Learning to Predict the Utility of Subset Patterns

Anna Novin and Shaul Markovitch

August 2023

1 Introduction

The optimization of Subset Selection Problems (SSPs) has long been a core pursuit within the realm of computational research, encompassing an array of applications in diverse domains across computer science.

The difficulty in optimization of subset selection problems can vary significantly based on the specific problem details and constrains. Some of these problems, such as Set Cover Problem, Vertex Cover Problem and Knapsack Problem are classified as NP-hard. This classification implies that this type of computational problem is deemed to be difficult to solve efficiently. Specifically, it's a problem for which no known algorithm exists that can provide a solution in polynomial time. This implies that as the problem size increases, the time required to find a solution can grow significantly, potentially reaching impractical or unmanageable levels, often in exponential time.

Addressing the challenge posed by NP-hardness, researchers have pursued innovative approaches to efficiently tackle subset selection problems. One widely employed and effective strategy involves the iterative generation of an initial subset followed by refinement through local search. To initiate this process, an initial subset is crafted using methods such as random selection, heuristic initialization, or greedy algorithms. Subsequently, a local search procedure is engaged, systematically exploring neighboring solutions and progressively enhancing the subset's composition to optimize the desired objective function.

In this work, we present a novel method for intelligent initialization in subset selection problems. Our method involves a meticulous curation of the initial subset, driven by a comprehensive evaluation of each element's contribution to the subset's objective. The motivation behind this method stems from the realization of the importance of constructing subsets with a meticulous examination of the individual contribution of each element. Moreover, our objective is to grasp the intricate interdependencies among these elements, allowing us to precisely measure their impact on the subset — whether it's positive, negative or negligible, taking into careful consideration their influence within the context of interactions with other elements. In utilizing these contributions, we will strategically select elements for inclusion in the subset in accordance with their impact.

We present two algorithms that encompass the proposed method. Both algorithms calculate the contribution of each element by analyzing a generated set of random subsets, and subsequently generate subsets that reflect those contributions. The initial algorithm treats each element in isolation, determining its contribution independently. The second algorithm, however, represents a more sophisticated approach. It surpasses the first by acknowledging the interdependence among elements and computes contributions in light of these relationships. Specifically, the second algorithm discerns patterns within subsets—whether an element is included or excluded — and incorporates this insight into the calculation of element contribution.

2 Problem Definition

Let $E = \{e_1 \ldots, e_n\}$ be a set of elements and let $U : 2^E \rightarrow R$ be a utility function. We define Subset Selection Problem as finding $S \subseteq E$ such that $U(S)$ is optimal.

3 Intelligent Initialization Method

We present a novel method for choosing an initial subset for subset selection problem - the first component of any subset selection algorithm. Our approach centers around selecting the initial subset elements based on their contribution to the subset.

The contribution manifest in three distinct manners: positively, where an element's inclusion augments the subset's utility; negatively, signifying a detraction from said utility; and insignificantly, denoting a negligible impact on utility. The limits of significance are defined by empirically determined threshold.

The calculation methodology for contribution of each element varies between the two proposed algorithms. However, a common thread among both algorithms is the generation of a set of random subsets, for which individual utilities are subsequently computed. From the said set, the contribution are calculated. The size of the random subset set is determined by the parameter k, representing the number of utility calculations conducted to generate the initial subset.

3.1 Contribution Driven Subset Generation: Assuming Element Independence

The algorithm requires three key components: a set of elements denoted as E, a utility function denoted by U and a specific computational limit indicated k.

The algorithm initiates by generating a set S of k random subsets and subsequently assessing their individual utilities. For every element $e \in E$, the algorithm computes its contribution, based on the generated set, using the following formula:

$$\Delta U(e) = AVG_{e(S_j)=1}(U(S_j)) - AVG_{e(S_j)=0}(U(S_j))$$

This formula determines the contribution of individual elements by computing the average utility of subsets containing the element and then subtracting the average utility of subsets without it.

It's essential to highlight that the algorithm determines element contributions without considering interactions among elements. Specifically, the algorithm assumes that the effect on subset utility caused by including or excluding an element is independent from the presence or absence of other elements.

Example of the calculated contribution:

Elements	Contributions
e_1	0.00889
e_2	-0.12377
e_3	0.00083
e_4	0.00817
e_5	-0.00037

After the contribution calculations are finished, a threshold is applied to determine the significance of each contribution. This threshold is chosen based on empirical analysis of various thresholds across different k values, specifically tailored to the test set. The threshold that results in the highest utility across a range of k values is selected.

Once the element contributions are calculated, the algorithm proceeds to estimate the likelihood of selecting each element for inclusion in the subset based on their respective contributions. To elaborate, if an element's absolute contribution surpasses the threshold and the contribution is positive, its inclusion probability is set to 1. Conversely, if the contribution is negative, the inclusion probability becomes 0. On the occasion that the absolute contribution falls below the threshold, the inclusion probability defaults to 0.5, signifying randomness.

Example of calculated probabilities with threshold 0.001:

Elements	Probabilities
e_1	1
e_2	0
e_3	1
e_4	1
e_5	0.5

In the end, an initial subset is generated using the probabilities derived from the computed contributions. These probabilities determine the association between each element's contribution and its likelihood of being included in the subset.

3.2 Pattern Driven Subset Generation: Assuming Elements Interdependence

The algorithm requires the same three key components as the first algorithm: a set of elements E, a utility function U and a computational limit k.

This algorithm represents a more sophisticated approach to calculating element contributions. While the initial algorithm assumed that the impact on the utility of including or excluding each element from the subset was independent of the presence or absence of other elements, this advanced algorithm takes into account the interdependencies between elements. This is achieved by discerning patterns within subsets and their corresponding utility outcomes. These patterns encompass the elements included in a subset (denoted as 1), the elements excluded from a subset (denoted as 0), and the elements that may be either included or excluded (indicated by *), as they hold minimal influence on the overall subset utility. We called the elements that have specific assignments in the patter as core elements. To illustrate, consider the pattern 1***10, which signifies the inclusion of the first and sixth elements, exclusion of the seventh element, and flexibility in including or excluding the remaining elements. The first, sixth and seventh elements are core elements in the pattern.

Example of learned patterns and their utilities:

Pattern	Utility
100**1*	0.82
**1*01*	0.96
1****10	0.60
1**001*	0.85
0*1*1*0	0.75

To identify subset patterns and their corresponding utilities, the algorithm initiates by generating a set of k random subsets and subsequently evaluating their individual utilities. This step forms the foundation for constructing a regression tree, which is employed in the next phase.

A regression tree is a specialized variant of decision tree utilized in machine learning to predict continuous values. It operates by segregating data into distinct subsets based on input features, with the overarching objective of minimizing the variance of the target variable within each subset. This process is executed

through recursive feature and threshold selections, leading to data partitioning. Subsequently, this partitioned data is utilized to build the regression tree. As a result, the tree structure is designed to guide the navigation of new data instances through the model, predicting target values with a focus on comprehending the correlations between the generated subsets and their consequential utilities.

Constructing a regression tree enables us to compute subset patterns and their corresponding utilities. This is achieved by tracing each path from the root node to a leaf node, where the leaf node holds the utility value of its respective path.

The regression tree's design ensures that each path encapsulates the specific elements it comprises, denoting their inclusion (marked as 1) or exclusion (marked as 0) within the subset. These marked elements hold substantial impact on the utility. Conversely, elements absent from a given path are deemed by the regression tree to exert minimal influence on the subset's utility. These elements are denoted by * in the resulting pattern, indicating their negligible significance.

In the ensuing stage, the algorithm pinpoints the optimal pattern, characterized by the highest utility. This is achieved by identifying the path leading to the leaf node with the highest utility, and then producing a pattern accordingly. This optimal pattern encapsulates the most favorable combination of elements that contribute to achieving the peak utility within the given constraints.

In the final stage, an initial subset is generated according to the optimal patter. Elements are included or excluded as determined by the pattern's assignments, while the remaining elements are randomly selected.

It is interesting to note, that similarly to the first algorithm, this approach also incorporates a threshold to ascertain the importance of each element's contribution. However, the threshold determination is less straightforward in this context. While the initial algorithm establishes the threshold during a distinct phase, involving empirical tests across a spectrum of thresholds and k values, this advanced algorithm delegates the responsibility of identifying significant elements to the regression tree. By delegating this responsibility to the tree, the algorithm effectively decides their significance, streamlining computation and enhancing efficiency. This innovative approach ultimately conserves valuable computation time.

4 Empirical Evaluation

4.1 Methodology

We have tested both algorithms on uniform random 3-SAT test-sets from the SATLIB benchmark problems library. The Uniform Random 3-SAT is a family of SAT problems distributions obtained by randomly generating 3-CNF formulae.

The following test-sets were used:

- uf20-91 - 1000 instances, clause length is 3, 20 variables, 91 clauses
- uf50-218 - 1000 instances, clause length is 3, 50 variables, 218 clauses

Both algorithms are evaluated against a benchmark, an average utility of a sample of randomly generated subsets. The sample size, 1000 subsets, is the same for the benchmark and the proposed algorithms.

In all plots, each dot represent the average utility of the results of 20 instances from one of the test sets, where each result is evaluated on 1000 generated samples.

For each test, we measured the time required to compute the contributions, providing an additional parameter for comparing the performance of the two algorithms.

4.2 Performance of Contribution Driven Subset Generation Algorithm

4.2.1 Effect of Threshold Parameter on Average Utility

The primary objective of the conducted tests was to identify the optimal threshold for a predetermined set of issues. This investigative effort focused on a selection of 20 instances from the uf20-91 test-set and the uf50-218 test-set. Throughout the course of these tests, a diverse array of threshold values was considered, each in conjunction with various k parameters. The outcomes of this systematic inquiry are illustrated in the subsequent figures, offering a concise visual representation of the achieved results.

Effect of threshold parameter tested on uf20-91 test-set:
The following figure depicts the effect of the following threshold values: [0, 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009]; across the following k values: [10, 50, 150, 300, 500] tested on 20 instances from uf20-91 test-set.

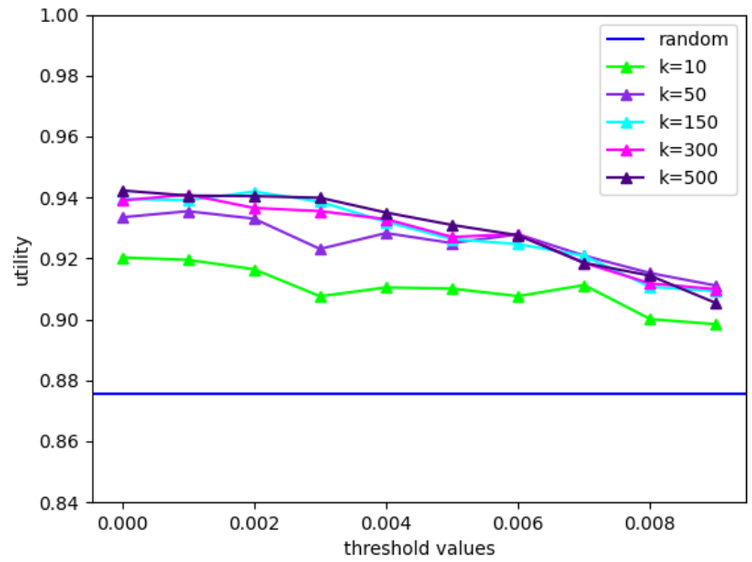

Figure 1: Effect of Threshold on Average Utility on uf20-91 test-set

From the figure above, it is evident that the highest utility for all k values is attained with a threshold below 0.001. To gain a deeper understanding of the behaviors within the range below 0.001, we conducted a specific test focused on these values. This test involved calculating the average utility for the following threshold values: [0, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 0.001], across varying k values: [10, 50, 150, 300, 500]. This analysis was performed on 20 instances from the uf20-91 test-set.

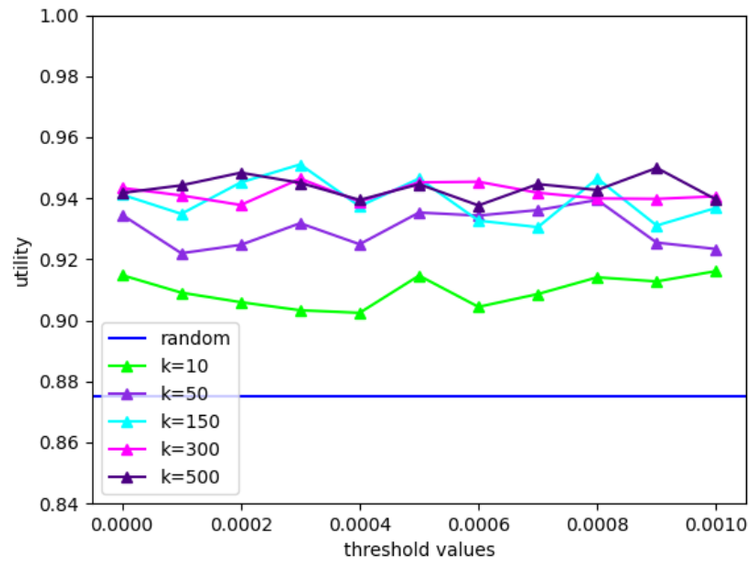

Figure 2: Effect of Threshold on Average Utility on uf20-91 test-set

As seen in the figure above, it's clear that the utility remains relatively stable for thresholds within the range below 0.001. Based on these findings, we have opted to utilize a threshold of 0.001 for all ensuing tests on the uf20-91 test set.

Effect of threshold parameter tested on uf50-218 test-set:
Similarly to the tests on uf20-91 test set, the following figure depicts the effect of the following threshold values: [0, 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009]; across the following k values: [10, 50, 150, 300, 500] tested on 20 instances from uf50-218 test-set.

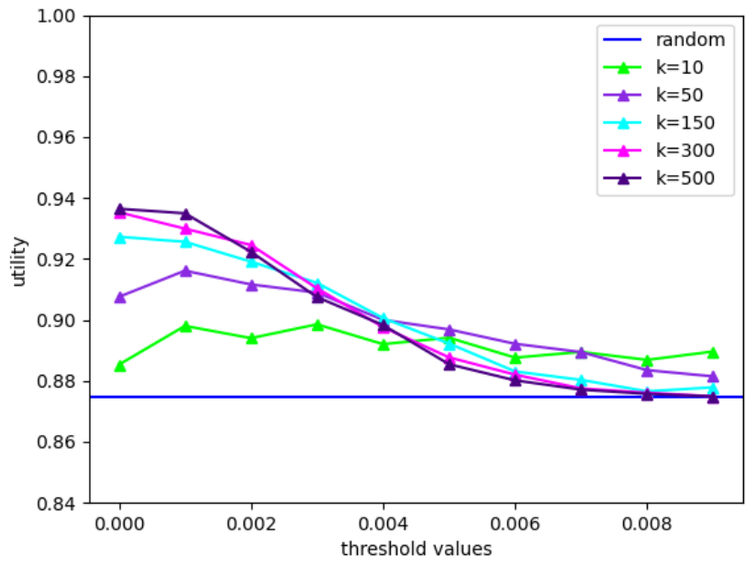

Figure 3: Effect of Threshold on Average Utility on uf50-218 test-set

From the depicted figure, it's clear that the peak utility is consistently achieved for all values of k when the threshold is set below 0.001. To delve deeper into the nuances within the sub-0.001 range, we conducted a focused investigation. This specific test aimed to shed light on this range by calculating the average utility across a set of threshold values: [0, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 0.001], encompassing the following k values: [10, 50, 150, 300, 500]. The analytical exploration comprised 20 instances drawn from the uf50-218 test-set.

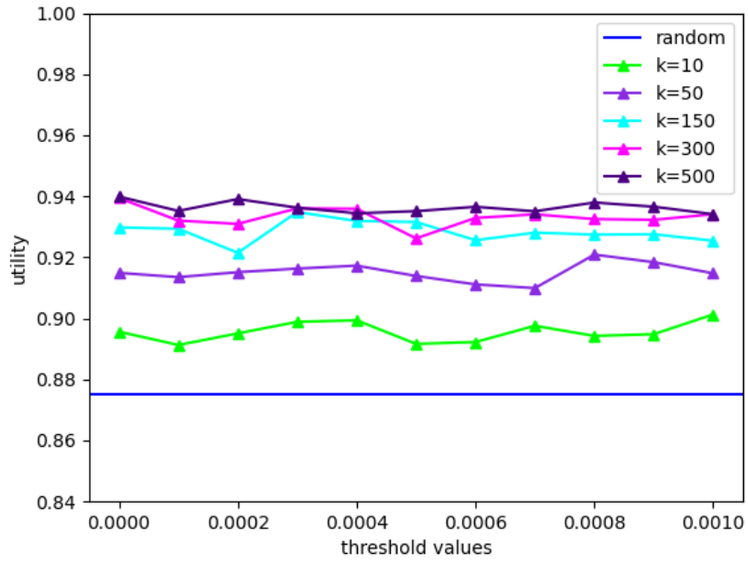

Figure 4: Effect of Threshold on Average Utility on uf50-218 test-set

The figures clearly show that, within the uf50-218 test set, the optimal threshold lies below 0.001. Detailed examination of the second figure reveals no substantial differences under the 0.001 threshold. Given this empirical insight, we've chosen to employ a 0.001 threshold for all subsequent tests on the uf50-218 test set.

Comparison of the results for uf20-91 and uf50-218 test-sets:
While the outcomes for the uf20-91 and uf50-218 test-sets slightly diverge, a common trend emerges: the peak utility across all k values is consistently attained with a threshold below 0.001. Moreover, no substantial differences are observed when the threshold is set below 0.001.

4.2.2 Effect of k Parameter on Average Utility

The following tests aim to assess the impact of the parameter k on the average utility of the produced subsets. Specifically, we will examine how varying the size of the random subset set, which is employed to establish the contribution of each variable, and its size denoted by k, influences the resulting utility of the generated subset based on those contributions. We will perform the tests on 20 instances from uf20-91 test-set and on 20 instances from uf50-218 test-set.

Effect of k parameter tested on uf20-91 test-set:
The following figure depicts the effect of the following k values: [5, 10, 20, 30, 50, 60, 70, 80, 90, 100] tested on 20 instances from uf20-91 test-set.

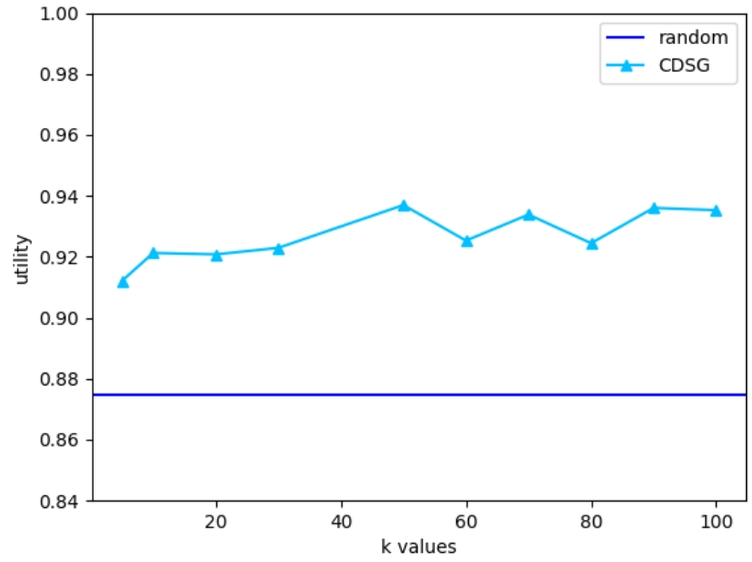

Figure 5: Effect of k parameter on Average Utility on uf20-91 test-set

The following figure depicts the effect of the following k values: [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000] tested on 20 instances from uf20-91 test-set.

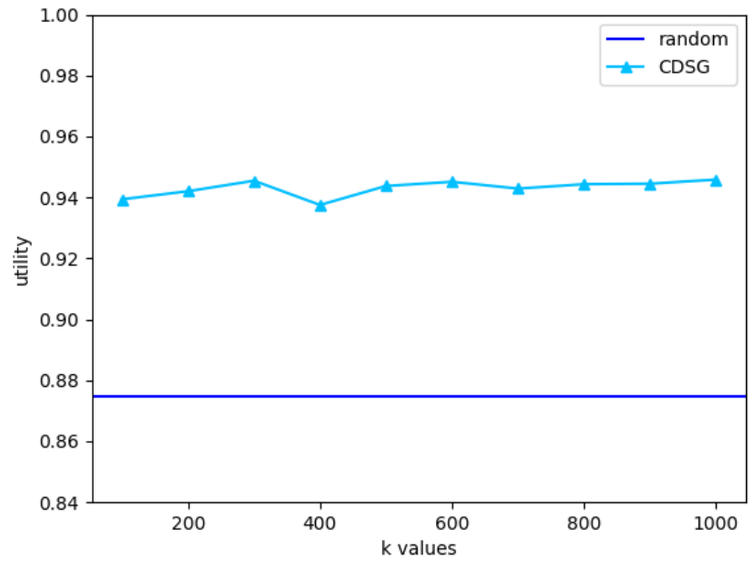

Figure 6: Effect of k parameter on Average Utility on uf20-91 test-set

The following figure depicts the effect of the following k values: [1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000] tested on 20 instances from uf20-91 test-set.

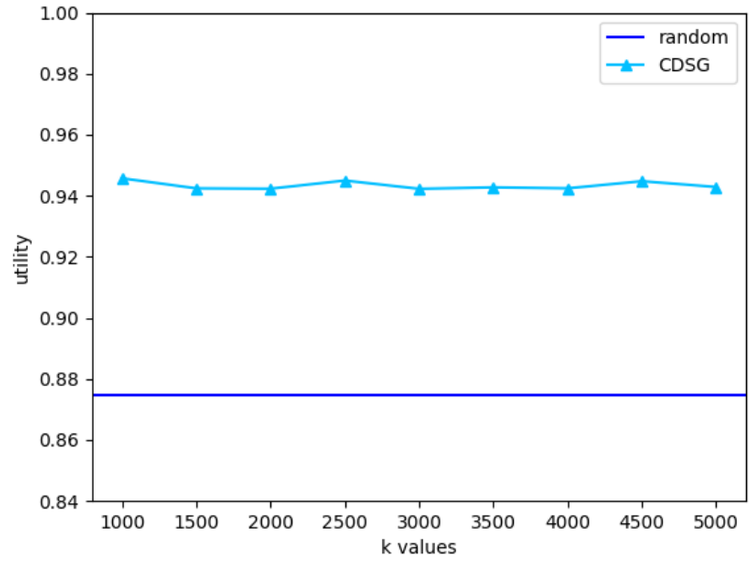

Figure 7: Effect of k parameter on Average Utility on uf20-91 test-set

The following figure depicts the effect of the following k values: [5000, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000] tested on 20 instances from uf20-91 test-set.

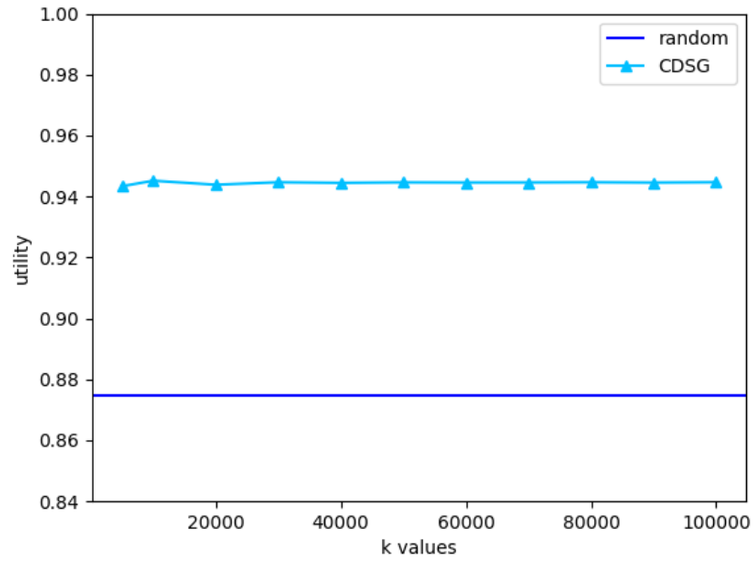

Figure 8: Effect of k parameter on Average Utility on uf20-91 test-set

Additionally we calculated the time (in seconds) that it takes to calculate to probabilities for each element, partial results are presented in the following table:

k	time (in seconds)
10	0.0028
50	0.0133
100	0.0280
500	0.1892
1000	0.2328
5000	1.4532
10000	2.8883
50000	10.0735
70000	14.2310
100000	20.0665

The above figures indicate that the highest utility at 0.94 is achieved at k value above 300. After this value, the utility does not change significantly even for k values as big as 100,000. From those funding we may conclude that the optimal k value, for the uf20-91 test set with the threshold 0.001 is 300.

Effect of k parameter tested on uf50-218 test-set:
The following figure depicts the effect of the following k values: [5, 10, 20, 30, 50, 60, 70, 80, 90, 100] tested on 20 instances from uf50-218 test-set.

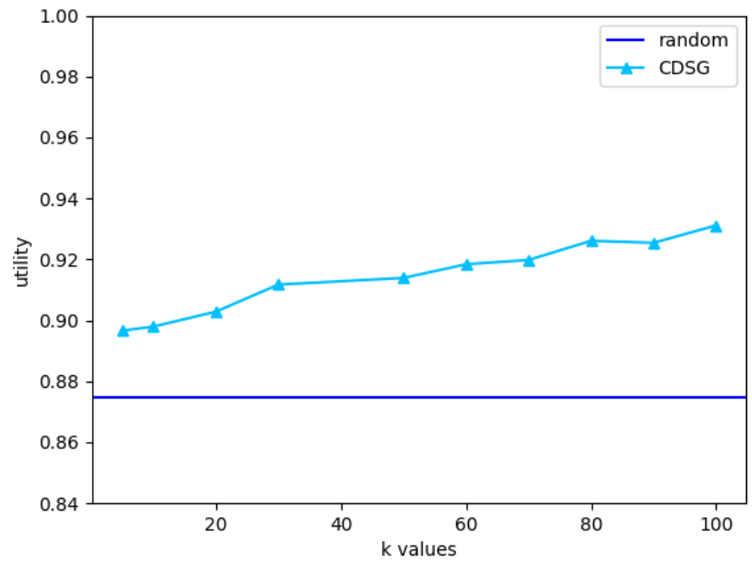

Figure 9: Effect of k parameter on Average Utility on uf50-218 test-set

The following figure depicts the effect of the following k values: [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000] tested on 20 instances from uf50-218 test-set.

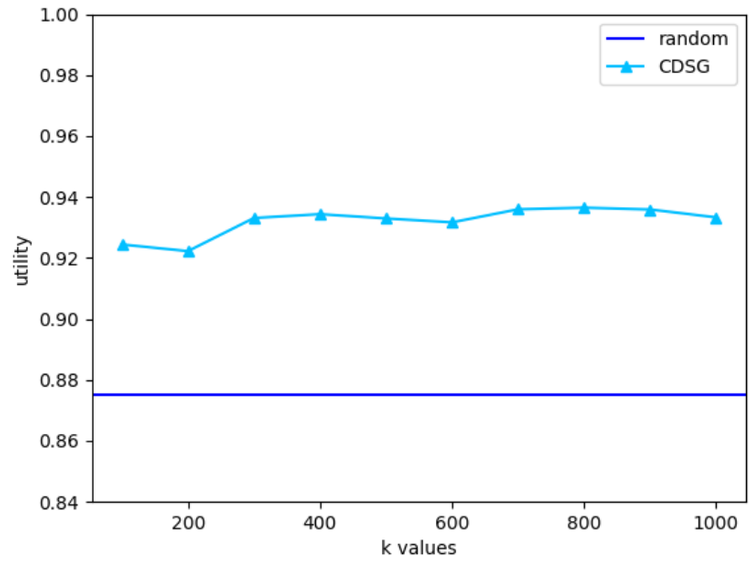

Figure 10: Effect of k parameter on Average Utility on uf50-218 test-set

The following figure depicts the effect of the following k values: [1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000] tested on 20 instances from uf50-218 test-set.

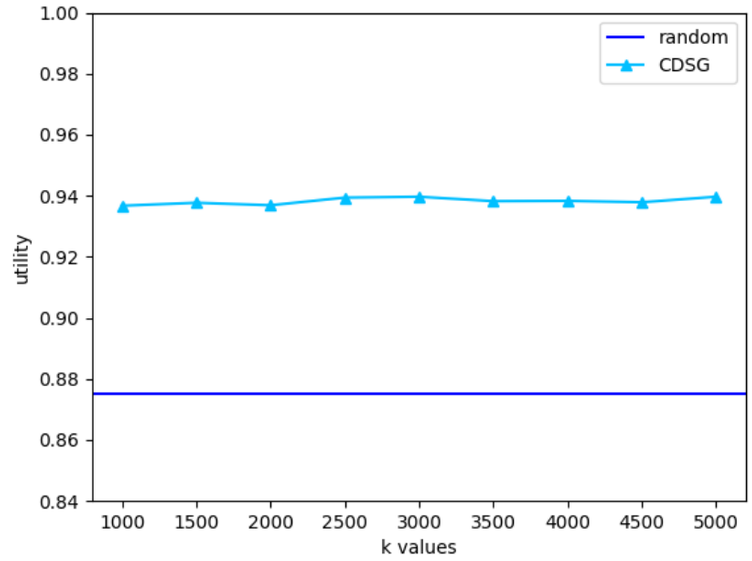

Figure 11: Effect of k parameter on Average Utility on uf50-218 test-set

The following figure depicts the effect of the following k values: [5000, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000] tested on 20 instances from uf50-218 test-set.

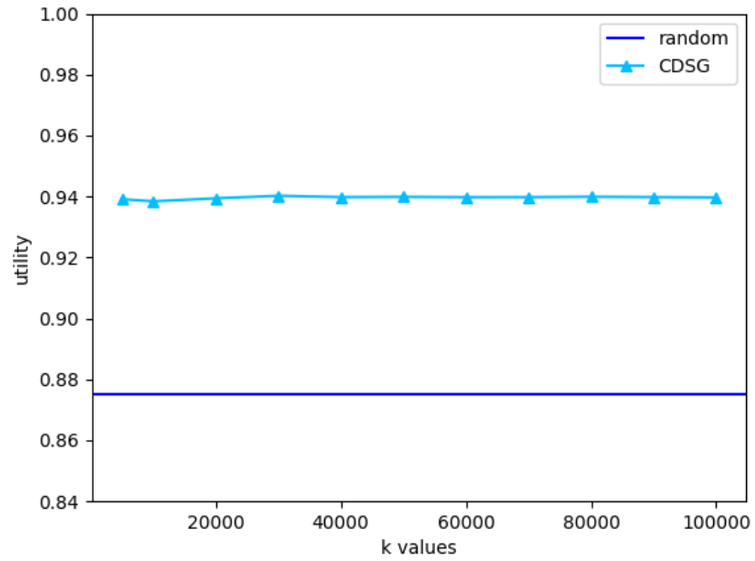

Figure 12: Effect of k parameter on Average Utility on uf50-218 test-set

Additionally we calculated the time (in seconds) that it takes to calculate to probabilities for each element, partial results are presented in the following table:

k	time (in seconds)
10	0.0105
50	0.0539
100	0.1222
500	0.6244
1000	0.9036
5000	6.3559
10000	14.5832
50000	52.9131
70000	72.2170
100000	103.4589

The above figures indicate that the highest utility at 0.94 is achieved at k value above 700. After this value, the utility does not change significantly even for k values as big as 100,000. From those funding we may conclude that the optimal k value, for the uf50-91 test set with the threshold 0.001 is 700.

Comparison of the results for uf20-91 and uf50-218 test-sets:
In both tests the peak utility is 0.94. On uf20-91 test set this utility is achieved with k values exceeding 300, while on uf50-218 test set this utility is achived with k values above 700. Beyond those thresholds, the utility remains relatively stable. Another difference between the test sets is noticeable for smaller k values, specifically those below 100. These results are illustrated in Figures 5 and 9. Notably, the uf20-91 test set demonstrates a faster attainment of the 0.94 utility value compared to the uf50-218 test set. The discrepancy in both cases arises from the differing sizes of the two test sets – the uf20-91 consists of 20 elements, whereas the uf50-218 comprises 50 elements. Consequently, the size of the random test set (k) required for calculating optimal contributions is influenced by the number of elements: the greater the number of elements, the larger the necessary k value.
Another difference that emerges from the results pertains to the time taken to compute the probabilities for each element within the distinct test sets. These outcomes are presented in the aforementioned tables. It's evident that calculating the probabilities in the uf50-218 test set consumed more time compared to the uf20-91 test set. This divergence can similarly be attributed to the varying number of elements in the test sets. More elements translate to longer computation times for probabilities.

4.3 Performance of Pattern Driven Subset Generation Algorithm

4.3.1 Effect of Max Depth of Regression Tree on Average Utility

The max depth parameter of the regression tree serves as the maximum threshold for the number of core elements in the optimal pattern.
The primary objective of this test was to evaluate the impact on the utility of the generated subset of modifying the max depth parameter in a regression tree, which in turn influences the maximum number of core elements in the optimal pattern. We conducted two evaluations: one using 20 instances from the uf20-91 test-set and another using 20 instances from the uf50-218 test-set. Both evaluations involved 1000 generated subsets. In the plotted graph, each dot corresponds to the average utility of 20 instances, where each instance represents the average of 1000 generated subsets.

Effect of max depth parameter tested on uf20-91 test-set:
In the following figure we visualize the effect of max depth parameter with a k=500 on average utility tested on 20 instances from uf20-91 test-set.

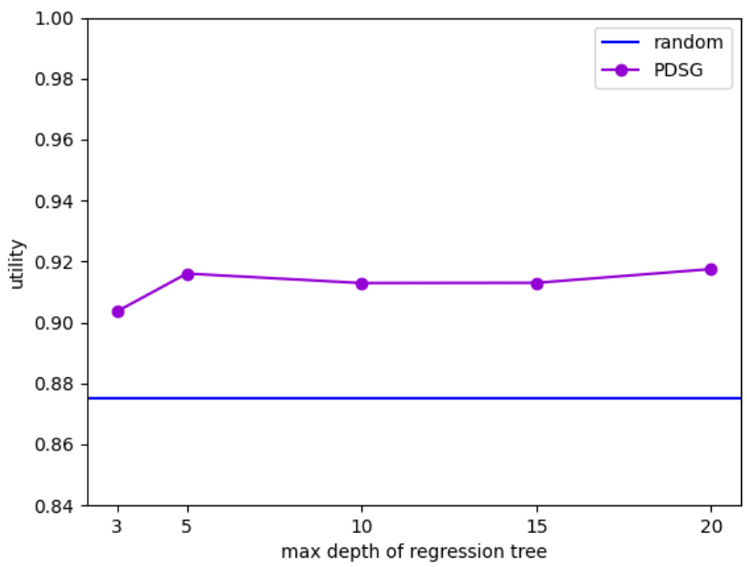

Figure 13: Effect of Max Depth Parameter with k=500 on Average Utility on uf20-91 test-set

In the following figures we visualize the effect of max depth parameter with a k=1000 on average utility tested on 20 instances from uf20-91 test-set.

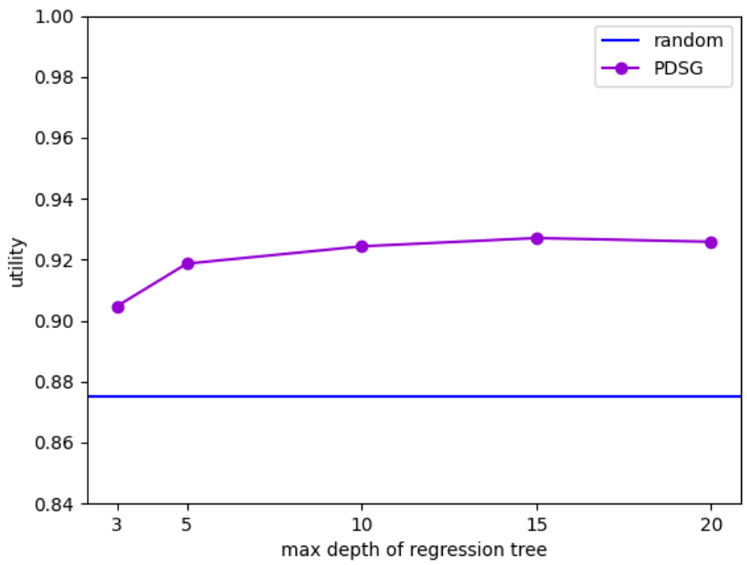

Figure 14: Effect of Max Depth Parameter with k=1000 on Average Utility on uf20-91 test-set

In the following figures we visualize the effect of max depth parameter with a k=1500 on average utility tested on 20 instances from uf20-91 test-set.

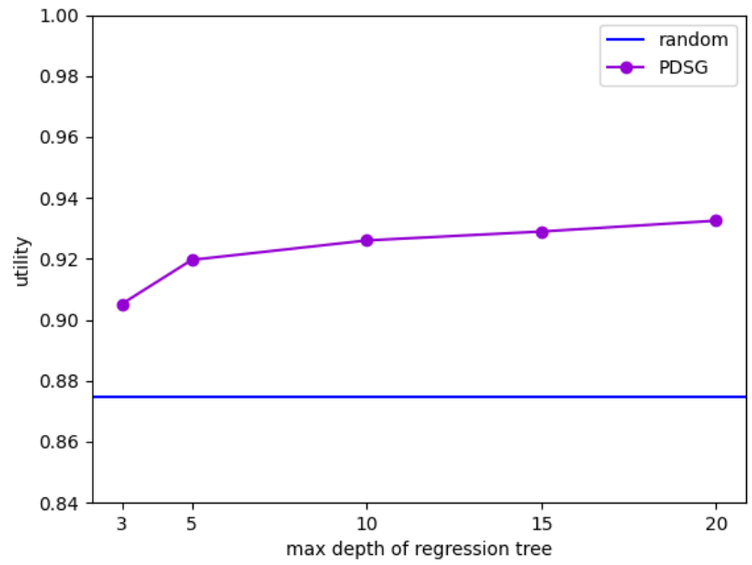

Figure 15: Effect of Max Depth Parameter with k=1500 on Average Utility on uf20-91 test-set

In the following figures we visualize the effect of max depth parameter with a k=10000 on average utility tested on 20 instances from uf20-91 test-set.

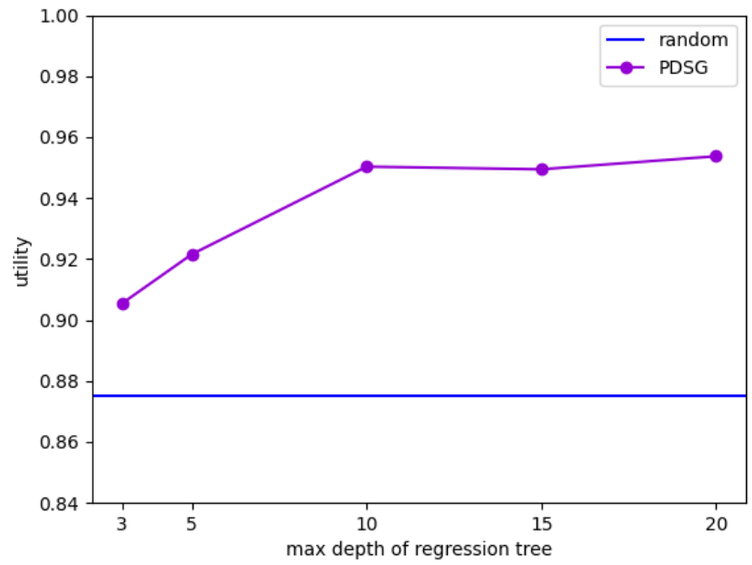

Figure 16: Effect of Max Depth Parameter with k=10000 on Average Utility on uf20-91 test-set

Evident from the presented figures is the minimal impact brought about by adjusting the max depth parameter within the regression tree on the resulting subset utility. Notably, a shallow depth within the range of 3-5 corresponds to lower utility, while changes in utility become negligible from a depth of 10 onwards. This phenomenon could be attributed to the upper limit for the depth of the regression tree. For instance, consider a regression tree with 20 features and a training set of size 10000 (k); its maximum depth, if the tree is uniform, would be approximately $log_2(10000) \approx 13$. Therefore, even with a max depth parameter in the range of 15-20, the actual depth of the tree might never reach those values due to this inherent limit.
Additionally, it's worth highlighting that variations in the k parameter do exhibit a notable impact on the observed outcome.

Effect of max depth parameter tested on uf50-218 test-set:
In the following figure we visualize the effect of max depth parameter with a k=500 on average utility tested on 20 instances from uf50-218 test-set.

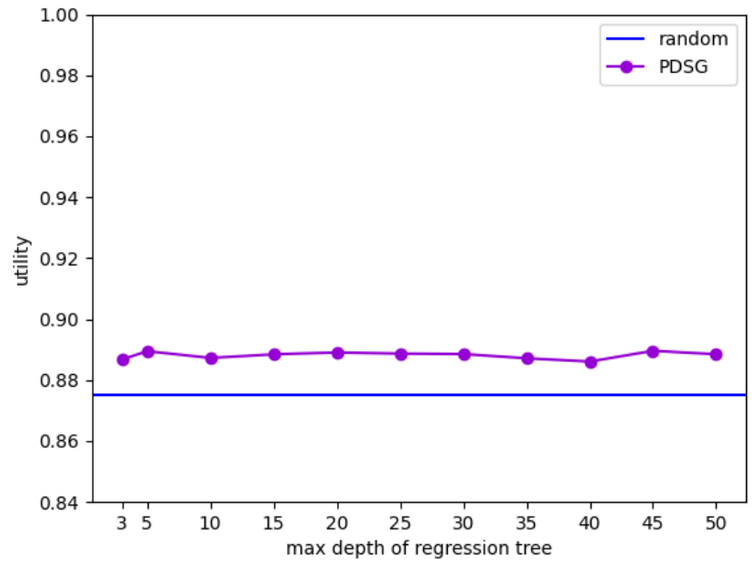

Figure 17: Effect of Max Depth Parameter with k=500 on Average Utility on uf50-218 test-set

In the following figures we visualize the effect of max depth parameter with a k=1000 on average utility tested on 20 instances from uf50-218 test-set.

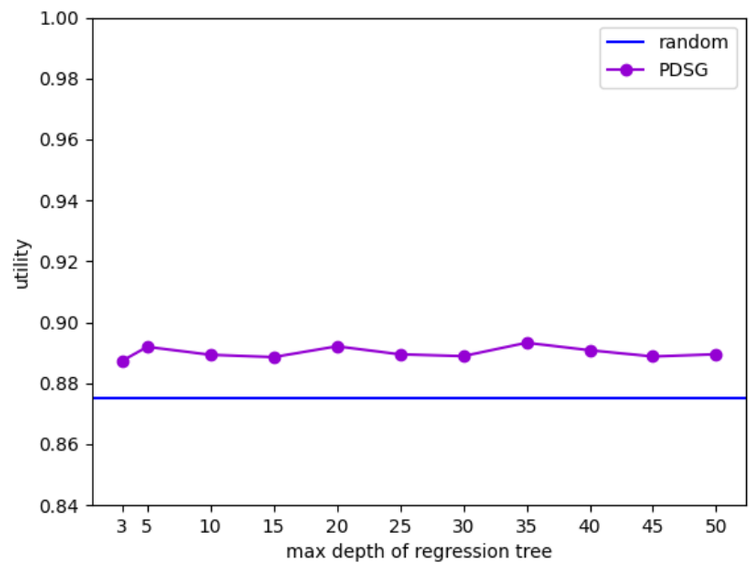

Figure 18: Effect of Max Depth Parameter with k=1000 on Average Utility on uf50-218 test-set

In the following figures we visualize the effect of max depth parameter with a k=1500 on average utility tested on 20 instances from uf50-218 test-set.

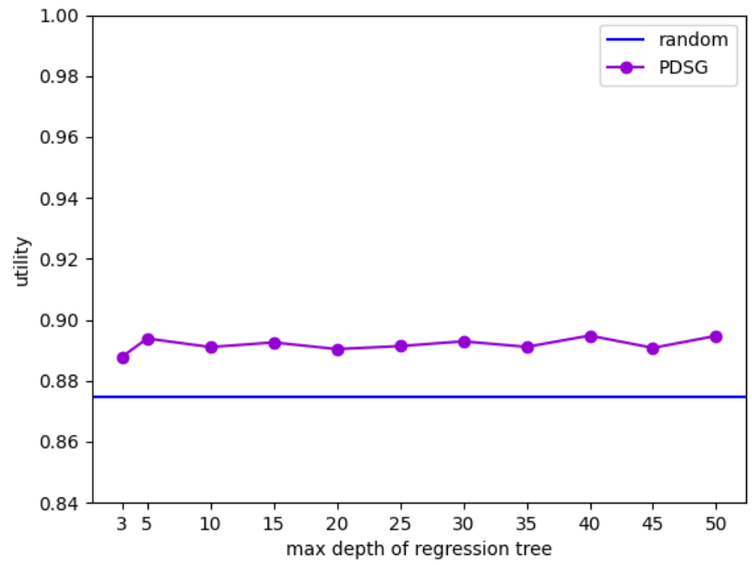

Figure 19: Effect of Max Depth Parameter with k=1500 on Average Utility on uf50-218 test-set

In the following figures we visualize the effect of max depth parameter with a k=10000 on average utility tested on 20 instances from uf50-218 test-set.

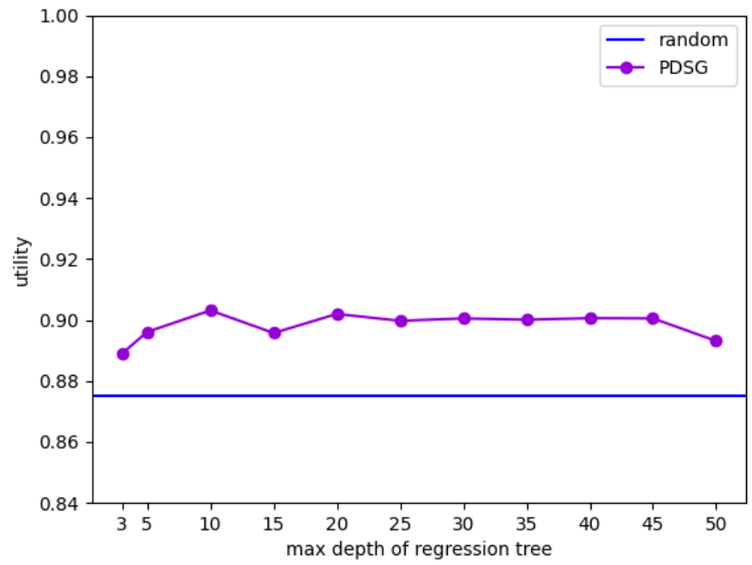

Figure 20: Effect of Max Depth Parameter with k=10000 on Average Utility on uf50-218 test-set

From the above figures it is apparent that max depth parameter within the regression tree has negligible influence on the resultant subset utility.

Comparison of the results for uf20-91 and uf50-218 test-sets:
The effect of the max depth parameter is more pronounced in the uf20-91 test set, whereas its influence is negligible in the uf50-218 set.

The most substantial disparity between the outcomes of the tests conducted on the uf20-91 and uf50-218 test sets lies in the range of utility values. In the uf20-91 test set, utilities spanned from 0.90 to 0.96, whereas the uf50-218 test set exhibited a narrower and lower range, fluctuating between 0.89 and 0.90. This evident drop in algorithm performance can be attributed to the distinct number of elements in the two test sets. The uf20-91 set encompasses 20 elements, while the uf50-218 set consists of 50 elements. It appears that an increased number of elements has a detrimental impact on algorithm performance. A plausible explanation for this lies in the potential depth of the regression tree. As mentioned earlier, the depth of a uniform regression tree, whether featuring 20 or 50 attributes and trained on 10000 (k) examples, is approximately $log_2(10000) \approx 13$. This implies that, for both the uf20-91 and uf50-218 test sets, the number of core elements is capped at 13.

In the case of the uf20-91 set with 20 elements, this limitation might not significantly affect results since more than half of the elements can be core elements. However, in the uf50-218 test set with 50 elements, less than one third of the elements can function as core elements. This discrepancy heavily impacts utility in a negative manner, leading to the observed decline in performance.

4.3.2 Effect of k Parameter on Average Utility

The upcoming tests aim to explore the influence of the training set's size in the regression tree, represented by the k parameter, on the average utility of the generated subsets. In these tests, we do not set the max depth parameter; instead, it is determined by the algorithm.

These assessments are conducted using 20 instances from both the uf20-91 and uf50-218 test sets. The algorithm's performance on each instance is assessed based on 1000 generated subsets. In the ensuing plots, each data point represents the average utility of 20 instances, either from the uf20-91 or uf50-218 test sets, evaluated across 1000 generated subsets.

Effect of k parameter with unlimited max depth on uf20-91 test-set:
The following figure depicts the effect of the following k values: [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000] tested on 20 instances from uf20-91 test-set.

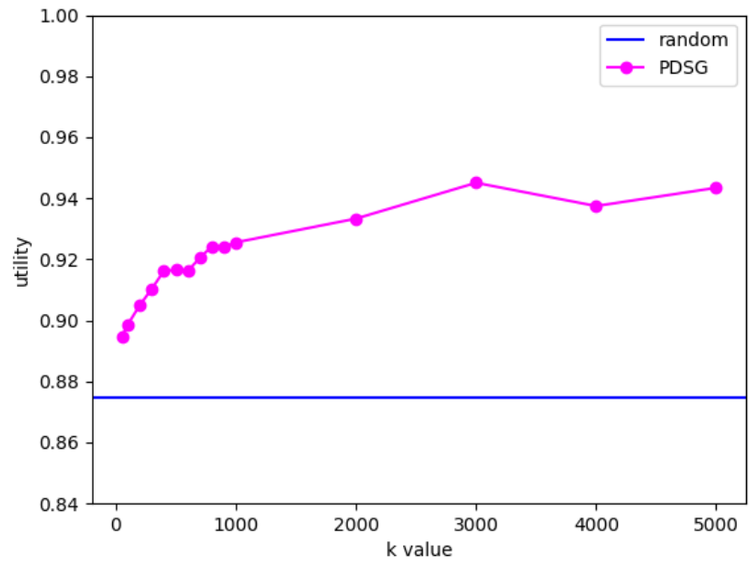

Figure 21: Effect of k parameter on Average Utility on uf20-91 test-set

The following figure depicts the effect of the following k values: [10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000] tested on 20 instances from uf20-91 test-set.

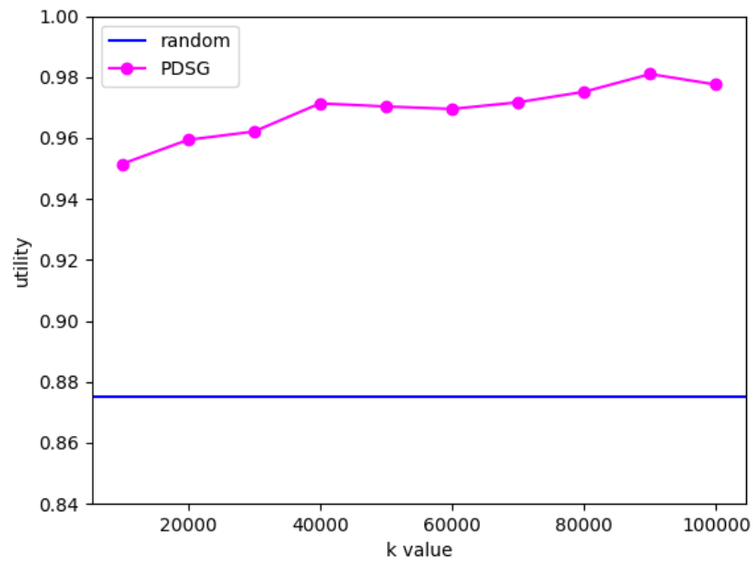

Figure 22: Effect of k parameter on Average Utility on uf20-91 test-set

Additionally we calculated the time that it takes to find the optimal pattern for different training set sizes (denoted by k), partial results are presented in the following table:

k	time (in seconds)
50	0.0081
100	0.0153
500	0.0620
1000	0.1604
5000	0.8143
10000	1.2308
50000	5.0923
70000	6.2379
100000	8.4932

From the figures above, it's evident that the k parameter has a significant impact on the average utility. Notably, there's a clear linear relationship between the k parameter and the average utility value. To be more precise, the utility increases most rapidly within the range of 50-1000. Afterward, the utility's growth rate decreases, ultimately reaching the value of 0.98

The computation time also rises as the size of the training set (denoted by k) increases.

Effect of k parameter with unlimited max depth on uf50-218 test-set:
The following figure depicts the effect of the following k values: [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000] tested on 20 instances from uf50-218 test-set.

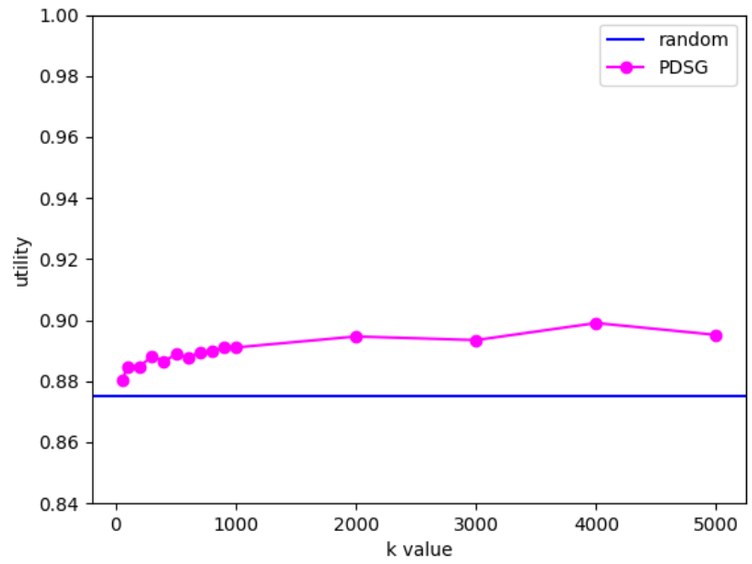

Figure 23: Effect of k parameter on Average Utility on uf50-218 test-set

The following figure depicts the effect of the following k values: [10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000] tested on 20 instances from uf50-218 test-set.

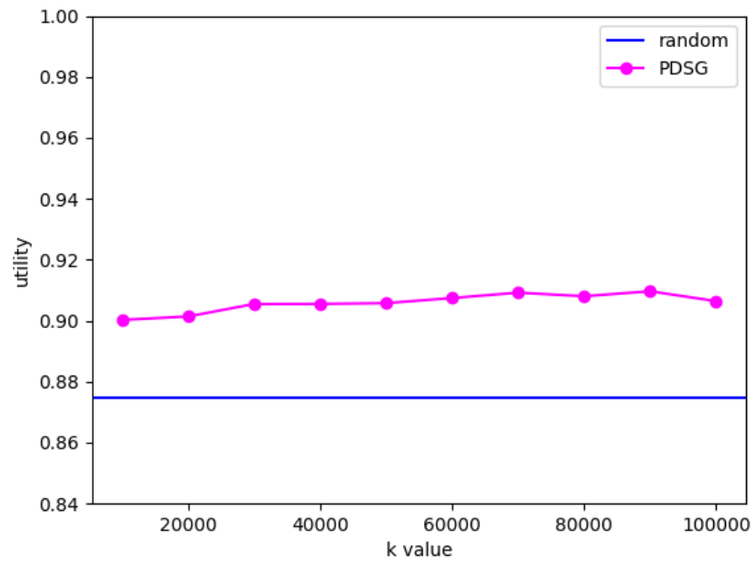

Figure 24: Effect of k parameter on Average Utility on uf50-218 test-set

Additionally we calculated the time that it takes to find the optimal pattern for different training set sizes (denoted by k), partial results are presented in the following table:

k	time (in seconds)
50	0.0119
100	0.0219
500	0.1031
1000	0.2071
5000	1.0319
10000	2.0557
50000	10.3554
70000	14.5820
100000	32.9925

Based on the figures above, it's evident that the k parameter has a mild impact on the average utility. The most substantial increase in utility occurs within the k range of 0 to 2000, where it climbs from 0.88 to 0.90. Beyond this range, the utility remains relatively stable around 0.91, even for very large k values such as 100,000.

In contrast, the computation times increase significantly with the expansion of the training set size (indicated by k).

Comparison of the results for uf20-91 and uf50-218 test-sets:
Based on the conducted tests, it's evident that the algorithm excels when applied to the uf20-91 test set, achieving an average utility of 0.98 with a training set size of 100,000. Conversely, when tested on the uf50-218 set, the algorithm only reaches an average utility of 0.91, even with a training set of the same size.

Another notable distinction in favor of the uf20-91 set is the computation times, which are considerably shorter (just 8.4932 seconds for a training set of size 100,000) compared to the uf50-218 test set (32.9925 seconds for the same size training set). Similar to the earlier tests on the max-depth parameter, this difference may be attributed to the increased number of features in the uf50-218 test set compared to the uf20-91 test set.

We hypothesize that with a significantly larger training set, the algorithm could potentially perform equally well on the uf50-218 test set as it does on the uf20-91 test set. However, it's important to consider that this approach may be inefficient, primarily because of prolonged computation times.

5 Conclusions and Future Work

In this work, we have presented an innovative method for solving subset selection problems, utilizing the contributions of elements within the set to generate optimal subsets.

Our work has produced two distinct algorithms employing this methodology. The first algorithm, CDSG, calculates the contribution of each element independently, while the second, PDSG, takes a more sophisticated approach, considering interdependencies among elements and leveraging this insight to compute contributions. PDSG accomplishes this by learning patterns within subsets and predicting their utility.

To evaluate the performance of these algorithms, we conducted tests on two datasets, uf20-91 and uf50-218 from the SATLIB benchmark problems library.

On the uf20-91 test set, the PDSG algorithm outperformed CDSG, achieving a utility of 0.98 with a training set size of 100,000, compared to CDSG's 0.95 utility with the same training set size. Conversely, on the uf50-218 test set, the initial algorithm, CDSG, exhibited superior performance, reaching a utility of 0.94 with a training set of 100,000, while the more complex PDSG algorithm attained a utility of 0.90 with the same training set size.

In terms of computation times, PDSG significantly outpaces CDSG on both test sets.

The reversal in performance between the two algorithms on the uf50-218 test set is likely due to the increased number of features (elements) present in the test set compared to the uf20-91 test set.

We see significant potential in refining the performance of the PDSG algorithm by influencing the core values in learned patterns. Our aim is to achieve this enhancement without significantly enlarging the training set size and incurring longer computation times, which could otherwise impede the algorithm's efficiency. This represents a valuable direction for future research.