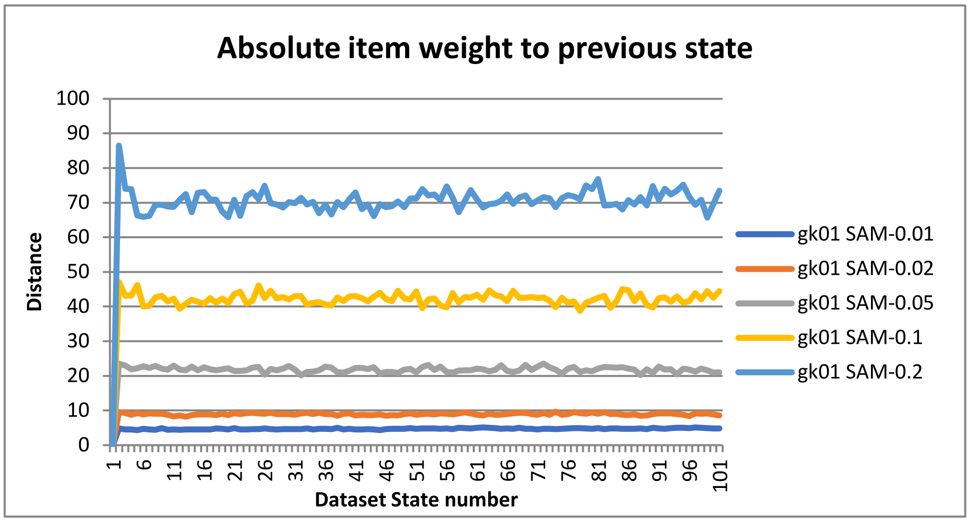

Fig. 7. Absolute item weight to the previous state for datasets generated from GK01.

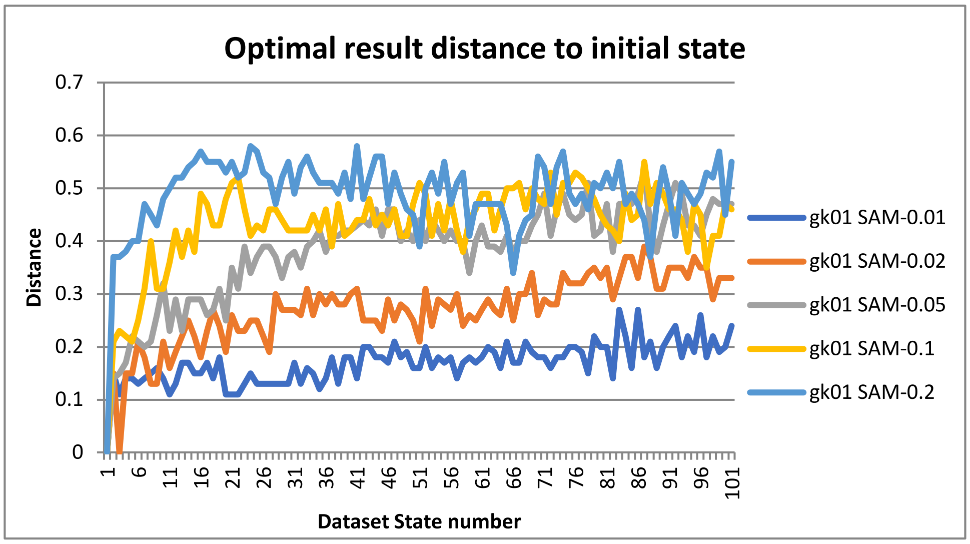

Fig. 8. Optimal result distance to initial state for datasets generated from GK01.

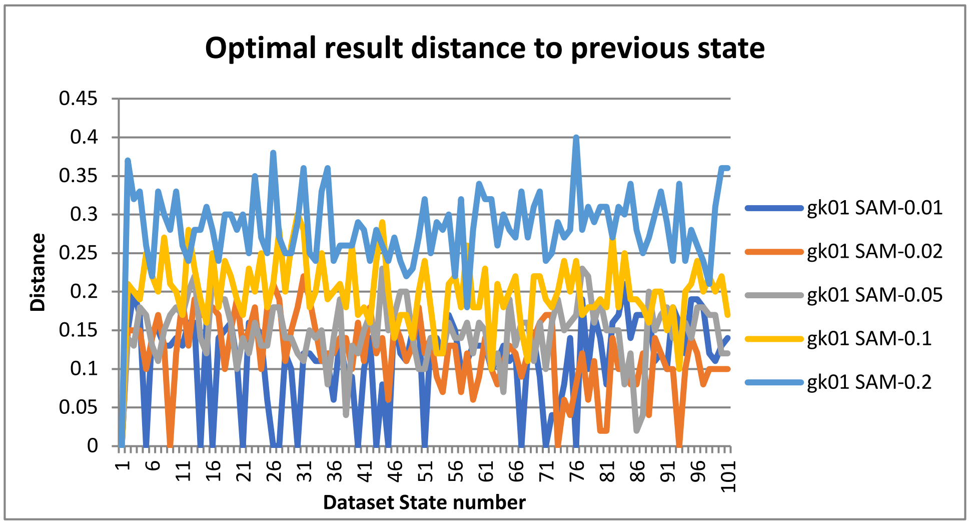

Fig. 9. Optimal result distance to the previous state for datasets generated from GK01.

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 possible theoretical absolute weight distance. These values follow the same set of constraints and are expressed using different calculations method.

The graph in Fig. 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 solved previous states to solving the next state.

The graph in Fig. 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.

The graph in Fig. 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 theoretical absolute weight distance.

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 analyze 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 [39].

3.1. Example GK01 dynamic dataset result analysis

The graph in Fig. 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. 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.

The graph in Fig. 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.

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 1, optimal result scores are shown of 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 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, 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
OR10 × 100-0.25_1 SAM-0.01	2836872	31205045	73830256	145018245	216448557	288212783
OR10 × 100-0.25_1 SAM-0.02	2836872	31176234	73779210	145071138	216174187	286840590
OR10 × 100-0.25_1 SAM-0.05	2836872	31123728	73635411	144442790	214350962	284811984
OR10 × 100-0.25_1 SAM-0.1	2836872	30805648	71709699	140660402	208541345	276564063
OR10 × 100-0.25_1 SAM-0.2	2836872	29648279	69427770	133693742	197679341	262428557
OR10 × 100-0.50_1 SAM-0.01	5091585	55953867	132336914	259552759	386384459	513376968
OR10 × 100-0.50_1 SAM-0.02	5091585	55866397	132033525	260305360	388821856	517324762
OR10 × 100-0.50_1 SAM-0.05	5091585	56065926	133118084	262589997	393619307	524410687
OR10 × 100-0.50_1 SAM-0.1	5091585	56629927	134571121	266159634	395126427	526352982
OR10 × 100-0.50_1 SAM-0.2	5091585	56542194	135328561	268860774	403977829	540243294
OR10 × 100-0.75_1 SAM-0.01	7057125	77643072	183396152	359231095	535672559	712133307
OR10 × 100-0.75_1 SAM-0.02	7057125	77369672	182892248	357984295	532064332	706249480
OR10 × 100-0.75_1 SAM-0.05	7057125	77602511	183329109	359336913	534961784	709195281
OR10 × 100-0.75_1 SAM-0.1	7057125	76740410	181074663	349629918	514015441	679144481
OR10 × 100-0.75_1 SAM-0.2	7057125	74281645	171926494	331859849	492799464	658580578
OR30 × 100-0.25_1 SAM-0.01	2699358	29687005	70191529	137774216	205487725	273471842
OR30 × 100-0.25_1 SAM-0.02	2699358	29702444	70268015	137971147	205397065	272700290
OR30 × 100-0.25_1 SAM-0.05	2699358	29502620	69475293	135491089	201383026	266403778
OR30 × 100-0.25_1 SAM-0.1	2699358	28900744	68183405	133807293	197430527	262757855
OR30 × 100-0.25_1 SAM-0.2	2699358	27977949	66054651	130252336	195671607	261708983
OR30 × 100-0.50_1 SAM-0.01	5014341	55059043	129891484	254621859	379817047	505328790
OR30 × 100-0.50_1 SAM-0.02	5014341	55025745	130037468	255079511	379914843	505524676
OR30 × 100-0.50_1 SAM-0.05	5014341	55200665	130045508	254327181	378085224	501037881
OR30 × 100-0.50_1 SAM-0.1	5014341	54810897	128082790	247397275	366761689	486607011
OR30 × 100-0.50_1 SAM-0.2	5014341	53103991	124497893	244395075	363760632	483039836
OR30 × 100-0.75_1 SAM-0.01	7071762	77864424	184162853	361344304	538697991	716206864
OR30 × 100-0.75_1 SAM-0.02	7071762	77780937	183865316	359936517	536422253	713362025
OR30 × 100-0.75_1 SAM-0.05	7071762	77932653	184544801	362035765	538160553	714949227
OR30 × 100-0.75_1 SAM-0.1	7071762	77557884	183180055	361505598	542978957	725238465
OR30 × 100-0.75_1 SAM-0.2	7071762	77189609	183136611	365412793	547980898	727926255
OR5 × 100-0.25_1 SAM-0.01	2998863	32939171	77757416	152752166	227713579	302850314
OR5 × 100-0.25_1 SAM-0.02	2998863	32821440	77541184	152413231	227296574	302426478
OR5 × 100-0.25_1 SAM-0.05	2998863	33098483	78857803	155459878	233754591	311477600
OR5 × 100-0.25_1 SAM-0.1	2998863	32819219	79103918	159026997	238008299	318890576
OR5 × 100-0.25_1 SAM-0.2	2998863	31792534	75667725	150669330	224083685	298265765
OR5 × 100-0.50_1 SAM-0.01	5259111	57816931	136753398	268534896	400626535	532774988
OR5 × 100-0.50_1 SAM-0.02	5259111	57739290	136877350	268314426	400103228	532502998
OR5 × 100-0.50_1 SAM-0.05	5259111	57570616	136465452	269864698	403784094	536018915
OR5 × 100-0.50_1 SAM-0.1	5259111	58845857	142707776	289653226	440496588	591692649
OR5 × 100-0.50_1 SAM-0.2	5259111	62153210	151213344	304160231	460660467	610787683
OR5 × 100-0.75_1 SAM-0.01	7358106	81027232	191732938	376103226	559814098	742782860
OR5 × 100-0.75_1 SAM-0.02	7358106	81164628	191864001	375883541	559920437	744012008
OR5 × 100-0.75_1 SAM-0.05	7358106	81539986	193322737	378920251	562125666	744462914
OR5 × 100-0.75_1 SAM-0.1	7358106	81354829	191189919	372271347	550102686	726936473
OR5 × 100-0.75_1 SAM-0.2	7358106	80952636	187322886	367257155	546541749	724836393

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 for the MKP problem has been adapted to solve this Dynamic MKP benchmark [26]. The algorithm has been configured to perform two dynamic optimization strategies: Full-restart strategy and Pheromone-sharing strategy. 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 [40]. All tests have been executed on the AMD Threadripper 2990WX system with the clock running at 2.9Ghz, with execution parallelism is 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 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 [41]. Using the result gap allows comparing algorithm performance quantitatively across all benchmark instances.

Following in Figs. 10 and 11, the dynamic optimization results of the ACO algorithm are displayed. 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 Fig. 10, the ACO algorithm with 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

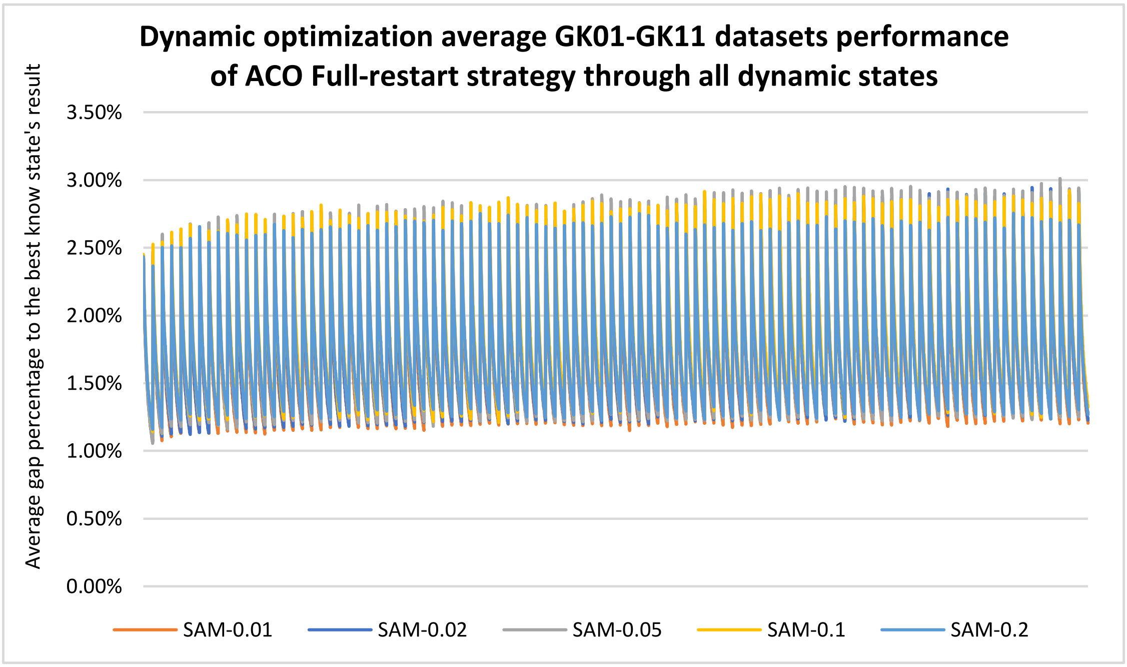

Fig. 10. Dynamic optimization performance of ACO Full-restart strategy for all SAM levels. Each line shows the average gap convergence of GK01-GK11 dynamic datasets group run 10 times each, totalling 110 runs.

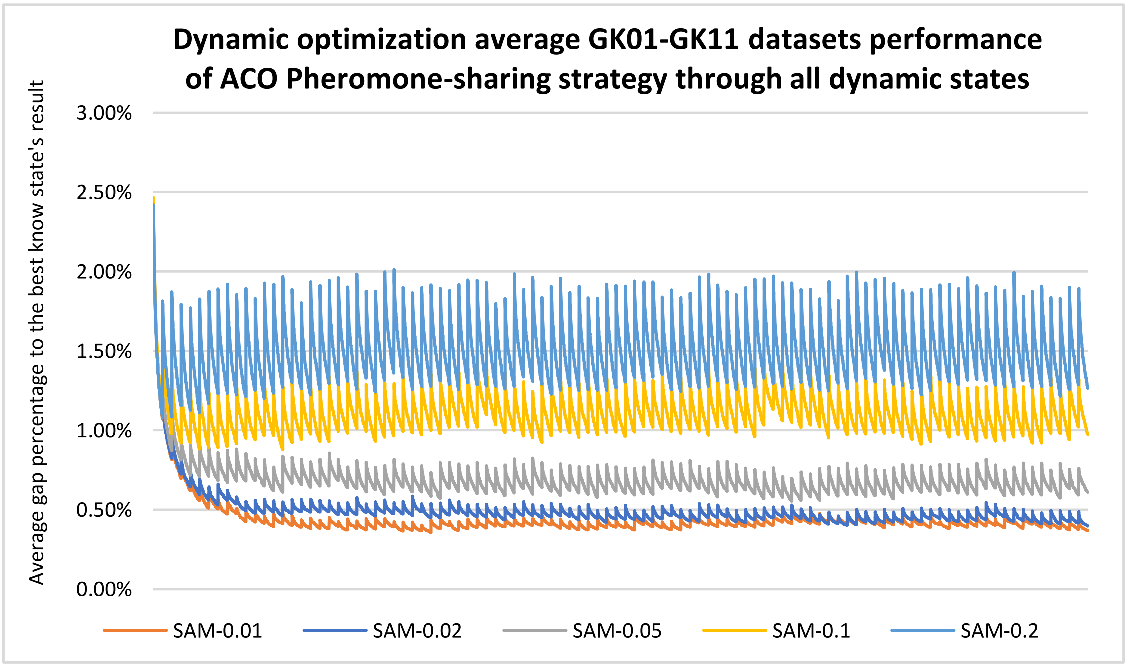

Fig. 11. Dynamic optimization performance of ACO Pheromone-sharing strategy for all SAM levels. Each line shows the average gap convergence of GK01-GK11 dynamic datasets group run 10 times each, totalling 110 runs.

quality for ACO with Full-restart strategy. In Fig. 11 ACO algorithm with 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.

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 [35].

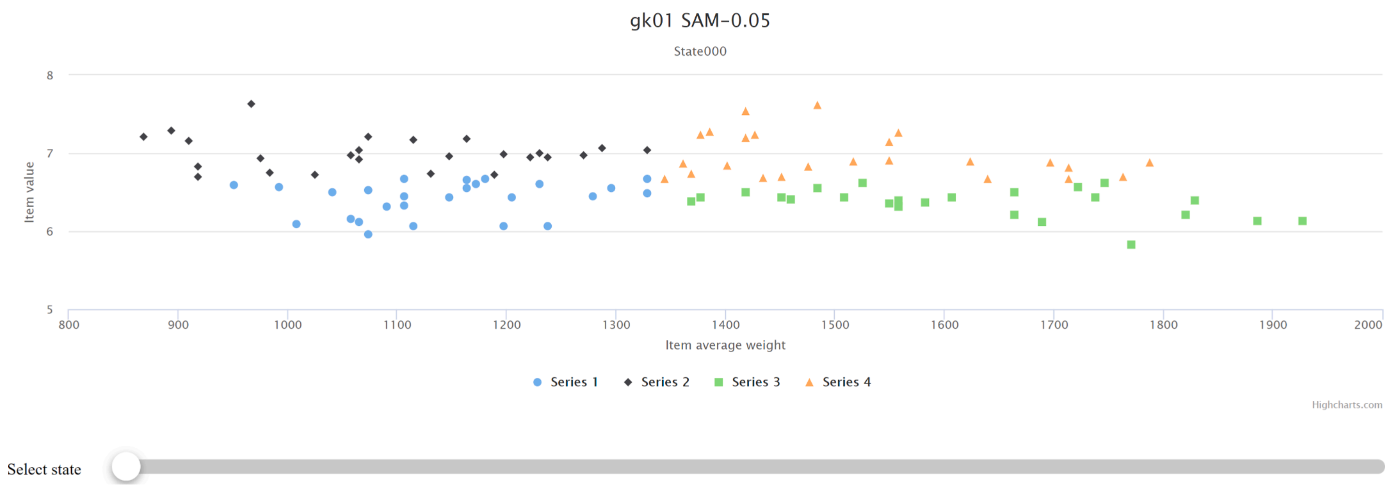

Fig. 12. Profit and weight distance effect for GK01 SAM-0.05 dataset initial state. Each series represent the division of each item's value and average weight into a quadrant based on the initial state. On the initial state, the division is clearly visible.

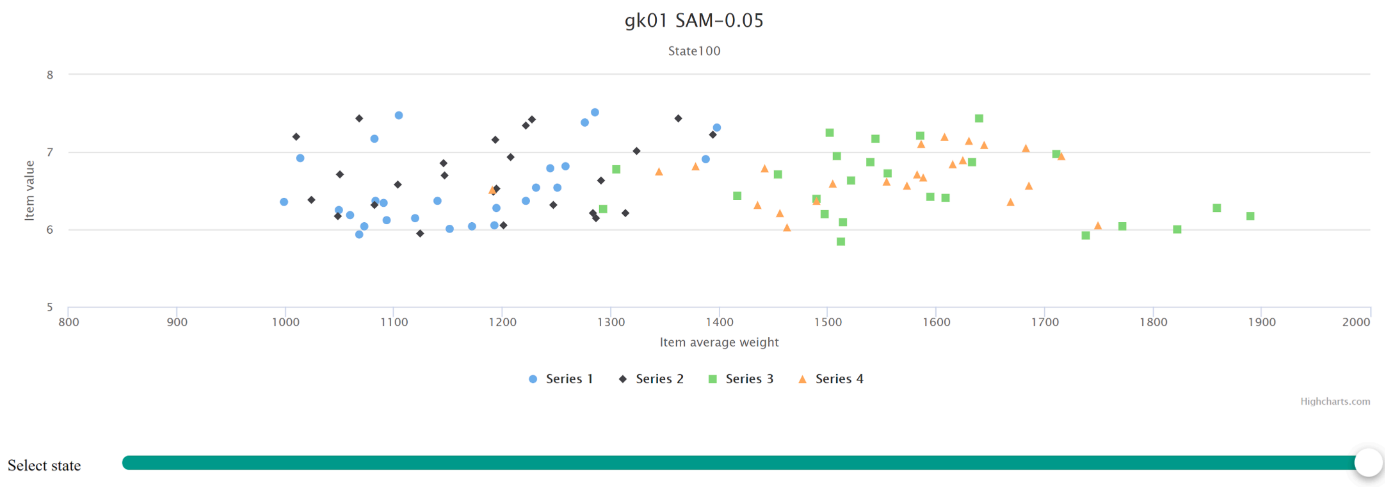

Fig. 13. Profit and weight distance effect for GK01 SAM-0.05 dataset last state. Each series represent the division of each item's value and average weight into a quadrant based on the initial state. On the last state, items are significantly mixed up.

5.1. *Profit and weight distance effect*

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 size and their profit value. Groups are chosen considering two factors. First, whether

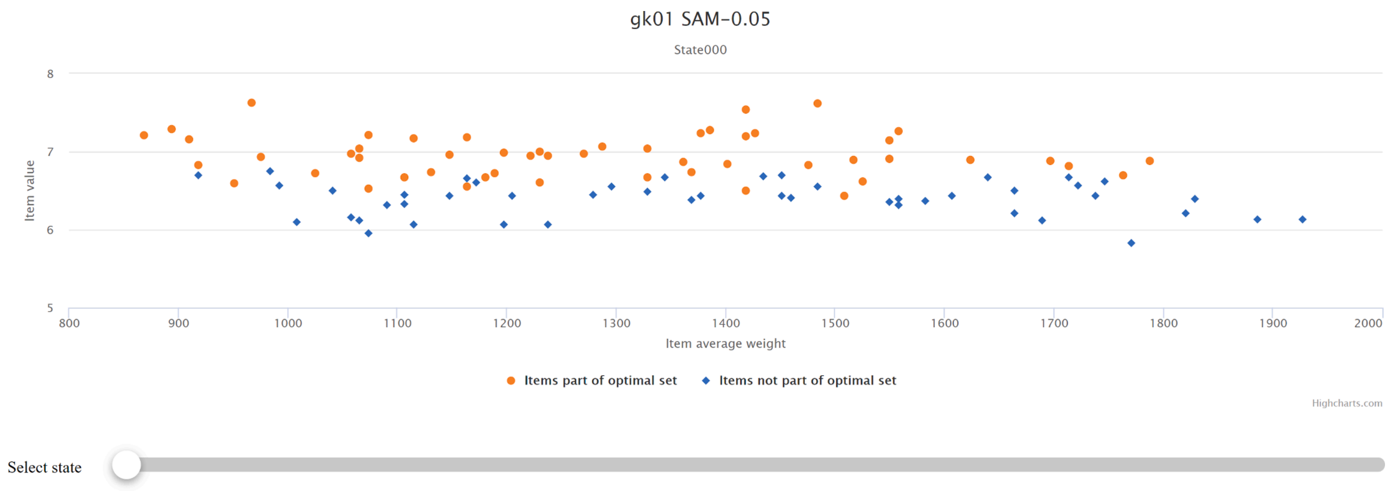

Fig. 14. Optimal result effect GK01 SAM-0.05 dataset initial state.

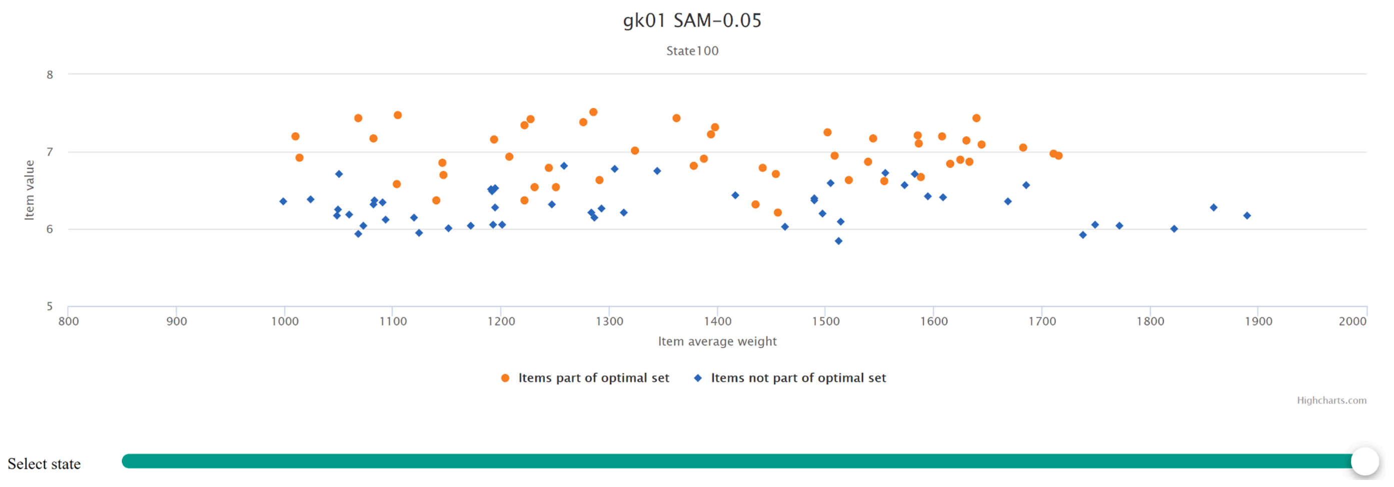

Fig. 15. Optimal result effect, GK01 SAM-0.05 dataset last state.

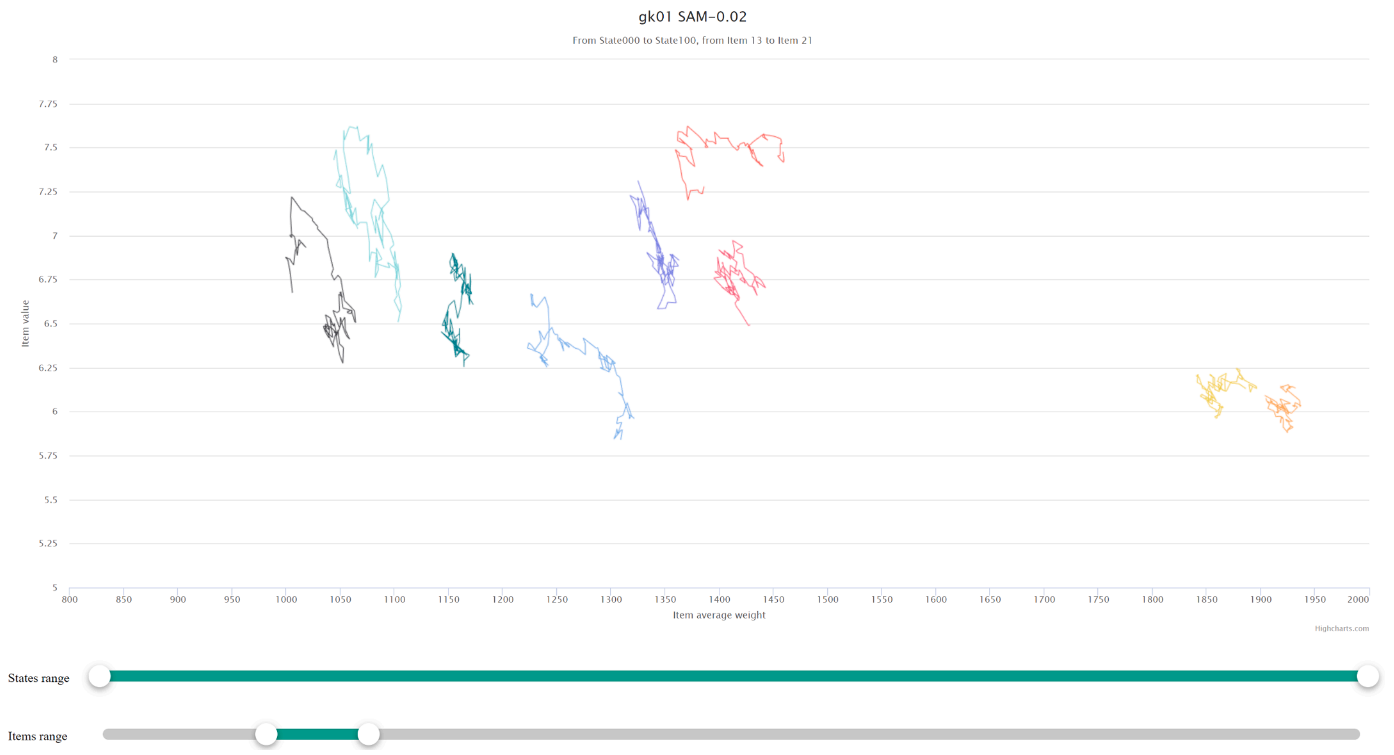

Fig. 16. Dynamic dataset constraint coverage effect, GK01 SAM-0.02 dataset, items range 13-21 inclusive.

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 Figs. 12and 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.

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 of 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 Figs. 14 and 15. In both

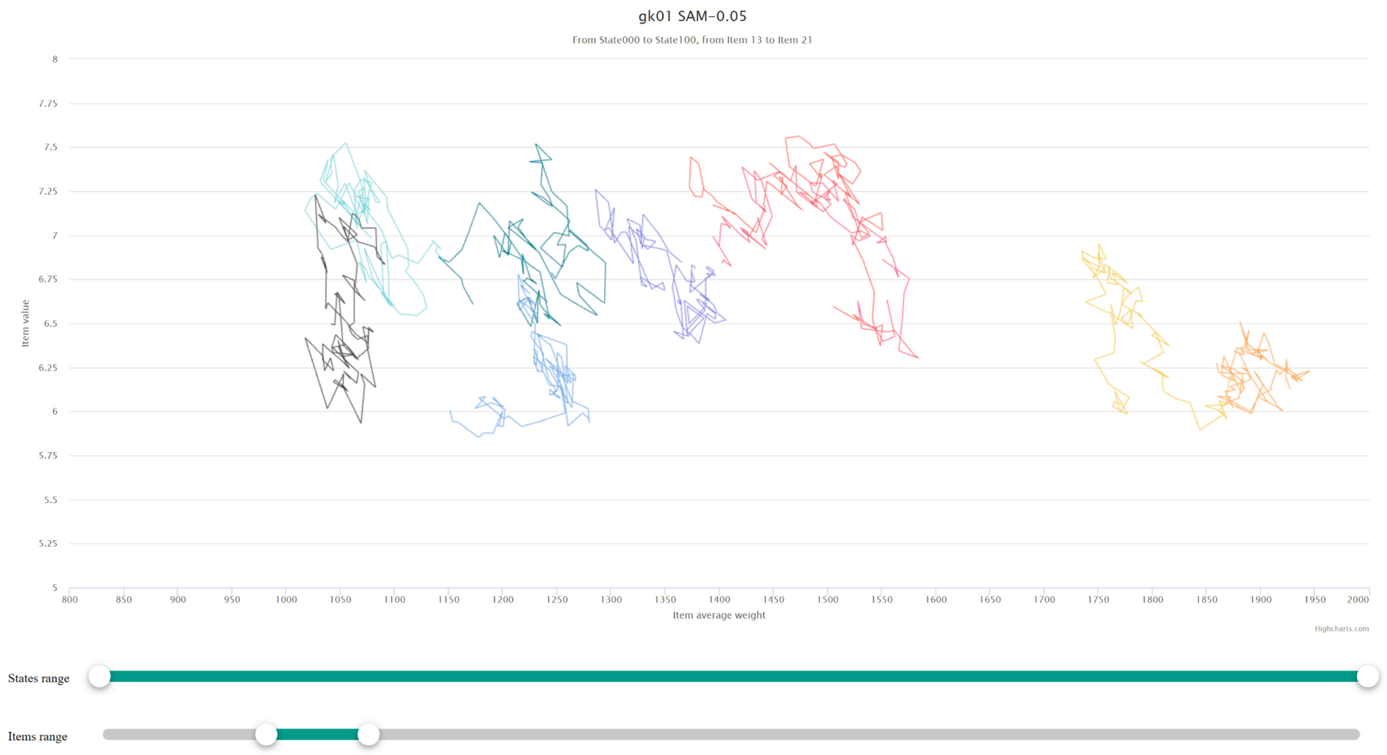

Fig. 17. Dynamic dataset constraint coverage effect, GK01 SAM-0.05 dataset, items range 13-21 inclusive.

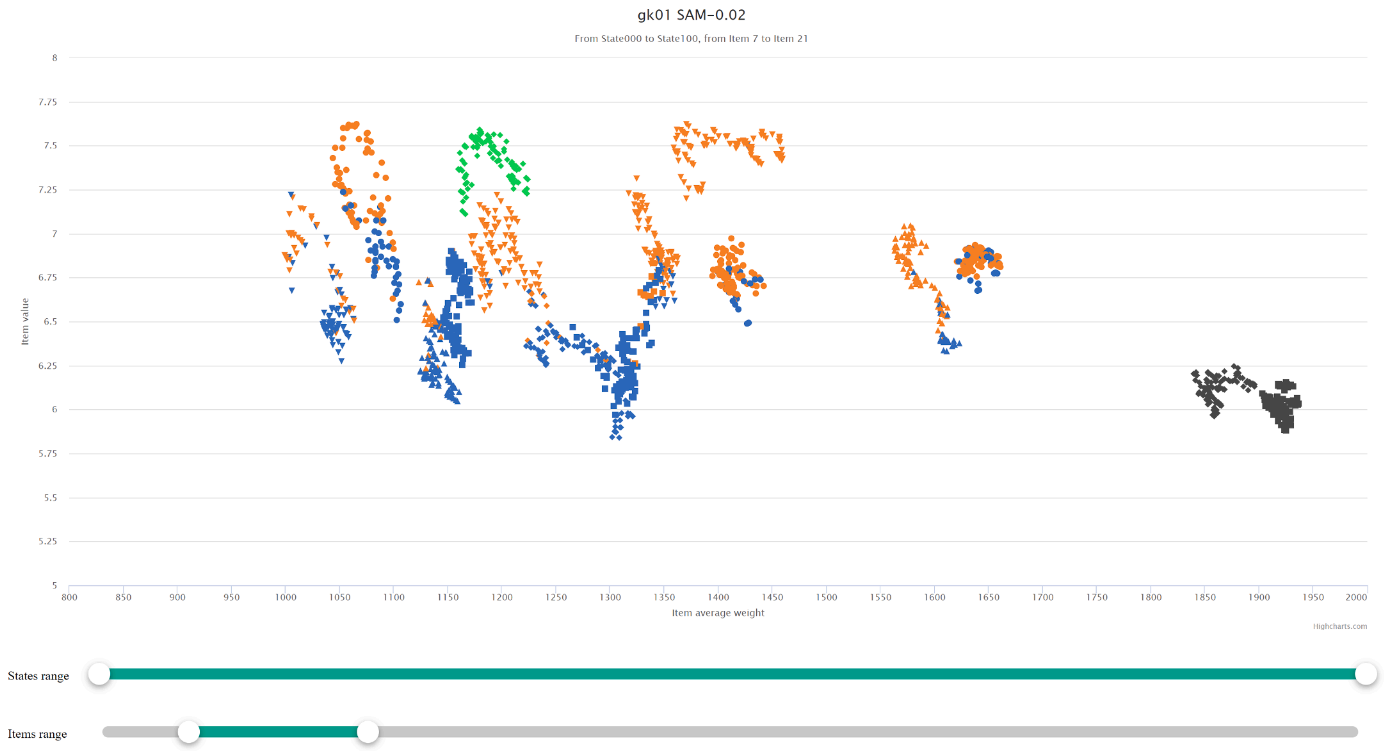

Fig. 18. Dynamic dataset optimal result coverage effect, GK01 SAM-0.02 dataset, items range 7-21 inclusive.

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 to be included in the optimal result decision vector.

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

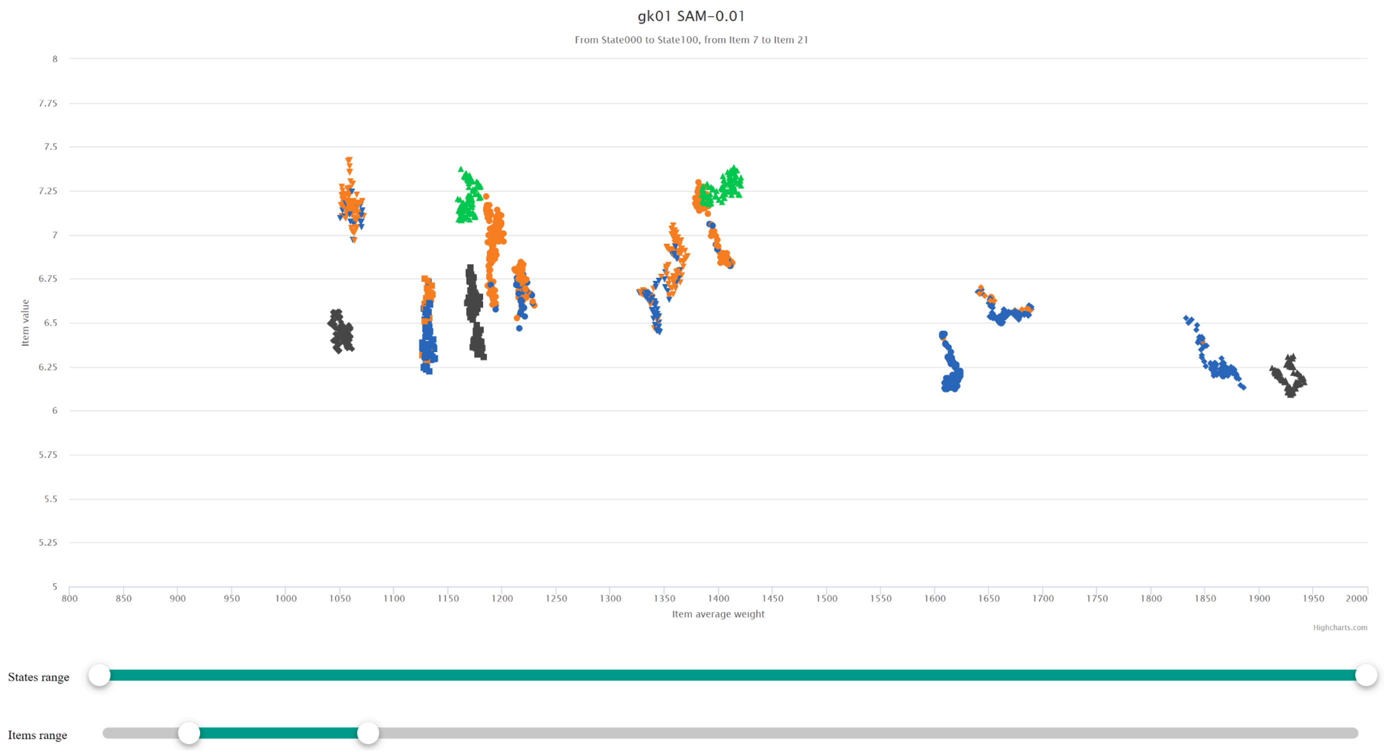

Fig. 19. Dynamic dataset optimal result coverage effect, GK01 SAM-0.01 dataset, items range 7-21 inclusive.

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 Figs. 16 and 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.

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 color represents an item in the optimal set and blue color represents a not optimal item. The chart also differentiates the items always part of the optimal set with green color series and items that are never part of the optimal set with black color 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 Figs. 18 and 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 larger constraint area and therefore larger changes in size and value have an effect on the optimal solution.

6. Conclusion

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

To solve that problem, this research proposes 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 have been generated from the existing static MKP benchmark dataset library. Then those dynamic datasets have been published to be used as Dynamic MKP benchmark.

This work also provides Dynamic MKP benchmark datasets analysis. The quantitative analysis shows the range of dynamism of all dataset parameters. Optimal result dynamics are analyzed of 455 datasets with low combinatorial complexity of 100 items, where all states have been solved to optimal result using linear solver. Then developed an interactive tool for an additional dynamic demonstration which helps to do more analysis and develop an intuitive understanding of the dynamics of the datasets. -

Conflicts of interest

We have no conflicts of interest to disclose.

References

[1] C. Arango, P. Cortés, L. Onieva, A. Escudero, Simulation-Optimization Models for the Dynamic Berth Allocation Problem, Computer-Aided Civil and Infrastructure Engineering 28 (10) (2013) 769–779.
[2] Z. Yao, Y. Jiang, B. Zhao, X. Luo, B. Peng, A dynamic optimization method for adaptive signal control in a connected vehicle environment, Journal of Intelligent Transportation Systems 24 (2) (2020) 184–200.
[3] Y. Zhou, X. Liu, Control Parameterization-Based Adaptive Particle Swarm Approach for Solving Chemical Dynamic Optimization Problems, Chemical Engineering & Technology 37 (4) (2014) 692–702.
[4] A.A. Ateya, A. Muthanna, A. Vybornova, A.D. Algarni, A. Abuarqoub, Y. Koucheryavy, A. Koucheryavy, Chaotic salp swarm algorithm for SDN multi-controller networks, Engineering Science and Technology, an International Journal 22 (4) (2019) 1001–1012.
[5] G.-G. Wang, C.-L. Wei, Y. Wang, W. Pedrycz, Improving distributed anti-flocking algorithm for dynamic coverage of mobile wireless networks with obstacle avoidance, Knowledge-based systems 225 (2021), 107133.

[6] J. Eaton, S. Yang, M. Gongora, Ant Colony Optimization for Simulated Dynamic Multi-Objective Railway Junction Rescheduling, IEEE Transactions on Intelligent Transportation Systems 18 (11) (2017) 2980–2992.
[7] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable multi-objective optimization test problems, in: Proceedings of the 2002 Congress on Evolutionary Computation 1, 2002, pp. 825–830.
[8] A. Ahrari, S. Elsayed, R. Sarker, D. Essam, A New Prediction Approach for Dynamic Multiobjective Optimization, IEEE Congress on Evolutionary Computation (2019) 2268–2275.
[9] A. Sharifi, J. Kazemi Kordestani, M. Mahdaviani, M.R. Meybodi, A novel hybrid adaptive collaborative approach based on particle swarm optimization and local search for dynamic optimization problems, Applied Soft Computing 32 (2015) 432–448.
[10] T. Friedrich, T. Kroeger, F. Neumann, Weighted preferences in evolutionary multi-objective optimization, Machine Learning and Cybernetics 4 (2) (04/2013) 139–148.
[11] M.E. Breaban, A. Iftene, Dynamic Objective Sampling in Many-objective Optimization, Procedia Computer Science 60 (2015) 178–187.
[12] H. Zhang, G.-G. Wang, Improved NSGA-III using transfer learning and centroid distance for dynamic multi-objective optimization, Complex & intelligent systems (2021).
[13] S. Rostami, A. Shenfield, A multi-tier adaptive grid algorithm for the evolutionary multi-objective optimisation of complex problems, Soft Computing 21 (17) (09/2017) 4963–4979.
[14] G. Li, G.-G. Wang, J. Dong, W.-C. Yeh, K. Li, DLEA: A dynamic learning evolution algorithm for many-objective optimization, Information sciences 574 (2021) 567–589.
[15] H. Abbass, R. Sarker, C. Newton, PDE: a Pareto-frontier differential evolution approach for multi-objective optimization problems, Evolutionary Computation, 2001. Proceedings of the 2001 Congress 2 (2001) 971–978.
[16] F.B. Ozsoydan, A. Baykasoğlu, Quantum firefly swarms for multimodal dynamic optimization problems, Expert Systems With Applications 115 (01/2019) 189–199.
[17] R. Liu, X. Song, L. Fang, More..., "An r-dominance-based preference multi-objective optimization for many-objective optimization, Soft Computing 21 (17) (09/2017) 5003–5024.
[18] D. Yazdani, M.N. Omidvar, J. Branke, T.T. Nguyen, X. Yao, Scaling Up Dynamic Optimization Problems: A Divide-and-Conquer Approach, IEEE Transactions on Evolutionary Computation 24 (1) (2020) 1–15.
[19] H. Fu, B. Sendhoff, K. Tang, X. Yao, Robust Optimization Over Time: Problem Difficulties and Benchmark Problems, IEEE Transactions on Evolutionary Computation 19 (5) (2015) 731–745.
[20] A.Ş. Uyar, Experimental Comparison of Replacement Strategies in Steady State Genetic Algorithms for the Dynamic MKP, Applications of Evolutionary Computing 4448 (2007) 647–656.
[21] Ł. Strąk, R. Skinderowicz, U. Boryczka, Adjustability of a discrete particle swarm optimization for the dynamic TSP, Soft computing (Berlin, Germany) 22 (22) (2018) 7633–7648.
[22] Ł. Strąk, R. Skinderowicz, U. Boryczka, A. Nowakowski, A Self-Adaptive Discrete PSO Algorithm with Heterogeneous Parameter Values for Dynamic TSP, Entropy (Basel, Switzerland) 21 (8) (2019).
[23] N. Ouertani, H.B. Ramdhan, S. Krichen, I. Nouaouri, H. Allaoui, A New Evolutionary Method to Deal with the Dynamic Vehicle Routing Problem, in: 2018 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD), 2018, pp. 1–5.
[24] C. Groba, A. Sartal, 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 56 (2015) 22–32.
[25] B.H. Nguyen, B. Xue, P. Andreae, M. Zhang, A New Binary Particle Swarm Optimization Approach: Momentum and Dynamic Balance Between Exploration and Exploitation, IEEE Transactions on Cybernetics 51 (2) (2021) 589–603.
[26] J. Skackauskas, T. Kalganova, I. Dear, M. Janakiram, Dynamic impact for ant colony optimization algorithm, Swarm and Evolutionary Computation 69 (2022), 100993.
[27] Y. Feng, G.-G. Wang, L. Wang, Solving randomized time-varying knapsack problems by a novel global firefly algorithm, Engineering with Computers 34 (3) (2017) 621–635.
[28] Y. Feng, G.-G. Wang, S. Deb, M. Lu, X.-J. Zhao, Solving 0–1 knapsack problem by a novel binary monarch butterfly optimization, Neural Computing and Applications 28 (7) (2015) 1619–1634.
[29] Y. Feng, G.-G. Wang, X.-Z. Gao, A Novel Hybrid Cuckoo Search Algorithm with Global Harmony Search for 0-1 Knapsack Problems, International Journal of Computational Intelligence Systems 9 (6) (2016) 1174.
[30] J. Cao, B. Yin, X. Lu, Y. Kang, X. Chen, A modified artificial bee colony approach for the 0-1 knapsack problem, Applied intelligence 48 (6) (2017) 1582–1595.
[31] Y. Feng, G.-G. Wang, A binary moth search algorithm based on self-learning for multidimensional knapsack problems, Future Generation Computer Systems 126 (2022) 48–64.
[32] B. Abdollahzadeh, S. Barshandeh, H. Javadi, N. Epicoco, An enhanced binary slime mould algorithm for solving the 0–1 knapsack problem, Engineering with Computers (2021).
[33] J. Skackauskas, GitHub-Dynamic MKP Datasets Generator (2021) [Online]. Available, https://github.com/jonasska/Dynamic-MKP-Datasets-Generator [Accessed 24 2 2021].
[34] J. Skackauskas, GitHub (2021) [Online]. Available, https://github.com/jonasska/Dynamic-MKP-Benchmark-Datasets [Accessed 14 4 2021].
[35] J. Skackauskas, GitHub-Dynamic MKP Datasets Visualization (2021) [Online]. Available, https://github.com/jonasska/Dynamic-MKP-Datasets-Visualization [Accessed 1 1 2021].
[36] J. Branke, M. Orbayı, Ş. Uyar, The Role of Representations in Dynamic Knapsack Problems, Applications of Evolutionary Computing (2006) 764–775.
[37] A. Baykasoğlu, F.B. Ozsoydan, Evolutionary and population-based methods versus constructive search strategies in dynamic combinatorial optimization, Information Sciences 420 (2017) 159–183.
[38] 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].
[39] Google, GitHub-or-tools, Google (2015) [Online]. Available, https://github.com/google/or-tools [Accessed 16 12 2021].
[40] D. Angus, T. Hendtlass, Dynamic Ant Colony Optimisation, Applied Intelligence 23 (1) (07/2005) 33–38.
[41] 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].