Programmatic optimization of ad pods for maximizing consumer engagement and revenue

Niranjan Kumawat
n.kumawat@samsung.com
Samsung Research Institute
Bangalore, Karnataka, India

Manu Vajpai
manu.vajpai@samsung.com
Samsung Research Institute
Bangalore, Karnataka, India

Nitish Varshney
nitish.var@samsung.com
Samsung Research Institute
Bangalore, Karnataka, India

ABSTRACT

CTVs serve streaming content to users via the internet or a local network. These devices replicate the traditional TV experience and can serve sequential ads which are bundled together and then inserted into content. This bundling of ads is accomplished by ad-podding. However, un-optimized display of ads in these pods can impact consumer engagement and revenue. This work presents one of the first programmatic approaches to achieve optimal packaging of ads in pods with multiple constraints. We first establish the underlying problem as a multi-objective Knapsack problem and then explore established approaches of evolutionary algorithms and dynamic programming to tackle it. We further propose a new greedy approach which solves the problem efficiently for online deployment. Testing, with our in-house and public datasets, establishes the relative effectiveness and applicability of these approaches for optimizing podded creatives.

CCS CONCEPTS

• **Information systems → Computational advertising**; **Display advertising**.

KEYWORDS

Digital advertising, Computational advertising, Ad-Podding, Knapsack, Evolutionary algorithms, Greedy algorithms, Real-time bidding

1 INTRODUCTION

Over-The-Top (OTT) media services, like Samsung TV Plus, Disney+, Netflix, and Hotstar have emerged as major players in the media space with millions of hours of content streamed every day. The emergence of Connected Televisions (CTVs) have further increased market penetration for OTT media. CTVs accomplish this by integrating the convenience of traditional TVs with the connected nature of OTT services.

Monetization is achieved for CTVs in a similar fashion to the mobile advertising ecosystem – the process starts with an impression consumer (IC) requesting an impression provider service (IPS) for a possible ad-impression opportunity. This request is then forwarded to a Real-Time Bidding (RTB) system, where multiple Impression Buyers (IBs) bid. The winner is allocated the impression and is charged a fee, using an auction mechanism of choice, e.g. first-price or second-price auctions [9, 11].

In CTVs, these ad opportunities usually come in the form of linear ad breaks, which can be referred to as ad-pods in the advertising space. Ad-pods are sets of sequential ads, which are played together in an ad break. This concept of ad-pods has direct roots in the television medium and is otherwise absent in the traditional digital advertising ecosystems. As per the Video Ad Serving Template (VAST) 3.0 specification, pods contain one VAST response with multiple linear ads present in the sequence attribute [1]. Ad pods have a limited duration and allow bundling of multiple ads within the constraints defined by the industry.

The advantages of ad podding are two-fold. Firstly, it decreases consumer fatigue by making sure that content is not cluttered by individual ads. Secondly, it increases the number of available impressions in a single ad slot, since multiple ads can be viewed by the same end-user, thereby increasing the potential revenue and exposure. It has been reported that ad-podding results in a 50% increase in revenue [3].

However, indiscriminate bundling of ads can result in a reduction of consumer engagement for brands. This issue has been observed in traditional TV as well and arises because of the presence of ads with conflicting branding and content [7]. To resolve such issues, ad industry allows creation of ad pods with constraints on similar ads from different impression buyers (IBs), which would otherwise result in consumer fatigue. Hence, usually same category or same top level web domain ads are not allowed together. Similarity of categories is usually identified by Interactive Advertising Bureau (IAB) ad category details shared by the IBs [4].

Additionally, indiscriminate podding does not ensure optimal ad revenue as there are variations in both the duration of, and revenue obtained from, constitutive ads. To further alleviate the challenge, these problems have to be addressed in ways that ensure that the solution is scalable and efficient. Scalability is essential for deployment in a large-scale RTB setup that serves millions of clients concurrently. Meanwhile, efficiency is required to meet the low-latency requirements of the same.

The contributions of this work, towards optimization the three indicators – consumer engagement, revenue and scalability – for ad pods, are as follows:

(1) Formalize the problem of optimal packaging in CTV ad-pods as a multi-objective 0/1 Knapsack problem.
(2) Propose multiple solutions, notably an evolutionary approach which can be tuned to focus on consumer engagement or revenue, followed by a greedy approach which generates

AdKDD'22, August 2022, Washington, DC, USA
© 2022 Association for Computing Machinery.

near-optimal solutions, without any compromises in scalability.

(3) To compare said approaches in the context of scalable deployment in an RTB environment.

2 MODELING AS MULTI-OBJECTIVE 0/1 KNAPSACK PROBLEM

Let us consider an ad-podding opportunity I requested by an impression consumer (IC) while viewing long-form video content or live stream. The ad-pod has a limited duration D and a fixed number of displayable ads N. The total duration D is modeled as the capacity of the knapsack representing the ad-pod. B represents the set of all bids (and corresponding ads) available for selection. For I, IC sends an impression provider service (IPS) a multi-impression ad request for the ad-pod, stating D and N. IPS further requests for bids to M impression buyers (IBs). Depending on the opportunity each IB responds with multiple bids, numbering k ($k <= N$). For the sake of simplicity, this work assumes that all IBs/bidders respond with exactly N bids ($k = N$), resulting in the accumulation of K total bids at the IPS, where $K = M * N$.

Each bid $b_i \ \in B, i \in \{1, 2, 3, \text{ﬁ.}, K\}$, is associated with following:

- a_i : identifier of the linear ad
- p_i : cost to display a_i or it's bidding price.
- d_i : duration of a_i.
- C_i : set of IAB categories of a_i.
- A_i : set of ad domains of a_i.

A real-time bidding (RTB) auction is carried out with the bids b_i to select the winning IB. However, unlike traditional RTB protocols where bid values or effective Cost Per Mille (eCPM) majorly dictate the allocation scheme, the winner is not decided by comparing just the bid values. The primary difference arises owing to the objective for an IPS to maximize revenue while ensuring that the total duration of the pod is not breached. i.e.:

$$max \sum_{i=1}^{K} p_i x_i, \ s.t. \sum_{i=1}^{K} d_i x_i \leq D$$

$$where, \ x_i = \begin{cases} 0, if \ b_i \notin S \\ 1, if \ b_i \in S \end{cases}$$

$$\forall b_p, b_q \in S, \ C_p \cap C_q = \phi \ or \ A_p \cap A_q = \phi \tag{1}$$

Here, $S = \{b_p, b_q, b_r, \ldots, b_s\}$ represents the solution set corresponding to the optimized ad-pod, with $\forall b_i \in S; 0 \leq i \leq N$. For a given pod I the selection of bids, as represented in equation 1, can be viewed as a multi-objective Knapsack Problem. We refer to this selection of bids as *ad podding auction.* With only capacity as a constraint in equation 1, it represents a single-objective 0/1 Knapsack problem, for which optimal solutions exist. However, the introduction of constraints on bid's (ad's) category or ad doma in significantly increases the problem's complexity, resulting in a Multi-Objective 0/1 Knapsack (MOK) problem. The algorithms that we have used to solve this problem and the reasons for their choices are discussed in the upcoming sections.

3 RELATED WORK

MOKs are well-studied problems and are known to be notoriously difficult to solve since the general formulation is NP-hard in time complexity [19]. Several algorithms that have been used to solve a subset of these problems efficiently, in the available literature. These can be sorted into three categories of *exact, heuristic, and deep learning-based approaches.*

Dynamic programming is a pivotal exact approach used to solve the diluted variant of 0/1 knapsack problems [18]. It has been extended to MOK case by adding the handling of multiple objectives using a multiple-choice knapsack methodology [22]. Another commonly used method to solve 0/1 knapsack problems is the branch and bound algorithm [6]. Both of these methods suffer from exponential time complexity with respect to the number of constraints on the knapsacks, which can be detrimental for RTB deployment. Frolios *et al.* have obtained the exact solution of a subfamily of MOKs, using branch and bound algorithm.

Heuristic solutions are known to converge within reasonable time scales. These are approximate approaches that do not guarantee the optimal solution, though. They trade-off accuracy for computational complexity. However, with the appropriate choice of parameters, they can converge to, or very near to, optimal accuracy levels. There are three subcategories of heuristic approaches used to solve multi-objective 0/1 knapsack problems as listed below:

Evolutionary algorithms (EAs) are heuristic approaches to solve multi-objective problems [25], in particular, they have been used extensively to solve problems in the advertising industry [8, 13]. They are preferred for MOKs because of their relative accuracy and fast convergence [24].

Swarm algorithms have also emerged as viable solutions to multi-objective problems, and have been used in problem-domains ranging from delivery optimization to modeling physical systems [21]. Derivatives like strawberry optimization have been shown to efficiently solve MOKs [20]. They are also heuristic approaches but are generally observed to converge towards better solutions than evolutionary algorithms, in terms of solution quality, but at the expense of running time.

Greedy formulations, that are tailored to the problem, can be vastly more efficient than generic optimization approaches, especially when coupled with machine learning or reinforcement learning. Hao *et al.* have proposed their Multi-channel Sequential Budget Constrained Bidding (MSBCB) approach to maximize revenue for advertisers under budgetary constraints, for prolonged ad exposure to consumers [15]. This work highlights the possibility of using multiple optimization approaches together to solve the problem, since it employs both reinforcement learning and greedy steps.

With the emergence of deep learning in the optimization landscape, several problems have been addressed by applying deep neural networks to them. Pointer networks are a prime example of such approaches [23]. Pointer networks have been shown to be efficient at solving 0/1 Knapsack problems using a transformer-based seq-to-seq architecture [14]. Other works address the MOK problem using deep reinforcement learning models [16]. A major downside of using deep learning models is their requirement for a large amount of data for training. Additionally, the runtime requirements of complex models can be a hindrance to RTB deployment.

4 METHODOLOGY

We have explored three approaches – dynamic programming, evolutionary algorithms and a tailored greedy approach – to solve the ad-podding problem. Each of these approaches has their own merits and demerits which are further explained in the coming subsections.

4.1 Dynamic programming for Multiple-choice knapsack

Dynamic programming (DP) provides the exact solution to the ad-podding problem by treating it as a multiple-choice knapsack problem [22]. This is done by first grouping together ads based on the constraints and restricting item selection based on the grouping while filling the DP matrix. The biggest drawback of this approach is that computational complexity increases with the number of objectives as $O(DK^2 + f(K, C))$. Here $f(K, C)$ is a function dependent on the number of bids available for selection (K) and the number of constraints (C). D is the total duration of the pod. For our case $O(f(K, C)) = O(K)$, since the only set of constraints in the grouping is with respect to categories. When multiple constraints are present from other groupings the search space can become infeasible. This is major drawback in an RTB environment where multiple constraints of ad categories, domains, and IB preferences are applied. However, DP is still effective as a benchmark, since brute-force solutions are inherently inefficient.

4.2 Evolutionary algorithms based approach

To optimize our MOK with EAs we initialize the initial population for the EA as randomly selected sets T_j $(0 \le j \le P_{ind})$, where P_{ind} is the number of individuals specified in the algorithm. The cardinality of the set of ads available for selection is K, which are used to generate P_{ind} pods of size N. These are then evolved for G generations with mutation probability μ and crossover probability θ. Mutation is achieved by swapping an ad from the pod with another ad from the set available for selection. Crossovers are achieved by splitting two ad pods (knapsacks) of current generation and swapping the end segments between the two. This results in the first pod having the second pod's end segment and vice versa. We then use the NSGA-II algorithm to select the next generation [10]. The objective functions are tuned for maximizing revenue, minimizing similarity of IAB categories and enforcing the knapsack capacity constraint. This implies that there are three components of the multi-objective penalties, one corresponding to each of the constraints. The three conditions are enforced as:

- Very high penalty to ensure domination for any bag that crosses the capacity constraint.
- Similarity penalty calculated as the product of a moderate penalty (σ) and the number of ad pairs in a pod with the same categories/domains, using equation 2. Here σ is a tunable parameter, i and j are indices over the ads in the pods and N is the number of ads in the pod. C_i and C_j are categories of i^{th} and j^{th} ads in the pod, respectively.
- The total value of the ads in the knapsack as the sum of eCPMs (effective cost per mille).

$$S = -\sigma \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} [C_i = C_j] \tag{2}$$

4.3 Greedy approach

When the selection of items in a 0/1 knapsack is either complete or partial, greedy algorithms provide an optimal solution [17]. However, since we are handling multiple objectives as constraints, a greedy approach will converge to a near-optimal solution. Our greedy approach incorporates the category and domain constraints discussed in equation 1. For each bid B_i we define Price per Duration Ratio (PDR) of a given I as the heuristic in equation 3.

$$PDR_i = \frac{p_i}{d_i} \tag{3}$$

Then, algorithm 1 is utilized to calculate the solution set S and revenue P_{total}.

Algorithm 1: Greedy algorithm for solving ad-podding MOK

```
Result: Optimized ad-pod $S$ and revenue $P_{total}$
Initialize selected categories ($C_{sel}$) as an empty set;
Alternatively, initialize selected ad domains ($A_{sel}$) as an
  empty set;
Initialize cumulative profit ($P_{total} \leftarrow 0$) and remaining
  duration ($D_{rem} \leftarrow D$);
sort B w.r.t. heuristic in descending order;
for $b_i \in B$ do
    if $D_{rem} - d_i \ge 0$ and $C_i \notin C_{sel}$ and $A_i \notin A_{sel}$ then
        $P_{total} \leftarrow P_{total} + p_i$;
        $D_{rem} \leftarrow D_{rem} - d_i$;
        $C_{sel} \leftarrow C_{sel} \cup C_i$;
        Alternatively, $A_{sel} \leftarrow A_{sel} \cup A_i$;
        $S \leftarrow S \cup b_i$;
    end
end
```

The PDR heuristic ensures that ads with a higher price to duration ratio are picked first to maximize the value obtained from each item added to the knapsack. However, when the bids have a similar PDR ratio there is a possibility that algorithm 1 will fill the knapsack with sub-optimal bids, lowering the cumulative profit. To prioritize profit in such a scenario, we have defined another heuristic in equation 4 as the Price per Duration Ratio with Price prioritized (PDRwP). This heuristic has an added factor of p_i, which prioritizes ads with higher price when p_i/d_i ratio is equal.

$$PDRwP_i = p_i\left(1 + \frac{1}{d_i}\right) \tag{4}$$

These two heuristics allow us to prioritize either the maximum number of ads or the maximization of revenue. The time complexity of both approaches is bounded by the time complexity of sorting B, i.e. $O(K \log(K))$, where K is the number of all the ads/bids available for selection.

As a note for practical implementation, these heuristics can reach unstable values when ads of near-zero duration are podded.

However, this is not an issue in practice because such small ads are usually filtered out from the bid responses. They can also be removed from the selection-set when applying the algorithm.

5 EXPERIMENTS

5.1 Datasets

We have used two datasets for the evaluation of the selected algorithms. The first is obtained from our auction logs and the second dataset is composed of ads from YouTube^{TM}, here onwards referred to as YT dataset [5]. This dataset is publicly available and can be downloaded from https://github.com/marianavsarantes/video-ads-dataset.

Our in-house dataset is obtained from auction-logs spanning over one week. It contains ads from 15 IAB categories that were found to be most prominent during the period. It contains category, domain, duration, and bid-value information for each advertisement. YT dataset is composed of more than 5000 video ads and their metadata. Because of the lack of cost per mille (CPM) values for bids in the YT dataset, we have modeled the same from the viewership (V) for each ad as V/F, where F is a factor used to scale down viewership. For YT dataset we chose F as 1,000,000.

5.2 Experimental setup

Baselines have been established using two approaches, both guaranteeing an optimal solution. The first approach is backtracking based, where knapsacks were built iteratively from the available selection of ads. Throughout backtracking, non-optimal knapsacks were eliminated, leaving the optimal knapsacks as the solutions. Knapsacks that exceed capacity or violate IAB category constraints were rejected to ensure efficient search, akin to pruning the DFS tree.

The second baseline was dynamic programming for multiple-choice knapsacks, using IAB category as the grouping scheme, following the approach of Sinha *et al* [22]. For this study, we have enforced similarity constraints only on IAB category codes since DP cannot cope up with the extra complexity introduced by additional constraints. This happens because the time-complexity of DP is dependent upon the cardinality of the set of all possible constraint combinations.

Our first exploratory approach, based on evolutionary algorithms, was implemented for ad-podding using NGSA-II algorithm. For these experiments, we varied the number of generations as $ceil(50 * (N/20))$, where $ceil$ returns integer ceiling of the number. Similarly, population size was varied as $ceil(70 * (N/20))$. We chose a high penalty of 1000 for knapsacks that cross capacity constraints. For similarity of categories, we chose σ as 100, and evaluated pods with equation 2.

For our second set of explorations, we have used both of our greedy heuristics and obtained two sets of greedy solutions. Greedy-PDR (G-PDR) was run with our initial heuristic and greedy-PDRwP (G-PDRwP) used the adjusted heuristic with profit prioritized. Each experiment was repeated 1000 times, for each knapsack size (N), to calculate deviations from exact solutions, and for estimating the computational requirements.

All computations were performed on a single memory-optimized virtual machine on Amazon Web Services (AWS) with 64GB of available RAM. The machine was configured with Ubuntu 18.04 LTS OS and Python 3.8, and numPy was used to optimize execution. EA was implemented using the DEAP library [12].

5.3 Experimental results

5.3.1 In-house dataset. To assess the accuracy of the results from these methods, we analyzed the profits obtained from each of them. This was done by calculating percentage deviation in profit, from the benchmark backtracking and DP solutions, using equation 5. Multiple test cases were run for each approach to get an estimate of it's performance.

$$Dev = \frac{1}{R}\sum_{t=1}^{R}(\frac{abs(\alpha_t - \beta_t)}{\beta_t} * 100) \tag{5}$$

Here α_t is profit from the selected approach and β_t is same from the optimal approaches of backtracking and DP. Both profits are means obtained from R total executions, with individual executions indexed by t. We have calculated percentiles of average deviation in profit (Dev) for $R = 5000$ of such executions per algorithm. The results for this analysis are collated in table 1. For reference, higher is the percentile value, higher is the deviation from optimum profit established by the optimal solution.

Table 1: Percentiles of average percentage deviation from profit of the selected algorithms, for in-house dataset.

Percentile	DP	EA	G-PDR	G-PDRwP
50th	0.0	47.57	11.84	0.0
95th	0.0	111.97	33.11	14.38
99th	0.0	133.64	41.94	29.05

As expected, dynamic programming has no profit deviation from the backtracking solution as it is also a deterministic approach that converges to exact solutions. For both datasets, EA shows a higher deviation than DP on the virtue of being a general heuristic solution. Greedy PDR (G-PDR) and PDRwP (G-PDRwP) based approaches show very low-profit deviation, with G-PDRwP outperforming all other heuristic solutions. To highlight it's accuracy, it has zero deviation from optimum profit at 50th percentile, and near-zero deviation at 95th and 99th percentiles.

Figure 1 shows the histograms for density of percentage profit deviation for our in-house dataset, plotted against pod size. These further substantiate the results from table 1. EA shows the highest spread profit of all approaches, irrespective of the number of ads, with profits generally getting overestimated compared to backtracking and DP. This overestimation occurs at the expense of ad similarity in the pods with most solutions being sub-optimal. In several cases, EA failed to manage both profit and ad category similarity, as overall objectives were not satisfied across the experiments (data not shown). However, with multiple runs of EA, some solutions can be obtained, which show a preference for either higher consumer engagement or revenue, at the expense of the other. This can be helpful in fine-tuning between the two factors, as per IB and IPS requirements.

Greedy approaches show lower deviation and result in solutions that have comparable profits with exact solutions, while managing to have low or no similarity across ads. Since we are observing densities for greedy approaches which are mostly centered around zero, the relatively sparse numbers of low-similarity pods with non-optimal solutions are not visible in the histogram.

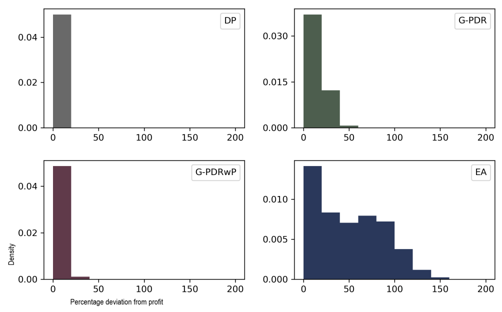

Figure 1: Density of percentage deviation of profit across different algorithms on in-house dataset.

In addition to profit and engagement, an important consideration in deploying any production solution for RTB environments is its execution time. Any latency introduced while processing these knapsacks during an auction can result in revenue loss because of timeouts to impression providers and buyers. In an RTB setup the total response time is limited to a few hundred milliseconds, between the bidder and auctioneer. While the internet latency between ad consumers (users) and auctioneer is on the order of hundreds of milliseconds, the relevant stakeholders in this problem are the advertisers/DSPs. Their internet latencies typically are in the order of tens of milliseconds. Since the RTB system has other functional components, which consume the available time for response, any reduction in latency is relevant.

To assess the performance of each of the approaches we monitored the execution time of each approach, which we averaged over 1000 iterations, for both the datasets. The results in table 2 showcase the impressive performance of our greedy approaches compared to other solutions. While the backtracking solution starts with feasible quick iterations, as the knapsack size increases (no of bids - N), its performance quickly degrades. The same is true for dynamic programming but its performance is better than backtracking here. This is because the latter needs to generate a large number of pods and hence needs multiple array-copy operations, unlike DP which needs to fill one DP matrix. EA performs worse than all other solutions apart from backtracking.

5.3.2 YouTube dataset. Similar results, as those for in-house dataset, are obtained for percentiles of average deviation from profit for the YT dataset, as shown in table 3, further corroborating our claims towards the accuracy of our greedy approaches. EA is still outperformed by both of our greedy heuristics, in particular PDRwP. Both heuristics are very close to exact solutions. The PDR heuristic performed slightly worse in this case as compared to the in-house dataset. However, the accuracy of PDRwP heuristic has improved to nearly the optimal level, with zero average deviation, even at 99th percentile.

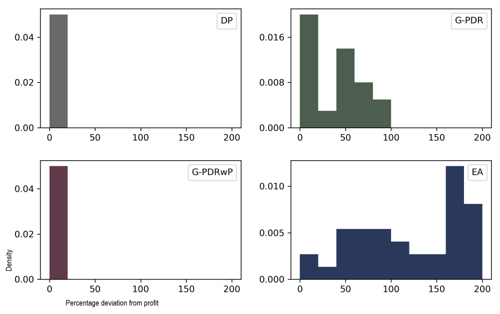

Figure 2: Density of percentage deviation of profit across different algorithms on Youtube dataset.

The results are validated by the density of percentage deviation from profit in figure 2. EA still has the highest variance in the density with DP giving 0 deviation across all solutions and pod sizes. Our G-PDR solutions have minimal deviation from profit. The G-PDRwP heuristic shows even better performance in the YouTube dataset and has minimal deviation from profit.

Average execution time shows similar behavior as in-house dataset for YT dataset, as documented in table 2. Our greedy approaches show at least six orders of magnitude improvement in execution time compared to backtracking and evolutionary algorithms, and around 100-fold improvement over DP. This result, combined with the relative accuracy of G-PDRwP, establishes the validity of our greedy approach in delivering a podding solution viable for production deployment.

6 CONCLUSIONS

We have devised the first formal study of optimal packaging of ads in an ad-pod for linear ads in connected TV ad breaks. Our results show that our greedy approaches outperform all other solutions on the factors of accuracy or computational efficiency by significant margins. They show low execution time which makes them feasible for production deployment in a real-time bidding setup with comparable accuracy to exact solutions. Our PDRwP based approach, in particular, generates optimal results, while being the most efficient and scalable. We believe that further exploration of this problem-domain with greedy or deep learning-based solutions have the potential to significantly improve profit and brand equity for publishers and engagement for users.

7 FUTURE DIRECTIONS

Some additions which we have not addressed in this work, that can be directly implemented in our solution as constraints originate from the specific requirements imposed by Impression Buyers (IBs). For example, an IB can request that a specific ad in the pod should be displayed first. An IB can also enforce that all the ads in their bid should be picked for podding, or their bid is invalid as allowed by the OpenRTB standard [2]. Additional constraints can also be enforced by capping ad frequency across the ad-pods displayed in

Table 2: Average execution time of selected algorithms, per iteration.

Bids in pod	Dataset	Backtracking	DP	EA	G-PDR	G-PDRwP
5	In-house	0.108 ± 0.0	0.256 ± 0.0	46.7 ± 0.235	0.019 ± 0.0	0.020 ± 0.0
	YT	0.143 ± 0.0	0.409 ± 0.001	48.2 ± 0.18	0.027 ± 0.0	0.028 ± 0.0
10	In-house	1.53 ± 0.004	0.914 ± 0.0	95.4 ± 0.512	0.045 ± 0.0	0.045 ± 0.0
	YT	1.05 ± 0.003	0.730 ± 0.003	98.8 ± 0.421	0.038 ± 0.0	0.038 ± 0.0
15	In-house	3.63 ± 0.013	1.17 ± 0.002	172 ± 0.308	0.055 ± 00	0.055 ± 0.0
	YT	9.35 ± 0.015	1.48 ± 0.038	140 ± 0.488	0.049 ± 0.0	0.051 ± 0.0
20	In-house	22.9 ± 0.061	2.13 ± 0.013	203 ± 0.786	0.067 ± 0.0	0.067 ± 0.0
	YT	23 ± 0.153	2.33 ± 0.01	195 ± 0.407	0.076 ± 0.0	0.074 ± 0.001
25	In-house	89 ± 0.290	2.12 ± 0.005	261 ± 2.17	0.083 ± 0.0	0.082 ± 0.0
	YT	86.2 ± 1.09	2.27 ± 0.046	248 ± 1.17	0.080 ± 0.0	0.081 ± 0.0
30	In-house	191 ± 1.11	3.54 ± 0.015	329 ± 1.74	0.104 ± 0.0	0.104 ± 0.0
	YT	126 ± 0.281	3.68 ± 0.032	305 ± 1.94	0.091 ± 0.0	0.093 ± 0.0
40	In-house	1000 ± 3.21	4.54 ± 0.011	462 ± 6.88	0.111 ± 0.0	0.123 ± 0.0
	YT	928 ± 4.07	4.95 ± 0.014	440 ± 4.45	0.113 ± 0.001	0.116 ± 0.0
50	In-house	3540 ± 18.1	5.96 ± 0.022	611 ± 1.05	0.125 ± 0.0	0.160 ± 0.0
	YT	3350 ± 11.8	8.3 ± 0.044	581 ± 4.34	0.155 ± 0.0	0.162 ± 0.001

Running times of the algorithms for varying pod sizes. Both in-house and YouTube dataset are listed. Time units are milliseconds.

Table 3: Percentiles of average percentage deviation of profit for the selected algorithms, for YouTube dataset.

Percentile	DP	EA	G-PDR	G-PDRwP
50th	0.0	152.73	15.26	0.0
95th	0.0	294.80	79.95	0.0
99th	0.0	405.45	87.62	0.0

a single long-form content. This can involve serving shorter ads at the end of an ad pod, to increase consumer engagement. These additional constraints can be implemented in our greedy approach by extending the formulation.

REFERENCES

[1] IAB 2012. *Video ad serving template (VAST) version 3.0.* IAB. Retrieved May 15, 2021 from https://www.iab.com/guidelines/vast/

[2] IAB 2016. *OpenRTB (Real-Time Bidding).* IAB. Retrieved May 15, 2021 from https://iabtechlab.com/standards/openrtb/

[3] Google ad manager 2019. *Smarter Ad Breaks provides control and flexibility to maximize fill rates and revenue in personalized commercial breaks.* Google ad manager. Retrieved May 15, 2021 from https://iabtechlab.com/standards/openrtb/

[4] IAB 2020. *Content taxonomy mapping v2.2.* IAB. Retrieved September 01, 2021 from https://iabtechlab.com/standards/content-taxonomy/

[5] Mariana Arantes, Flavio Figueiredo, and Jussara M Almeida. 2018. Towards understanding the consumption of video-ads on youtube. *The Journal of Web Science* 4 (2018).

[6] RL Bulfin, RG Parker, and CM Shetty. 1979. Computational results with a branch-and-bound algorithm for the general knapsack problem. *Naval Research Logistics Quarterly* 26, 1 (1979), 41–46.

[7] Raymond R Burke and Thomas K Srull. 1988. Competitive interference and consumer memory for advertising. *Journal of consumer research* 15, 1 (1988), 55–68.

[8] Arthur E Carter and Cliff T Ragsdale. 2002. Scheduling pre-printed newspaper advertising inserts using genetic algorithms. *Omega* 30, 6 (2002), 415–421.

[9] James C Cox, Vernon L Smith, and James M Walker. 1988. Theory and individual behavior of first-price auctions. *Journal of Risk and uncertainty* 1, 1 (1988), 61–99.

[10] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[11] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. 2007. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American economic review* 97, 1 (2007), 242–259.

[12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* 13, 1 (2012), 2171–2175.

[13] Farhad Ghassemi Tari and Reza Alaei. 2013. Scheduling TV commercials using genetic algorithms. *International Journal of Production Research* 51, 16 (2013), 4921–4929.

[14] Shenshen Gu and Tao Hao. 2018. A pointer network based deep learning algorithm for 0–1 knapsack problem. In *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, 473–477.

[15] Xiaotian Hao, Zhaoqing Peng, Yi Ma, Guan Wang, Junqi Jin, Jianye Hao, Shan Chen, Rongquan Bai, Mingzhou Xie, Miao Xu, et al. 2020. Dynamic Knapsack Optimization Towards Efficient Multi-Channel Sequential Advertising. In *International Conference on Machine Learning*. PMLR, 4060–4070.

[16] Kaiwen Li, Tao Zhang, and Rui Wang. 2020. Deep reinforcement learning for multiobjective optimization. *IEEE transactions on cybernetics* 51, 6 (2020), 3103–3114.

[17] Lu Liu. 2011. Solving 0-1 Knapsack Problems by Greedy Method and Dynamic Programming Method. In *Advanced Materials Research*, Vol. 282. Trans Tech Publ, 570–573.

[18] Silvano Martello, David Pisinger, and Paolo Toth. 1999. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management science* 45, 3 (1999), 414–424.

[19] Silvano Martello and Paolo Toth. 1987. Algorithms for knapsack problems. *North-Holland Mathematics Studies* 132 (1987), 213–257.

[20] Yamato Mizobe, Kei Ohnishi, and Akihiro Fujiwara. 2019. Strawberry optimization for multi-objective knapsack problem. In *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE, 481–483.

[21] Yuhui Shi and Russell Eberhart. 2001. Particle swarm optimization: developments, applications and resources. In *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*, Vol. 1. IEEE, 81–86.

[22] Prabhakant Sinha and Andris A Zoltners. 1979. The multiple-choice knapsack problem. *Operations Research* 27, 3 (1979), 503–515.

[23] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks.

[24] Meihong Wang and Wenhua Zeng. 2010. A comparison of four popular heuristics for task scheduling problem in computational grid. In *2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*. IEEE, 1–4.

[25] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. 2011. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation* 1, 1 (2011), 32–49.