Optimization engine

Data Science group,
Strike Social, Chicago

February 7, 2019

Abstract

In this report, we describe a current version of optimization engine developed within Strike Social company. The engine provides an optimal budget allocation among campaigns as well as optimal bidding at DoubleClick or Adwords auctions right in few hours of a project running. The engine can be used directly by a campaign manager (media buyer) to improve a campaign performance according to chosen campaign goals. It can also be applied within a SaaS platform to support advertising projects online.

1 Introduction

From analytical perspectives, there are two main stages in preparation and running an advertising project, called recommendation and optimization.

First, we have to recommend a set of audience groups (campaigns) which would fit best to the advertiser's objectives, for example, to maximize a total number of views, clicks, subscribers, minimize cost-per-view (CPV) and so on. Advertising project may also have a mixture of objectives. In the latter case, we should maximize a number of effectively "good" views ("engagement rate"). Provided a set of advertiser's criteria for a given campaign (e.g. business vertical, interest and keyword groups, demographics [age, gender, parental status], device [Desktop, Mobile, Tablet], network [In-Stream, In-Display, In-Search], etc), an output of the engine should be a set of campaigns ordered by some total score. This score is calculated using campaign key performance indicators (KPIs), major and minor, as well as similarities (probabilities to match) to the chosen advertiser's criteria. In other words, it should reflect association rules between the advertiser's request and campaigns, and order the campaigns by a strength of their association to this request. These recommendations would be used during a campaign planning process, and, potentially, later during the campaign run to make tunings "on fly".

Second, after the audience groups (campaigns) have been recommended, approved by a user (client), and launched on an advertising platform (e.g. DoubleClick), they have to be supported online to guarantee a maximal performance. By this we mean a delivery of a highest possible project score using a given fixed project budget. The latter can be done by applying standard maximization techniques with linear boundary conditions. These conditions may include, for example, a necessity to satisfy a total project budget, a minimum and maximum budget per each campaign, maximum allowed project CPV, a target view rate (VR) and some others. Additional limitation in the optimization problem can also be a mandatory budget delivery within some factors, for example, creatives, age, gender, device, targeting audience group.

To achieve a maximal flexibility at the optimization stage, the audience groups are split into campaigns at the recommendation stage by making almost all possible combinations of factors (some filtering based on historical data is imposed to prevent unreasonably big combinatorics) like age, gender, device. Such a fine granularity allows to increase the overall project performance by looking at all individual components and optimizing them separately.

Sections below provide a brief description of all main ingredients of the current optimization engine.

2 Data selection

Here we list main data sources that are used as an input for the optimization engine within its research version.

• *Adwords data*
Google Adwords API stores different types of performance metrics on all accounts and campaigns. Currently we mainly use two tables, CAMPAIGN_PERFORMANCE_REPORT and VIDEO_PERFORMANCE_REPORT. Other potentially important tables (AGE_RANGE_PERFORMANCE_REPORT, GENDER_PERFORMANCE_REPORT, and PARENTAL_STATUS_PERFORMANCE_REPORT) are currently used at the recommendation stage only.

• *DBM data*
This data currently include main metrics related with the campaign performance and costs, and are extracted hourly. So far, no device information is available.

• *Utopsa data*
For the purpose of the engine, Utopsa PostgreSQL data base is a source of a project ordered data like a budget, a number of views, network, device, demographics and other similar fields.

3 Scoring

A central part in a campaign optimization is defining their scores according to some criteria. Definition of the score is tightly related with a main project KPI and a type of user interaction (as well as relevant rates of interactions). In a current configuration, the main KPI is CPV. As a secondary KPI, the engine currently supports following interactions: views (and view rate, VR), complete views (and view completion rate, VCR), clicks (and click-through-rate, CTR) and subscriptions. A score of $i-$th campaign can be expressed as

$$Score_i = Views_i \times KPI_i^{w_{vr}} / CPV_i^{2-w_{vr}}. \tag{1}$$

Here KPI is one of the following secondary KPIs: VR, VCR, CTR. Factor w_{vr} is an importance factor of the secondary KPI. Since in most cases it is a view rate, it is often called below as a *VR importance*.

In the engine, the VR importance is determined automatically as a function of the project CPV, see Fig. 1 (It can also be specified manually using the interactive regime, see below.) When the project CPV is much below than a contracted "max CPV", the VR importance is high, and we promote more campaigns with high VR (or any secondary KPI, in general). Otherwise, we prefer more campaigns with lower CPV. Please note the power 2 in $CPV_i^{2-w_{vr}}$. If the project CPV is critically high, then $w_{vr} \to 0$, and we increase importance of low CPV campaigns quadratically.

For projects with rare events such as subscribers ($Subs$) or earned views as an objective, the score is defined as

$$Score_i = Subs_i / CPV_i^{2-w_{sr}}. \tag{2}$$

In this case, we are interested only in the number of subscribers bought at lowest possible CPV. The factor w_{sr} is a subscription rate importance and it plays a similar role to the VR importance shown in Fig. 1.

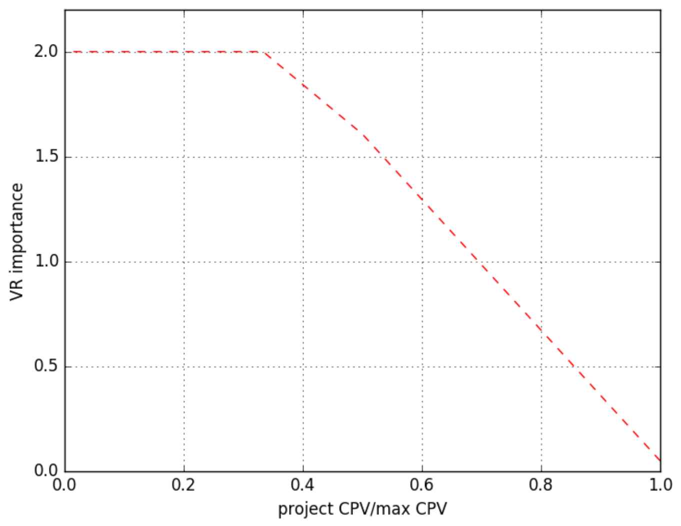


Figure 1: View Rate importance.

4 Optimization of budget allocation

In general, an optimization problem can be presented in this way:

$$\begin{aligned} \text{Maximize } \mathbf{C^T x} \\ \text{subject to}: \mathbf{Ax} \geq B_1 \text{ and } \mathbf{Ax} \leq B_2 \\ (\text{or just } \mathbf{Ax} = B), \\ x_i \geq 0, \\ \sum_i^N x_i = 1. \end{aligned} \tag{3}$$

Here $\mathbf{C}$ is a utility function (e.g. may correspond to the number of views, view rates) which should be maximized, $\mathbf{x}$ is a set of the individual campaign budget fractions, B is a total budget, and $\mathbf{A}$ is a set of N optimal coefficients which maximize the utility function $\mathbf{C}$.

Using definitions from the previous sections, the first two lines of Eq. 4 can be re-written as

$$\begin{aligned} \text{Maximize } \sum_i^{N_c} Score_i \\ \text{subject to}: \sum_i^{N_c} b_i = Budget\_per\_day, \\ Budget\_min(h) \leq b_i \leq Budget\_max(h), \end{aligned} \tag{4}$$

where N_c is a total number of campaigns, $Budget\_per\_day$ is the project budget that has to be delivered 'today', $Budget\_min(h)$, $Budget\_max(h)$ are min/max budgets that can be delivered by $i-$th campaign for a given hour h. In the current default implementation, we take

$$\begin{aligned} Budget\_min(h) = 0.25 Budget\_yest(h), \\ Budget\_max(h) = f_{corr} Budget\_hist(h) \end{aligned} \tag{5}$$

where $Budget\_yest(h)$ and $Budget\_hist(h)$ are the budgets delivered 'yesterday' and historical weighted average budget (see Eq. 6), respectively, delivered from hour h till end of the day, $[h, \text{EOD}]$. Factor f_{corr} is a correction factor showing by how much we can increase the historical average budget for a given hour. It is shown on Fig. 2 versus number of project days passed.

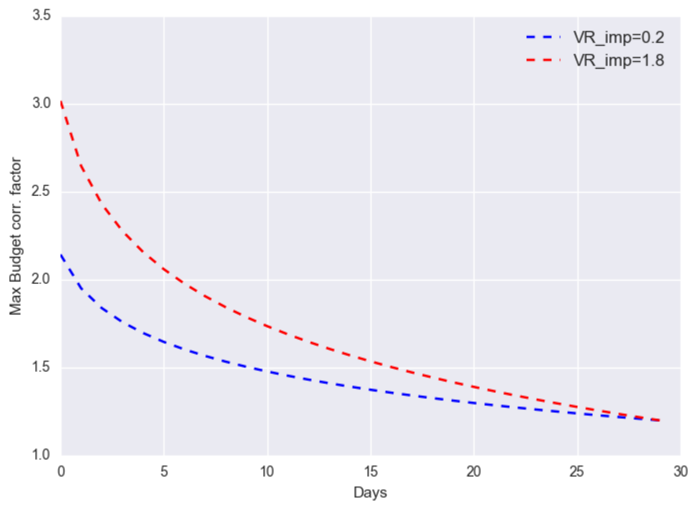


Figure 2: Budget correction factor vs days.

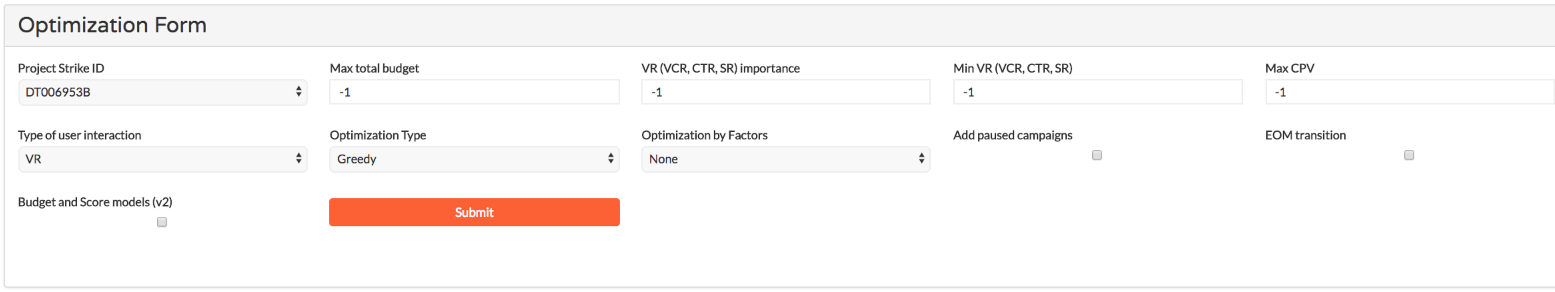
From Fig. 2 one can see, that we can be more risky in the beginning of the project and vary budget more. It also depends on the VR importance (= project CPV, see Fig. 1). The lower project CPV with respect to "max CPV", the more budget can be increased: in case of a lower budget delivery we have enough room to increase CPV later and improve the delivery rate.

Please see Section 9 where we describe modifications for $Budget\_max(h)$ implemented in a new version (v2) of the optimization engine.

The optimization goal in Eqs. 4 – 6 is to find a set of b_i that maximizes the sum of scores ("effective views"). All the components of the score defined in Eq. 1 depend on budget. So, in general, the optimization problem is non-linear, since the score is a non-linear function of the budget (see Section 8). Programmatically, the optimization is done using a scientific minimization package [1].

5 Input form and output tables.

In the engine's user interface (see Fig. 3), we allow a user to provide a budget for today, bottom limits on VR (VCR, CTR, SR) and the number of views, and upper limits on CPV of individual campaigns. In such a way we can directly cut off campaigns with a low performance in terms of those metrics and invest only into the remaining campaigns.



Optimization Form

Project Strike ID	Max total budget	VR (VCR, CTR, SR) importance	Min VR (VCR, CTR, SR)	Max CPV
DT006953B	-1	-1	-1	-1
Type of user interaction	Optimization Type	Optimization by Factors	Add paused campaigns	EOM transition
VR	Greedy	None	☐	☐
Budget and Score models (v2)	Submit			
☐				

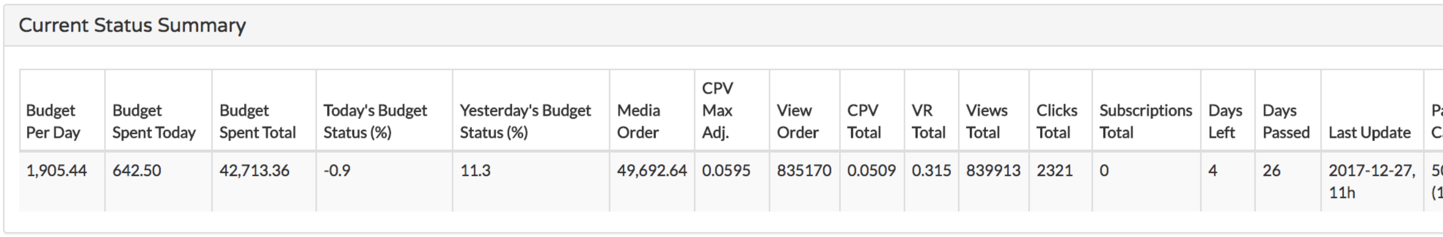
Figure 3: Interactive input form of optimization engine.

Output from the optimization engine run has three main tables: (1) "Current Status Summary" that shows main metrics related with a given project by this hour (see Fig. 4); (2) "Optimization Result", with optimized project budget, number of views, etc that are expected to achieve if we follow the optimized budget distribution, shown in the 3rd table (3) "Optimized Budget (and average daily metrics)". This table shows actual budgets and maximum CPV per each line item (campaign). Table in Fig. 6 also shows weighted average metrics obtained by averaging over last N=14 days.

$$\langle V \rangle = \frac{\sum_{d=0}^{N} V_i \gamma^d}{\sum_{d=0}^{N} \gamma^d}, \tag{6}$$

where V is any metrics (e.g. cost, views), and d stands for a difference in the number of days from today ($d = 0 - N$). Factor γ is slowly decreasing from 0.75 in the beginning of the flight to 0.5 in the end of flight. That is, by the end of flight, we pay more attention (weight more) to most recent days.

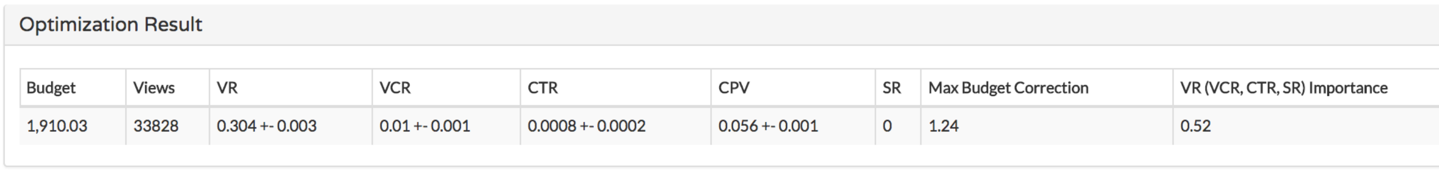
If there are campaigns to pause or to enable, they are placed in two separate tables, see Fig. 7 as an example. Table 'Campaigns to enable' looks very similar.



Current Status Summary

Budget Per Day	Budget Spent Today	Budget Spent Total	Today's Budget Status (%)	Yesterday's Budget Status (%)	Media Order	CPV Max Adj.	View Order	CPV Total	VR Total	Views Total	Clicks Total	Subscriptions Total	Days Left	Days Passed	Last Update	Pa C
1,905.44	642.50	42,713.36	-0.9	11.3	49,692.64	0.0595	835170	0.0509	0.315	839913	2321	0	4	26	2017-12-27, 11h	50 (1

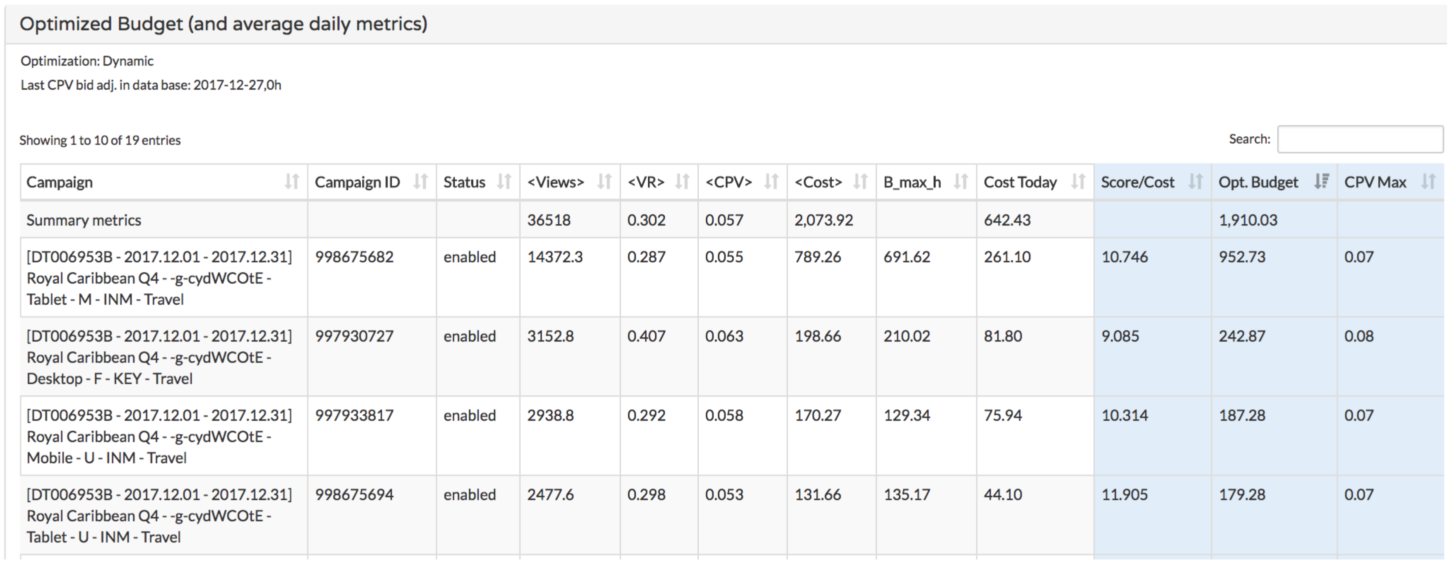
Figure 4: Optimization engine output: Current Status Summary.



Optimization Result

Budget	Views	VR	VCR	CTR	CPV	SR	Max Budget Correction	VR (VCR, CTR, SR) Importance
1,910.03	33828	0.304 +- 0.003	0.01 +- 0.001	0.0008 +- 0.0002	0.056 +- 0.001	0	1.24	0.52

Figure 5: Optimization engine output: Optimization Result.



Optimized Budget (and average daily metrics)

Optimization: Dynamic

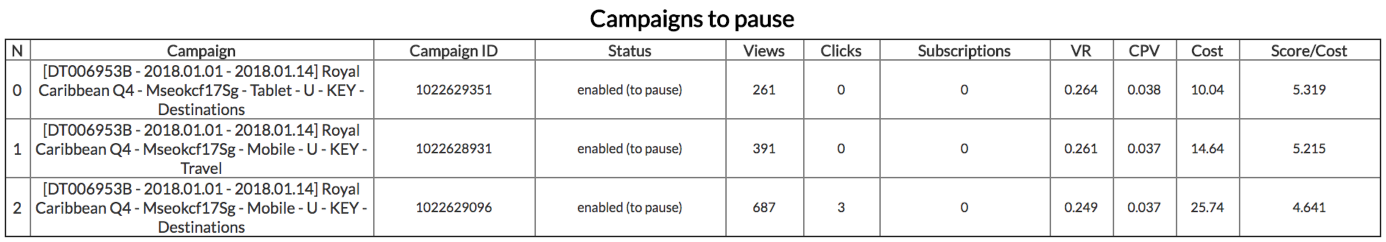
Last CPV bid adj. in data base: 2017-12-27,0h

Showing 1 to 10 of 19 entries

Search:

Campaign	Campaign ID	Status	<Views>	<VR>	<CPV>	<Cost>	B_max_h	Cost Today	Score/Cost	Opt. Budget	CPV Max
Summary metrics			36518	0.302	0.057	2,073.92		642.43		1,910.03	
[DT006953B - 2017.12.01 - 2017.12.31] Royal Caribbean Q4 - -g-cydWCOtE - Tablet - M - INM - Travel	998675682	enabled	14372.3	0.287	0.055	789.26	691.62	261.10	10.746	952.73	0.07
[DT006953B - 2017.12.01 - 2017.12.31] Royal Caribbean Q4 - -g-cydWCOtE - Desktop - F - KEY - Travel	997930727	enabled	3152.8	0.407	0.063	198.66	210.02	81.80	9.085	242.87	0.08
[DT006953B - 2017.12.01 - 2017.12.31] Royal Caribbean Q4 - -g-cydWCOtE - Mobile - U - INM - Travel	997933817	enabled	2938.8	0.292	0.058	170.27	129.34	75.94	10.314	187.28	0.07
[DT006953B - 2017.12.01 - 2017.12.31] Royal Caribbean Q4 - -g-cydWCOtE - Tablet - U - INM - Travel	998675694	enabled	2477.6	0.298	0.053	131.66	135.17	44.10	11.905	179.28	0.07

Figure 6: Optimization engine output: Optimized Budget (and average daily metrics).



Campaigns to pause

N	Campaign	Campaign ID	Status	Views	Clicks	Subscriptions	VR	CPV	Cost	Score/Cost
0	[DT006953B - 2018.01.01 - 2018.01.14] Royal Caribbean Q4 - Mseokcf17Sg - Tablet - U - KEY - Destinations	1022629351	enabled (to pause)	261	0	0	0.264	0.038	10.04	5.319
1	[DT006953B - 2018.01.01 - 2018.01.14] Royal Caribbean Q4 - Mseokcf17Sg - Mobile - U - KEY - Travel	1022628931	enabled (to pause)	391	0	0	0.261	0.037	14.64	5.215
2	[DT006953B - 2018.01.01 - 2018.01.14] Royal Caribbean Q4 - Mseokcf17Sg - Mobile - U - KEY - Destinations	1022629096	enabled (to pause)	687	3	0	0.249	0.037	25.74	4.641

Figure 7: Optimization engine output: Campaigns to pause. Table 'Campaigns to enable' looks very similar.

6 Some important options.

Below we explain some other input parameters used in the input form of the optimization engine.

- "Type of user interaction": VR, VCR, CTR, SR, please see the text of previous section.
- "Optimization Type". It can take two values: Greedy, Optimal for Today. The difference between these two regimes is following. With Optimal for Today, we make Score and CPV fits (see Section 10) using historical data for only remaining part of the day. For example, if we run the engine at 12pm, we consider historical data (last 4 days) for the period [12pm, end of day]. That is we try to find most optimal fits for the rest of the day. CPVs for four day segments (0-6, 6-12, 12-18, 18-24) are shown in Fig. 8. They may have different structure/laws, and not necessarily lay on a same CPV vs budget line. Weekend points are shown by a twice bigger markers.

  With Greedy option, we take data for all hours. In such a way, the fitted Score and CPV values are valid on average for a whole day. Also, "greedy" CPV are typically higher since we are looking for fair CPV price valid for all day times. For active media buyers we recommend Optimal for Today. The SaaS platform uses Greedy as a default since and we are not sure when a client will run the engine next time again, and where a stability with a budget delivery has a higher priority.
- "Optimization by Factors". It can take two values: None, All. This option can be used if we want to find most optimal budget distribution by some factors. For example, we have three creatives (videos), and a client requested 33.3%/33.3%/33.3% budget split by those creatives. In this case, the engine

makes a loop over all possible factors (see method main_optimization() in Fig. 24), two in this case, and make two separate optimizations with 33.3% of the today's daily budget per each factor (video). In this scenario, we follow a client's wish and do not make any optimization over factors, i.e. "Optimization by Factors" is None. Table Fig. 9 shows optimized daily budget by factors with this option. However, if "Optimization by Factors" is All, we ignore the initially requested mandatory budget split, and make our own optimization over all the factors. It may happen that the engine finds a better budget split from the point of view of the project goals. Such results can be reported to the client. See Table Fig. 10 as an example using same project as used in Table Fig. 9. In general, there can be a combination of a few factors, e.g. video and age. The optimization engine is agnostic with respect to specific factor names.

The campaign assignment and then budget assignment procedure by factors are implemented in methods make_campaigns() and get_kevin_factors(). None is a default regime.

- "Add paused campaigns". In the research version of the engine, a user may add the earlier paused campaigns to participate in the optimization. Then they may receive some budget, *provided that their scores are high enough* compared to other campaigns. In the SaaS platform, all campaigns are considered as enabled/active before an optimization run, and paused by the engine if the rules listed in Section 7 are satisfied.
- "EOM transition". This option is activated in the End Of Month, when we want to generate a budget for the next month provided that we will have same campaigns as in the current month. It allows to start a next month in most optimal way, using a wisdom from a previous month.
- "Budget and Score models (V2)." By allowing this option, user runs the engine with two advanced models, budget delivery model and new score model. These two models are described in Sections 9 and 12.

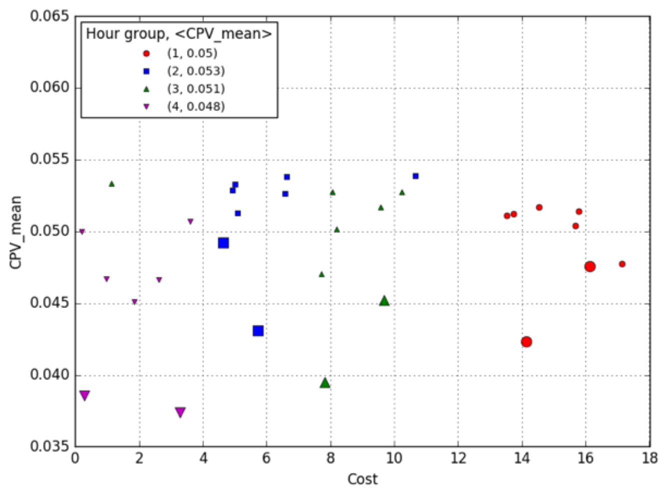
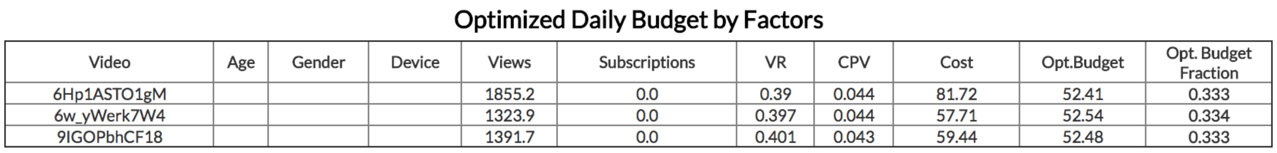


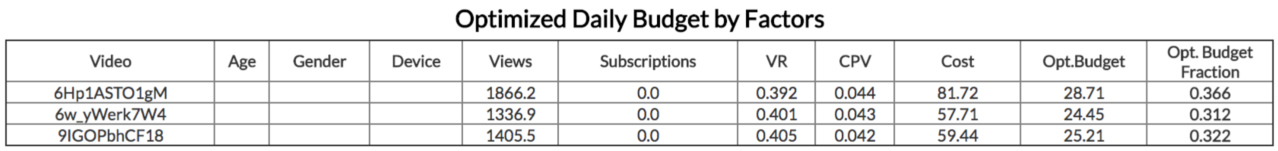
Figure 8: CPVs for four hour groups (0-6, 6-12, 12-18, 18-24). They may have different structure/laws, and not necessarily lay on a same CPV vs budget line. Weekend points are shown by a twice bigger markers.



Optimized Daily Budget by Factors

Video	Age	Gender	Device	Views	Subscriptions	VR	CPV	Cost	Opt.Budget	Opt. Budget Fraction
6Hp1ASTO1gM				1855.2	0.0	0.39	0.044	81.72	52.41	0.333
6w_yWerk7W4				1323.9	0.0	0.397	0.044	57.71	52.54	0.334
9IGOPbhCF18				1391.7	0.0	0.401	0.043	59.44	52.48	0.333

Figure 9: Optimization engine output: Optimized daily budget by factors with option 'Optimization by Factors' is None.



Optimized Daily Budget by Factors

Video	Age	Gender	Device	Views	Subscriptions	VR	CPV	Cost	Opt.Budget	Opt. Budget Fraction
6Hp1ASTO1gM				1866.2	0.0	0.392	0.044	81.72	28.71	0.366
6w_yWerk7W4				1336.9	0.0	0.401	0.043	57.71	24.45	0.312
9IGOPbhCF18				1405.5	0.0	0.405	0.042	59.44	25.21	0.322

Figure 10: Optimization engine output: Optimized daily budget by factors with option 'Optimization by Factors' is All factors.

7 Campaign pausing procedure.

A campaign gets paused if the following rules are satisfied:

- Pausing is allowed. It happens if
  days_passed $\geq$ min_days and days_left>0 and ($B^{today}_{frac} > 0.1$ or $\langle Views^{today}_{camp}\rangle > 15$).
  Here min_days = max(1,0.1*(#flight days)), B^{today}_{frac} is a budget fraction spent today, and $\langle Views^{today}_{camp}\rangle$ is average number of views per campaign we have already had today. In case of rare events, such as conversions or earned views, we require at least 7 days before we consider campaigns for pausing.
- Either Score/cost, KPI (VR, VCR, CTR), or CPV are below or above some critical value. Those critical values are determined as outliers from a distribution over corresponding variable. For example, see Figs. 11 – 13. One note: we look for outliers in CPV only we the project CPV is $> 0.98 CPV_{max}$ where CPV_{max} is a contracted max CPV..
- Optimized budget is below average project CPV.

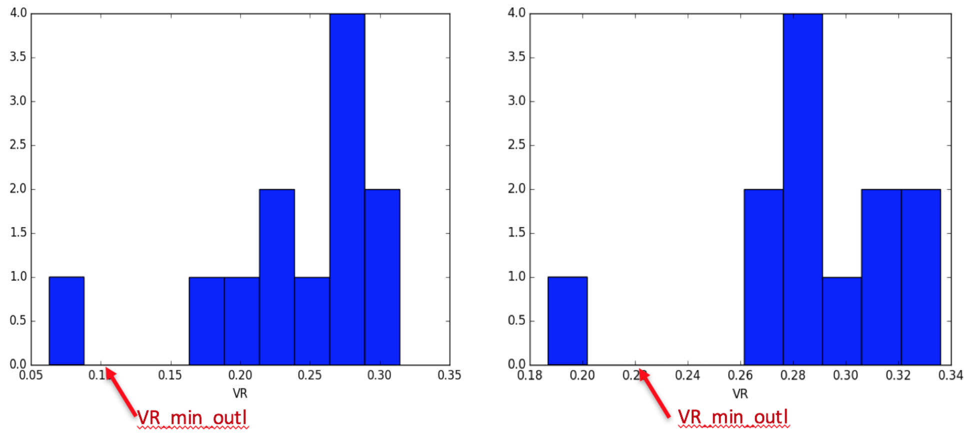


Figure 11: VR distribution of campaigns in one project. Location of the lower cut is shown.

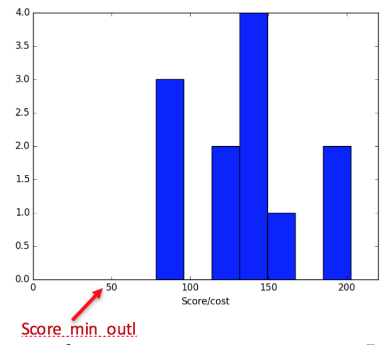


Figure 12: Score/cost distribution of campaigns in one project. Location of the lower cut is shown.

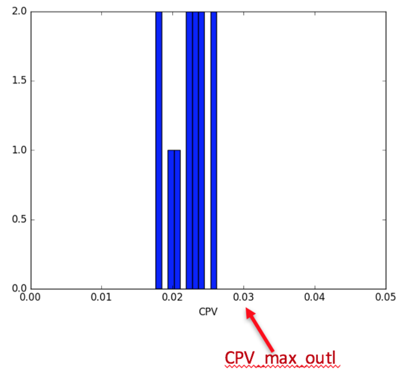


Figure 13: CPV distribution of campaigns in one project. Location of the upper cut is shown.

8 Budget allocations.

There are two options for the budget allocation, Static and Dynamic.

8.1 Static procedure.

We start with Static as a simple one. In case of Static optimization, we distribute budget in a loop over campaigns (line items), starting from a campaign with highest $\langle Score_i \rangle / \langle cost \rangle_i$ ratio. We allocate a max possible budget that a campaign may deliver. We check a feasibility to deliver a certain budget by analysing historical values, and make prediction for such a max value, see Section 9.

8.2 Dynamic procedure.

Scores for each i-th campaign are calculated as a weighted average metrics over last 4 days (see Eq. 6). That is, we are assigning scores and then distribute budget using information on the most recent campaign activities.

However, the scores may change. As a result, our today's budget allocation may be not optimal. To make it optimal, we have to analyze trends of scores vs budget, and express $Score_i$ in Eq. 1 as a function of budget B:

$$Score_i(B_{ij}) = \alpha_i B_{ij}^{\beta_i}, \tag{7}$$

where α_i and β_i are free parameters which have to be found for each i-th campaign using campaigns daily data for day j, and by fitting scores $Score_i(B_{ij})$ to this data. Figure 14 shows an example of such fits for some campaigns.

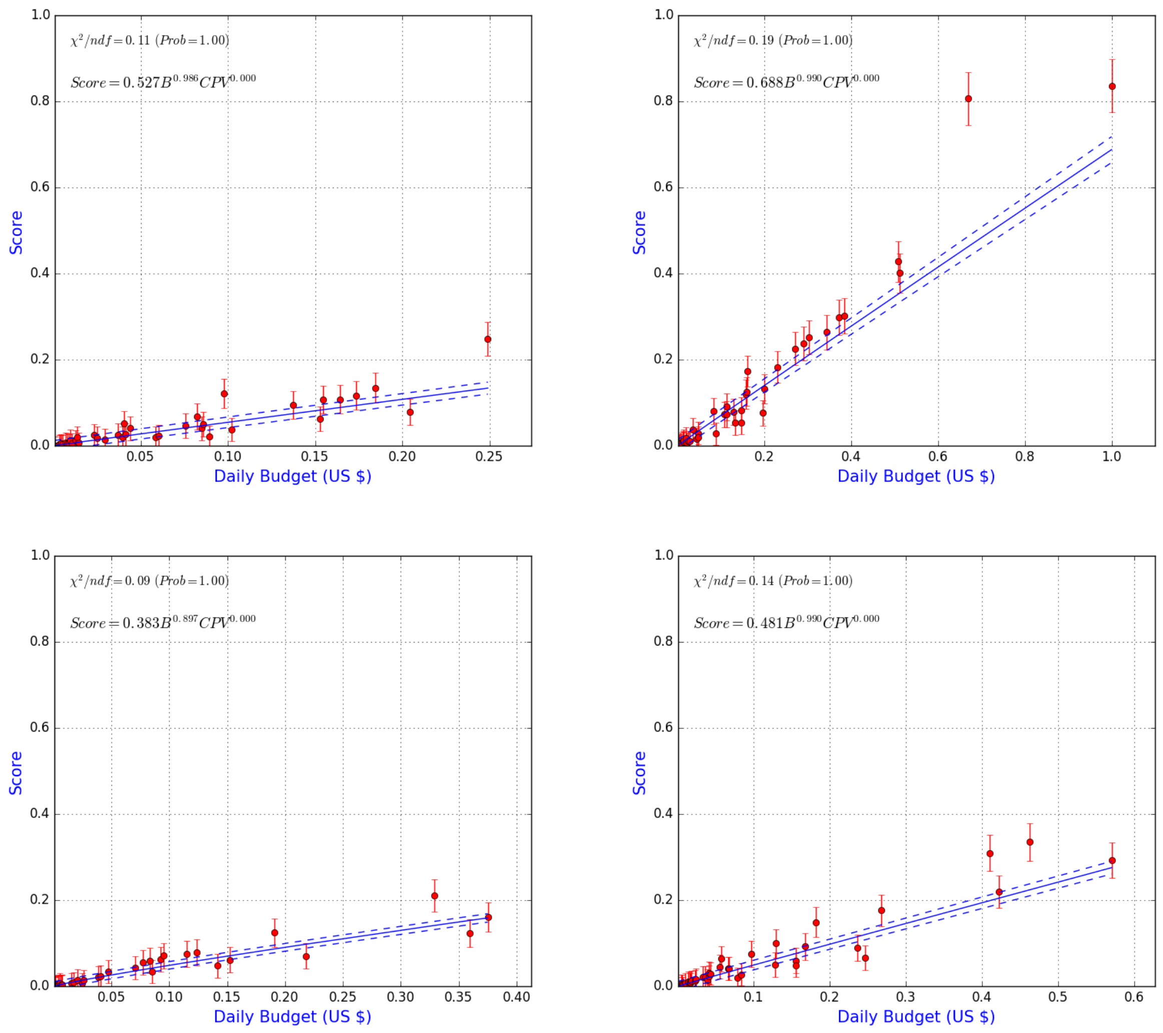


Figure 14: Score vs daily budget for four campaigns ids. Solid line shows main fit and dashed lines show $1-\sigma$ corridor of uncertainties.

Let's look at an example where we were asked to distribute a daily budget of $300 among most effective campaigns. Result is shown in Table 1.

In Table 1, columns 'cost' and 'Budget' show latest daily budget and a suggested optimized budget, respectively. $EffViews$ are Scores expected at the optimized budget value.

Score in Eq. 1 can be thought as a number of "effective views" (where the efficiency is represented by factor $VR_i^{w_{vr}}Prob_i^{w_{win}}$). In the optimization procedure, we maximize the total number of "effective views" by sliding back and forth along the fitted lines for all campaigns, and taking into account their slopes and absolute values. Most effective campaigns are those which scores would grow to the highest values and would give maximum contribution to the total sum per dollar spent. Mathematically, it happens in the point where a function derivative is highest. So, we have to meet a few following conditions: we should take a budget at a maximally possible derivative of the $Score_i(B)$ function, and the budget sum should be equal to the required maximum daily budget.

Priority of campaigns is ordered by their derivatives. It is important to underline that the order of the functional derivatives for campaigns at the averaged (over last 4 days) cost (see column 'cost'), which is

Table 1: Optimized campaigns using the dynamic budget allocation.

	campaign_id	impressions	views	cost	CPV	VR	VCR	Deriv0	Deriv1	EffViews	Budget
1	640156311	3264	782	26.20	0.034	0.240	0.242	17.24	17.49	1124	48.21
2	640156314	2596	653	27.04	0.041	0.252	0.256	12.54	11.95	871.3	49.76
3	640156284	254	73	3.05	0.042	0.287	0.285	9.16	10.74	66.5	5.61
4	640156317	2468	612	23.68	0.039	0.248	0.254	11.65	10.65	728.7	43.57
5	640156290	3044	812	36.71	0.045	0.267	0.275	11.16	10.20	1081	67.55
6	640156287	3269	800	36.78	0.046	0.245	0.254	10.70	9.60	1052.2	67.68
7	640156278	47	18	0.73	0.041	0.383	0.383	6.94	8.55	12	1.35
8	640156440	84	20	0.76	0.038	0.238	0.253	6.77	7.63	12.4	1.40
9	640156275	491	103	4.31	0.042	0.210	0.212	7.13	6.85	78.7	7.93
10	640156299	389	110	5.36	0.049	0.283	0.278	6.71	6.33	64.2	6.94

shown in column 'Deriv0', and the functional derivatives at the optimal budget point, shown in column 'Deriv1', is different. From Table 1 we see that the order of campaigns, beign ordered by Deriv0 or Deriv1 (current default) does differ. For example, campaign_ids 640156284 and 640156275 are out of order in terms at the optimal budget points. In the first case, $Deriv0 = \langle Score \rangle / \langle cost \rangle$, and does not depend on budget, i.e. it is a static variable ($\beta_i = 1$ in Eq. 7). While in the second case, $Deriv1 = \partial Score / \, \partial B = \alpha \beta B^{\beta - 1}$, and does depend on the budget if $\beta \neq 1$.

In Fig. 2 we showed the factor by which we allowed for the daily budget to grow. The main reason why we don't allow to grow the budget even more is related with an increasing size of uncertainties for higher budgets (see Figure 14). Instead, if needed, we can increase a maximum budget for a given advanced campaign using day-by-day increase, and achieve a factor 8–15 in 4-5 days (if we assume that for some first days, its average value is 1.7, then $1.7^4 = 8.4$ and $1.5^5 = 14.2$).

9 Maximum daily campaign budget.

When we plan a budget increase, it is important to know what is daily *allowed* maximal budget (B_{max}) is. It consists of the two parts, the budget that is already delivered, B_{now} (also called B_{min} in the code), and the budget that we can potentially deliver from a given hour till the end of the day, B_{max-h}:

$$B_{max} = B_{now} + B_{max-h} \tag{8}$$

Note that all of the three variables depend on hour. B_{now} is always known, of course, from DBM (Adwords) report. To determine B_{max-h}, we utilize two models. In the first (current default) model, we use either yesterday's budget that we delivered during [this hour, end of day], or weighted average value during last 4 days. Let's call it $cost_{prev-h}$. This value gets multiplied by f_{corr} factor (explained in Section 2) to get B_{max-h}:

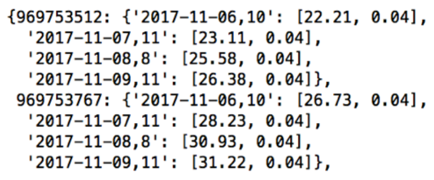
$$B_{max-h} = f_{corr} \; cost_{prev-h}. \tag{9}$$

During first project days we learn what budget we can deliver. If 'today' we will not deliver B_{max-h}, 'tomorrow' we will be scaling from a new actually delivered value, $cost_{prev-h}$.

A disadvantage of this approach is that we have to scale by f_{corr} irregardless of a budget delivery rate (DR)/delivery probability, for a certain budget B_{max-h}. Another approach allows us to learn this rate versus budget, DR(B). For this purpose, we store information on the previous actions, {day,h: [B_max, CPV_max], ...}, for each campaign, and later fit it, see Fig. 15. In such a way, we would be able to predict what budget we can deliver for a given hour as a function of the hour and B_max values. For example, from Fig. 16 one can see that for 12pm, the budget that we can deliver with DR = 0.95 is \$7.73. We can probably deliver more, but chances are less.

In the optimization engine code, the actions are retrieved and assigned to the data frame with hourly data in read_actions.py.

Such budget prediction model is expected to be trained on the passed days every night, between 1-2 am, and then stored in a data base. Method get_budget_and_score_models_v2() shows how to deal with the new budget (and score) model. Then during a day, the model will be retrieved every hour to make a prediction for B_{max_h} (see get_max_budget()). Model is trained using train_models.py and budget_delivery_model.py.



```
{969753512: {'2017-11-06,10': [22.21, 0.04],
  '2017-11-07,11': [23.11, 0.04],
  '2017-11-08,8': [25.58, 0.04],
  '2017-11-09,11': [26.38, 0.04]},
 969753767: {'2017-11-06,10': [26.73, 0.04],
  '2017-11-07,11': [28.23, 0.04],
  '2017-11-08,8': [30.93, 0.04],
  '2017-11-09,11': [31.22, 0.04]},
```

Figure 15: Dictionary of previous actions stored in a data base in the format {campaign_id1: {day,h: [B_max, CPV_max], ...}, campaign_id2: {..}, ... }.

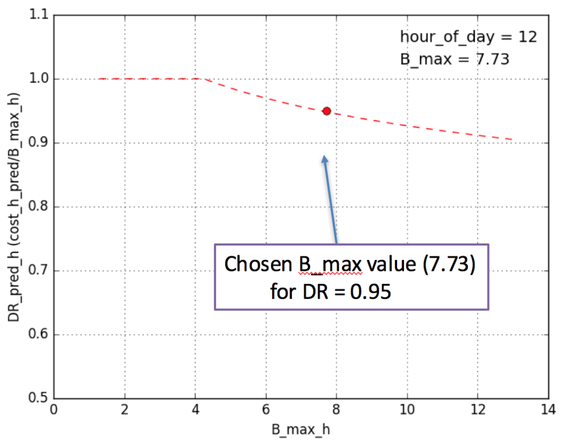


Figure 16: Budget delivery rate vs budget, calculated for each hour.

9.1 Promotional factors

We also make f_{corr} factor dependent on a campaign performance by implementing so-called *promotional* budget scaling factors. These factors are proportional to $log([Score_i/cost_i]/Norm)$, where the normalization factor $Norm$ is 80% quantile of Score-to-cost ratios for all campaign of a given flight. At allows us to assign a higher upper budget limits for a well performing campaigns.

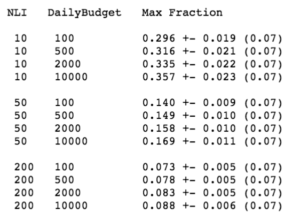
9.2 Absolute max budget per line item.

It is important to impose an upper limit on maximum campaign budget from a different perspective. Advertisers would prefer to keep most of the initial audience groups running, and we should not allow to concentrate all the flight budget within 1-2 campaigns. This budget fraction should depend on total daily budget and a total number of line items.

To make such study, we took last 3 month of data which It include 132 unique insertion ids. Only data with total daily $cost > 100$ and passed days > 5 were considered. We found that such maximum fraction can be parametrized as

$$fitted\_max\_fraction = aB_{daily}^{b}N_{LI}^{c} \tag{10}$$

where $a, b, c = 0.719, 0.041, -0.467$ (R^2 quality of the fit is 0.57). Final max fraction is never higher than 2*fitted_max_fraction.



```
NLI   DailyBudget   Max Fraction

 10    100          0.296 +- 0.019 (0.07)
 10    500          0.316 +- 0.021 (0.07)
 10    2000         0.335 +- 0.022 (0.07)
 10    10000        0.357 +- 0.023 (0.07)

 50    100          0.140 +- 0.009 (0.07)
 50    500          0.149 +- 0.010 (0.07)
 50    2000         0.158 +- 0.010 (0.07)
 50    10000        0.169 +- 0.011 (0.07)

 200   100          0.073 +- 0.005 (0.07)
 200   500          0.078 +- 0.005 (0.07)
 200   2000         0.083 +- 0.005 (0.07)
 200   10000        0.088 +- 0.006 (0.07)
```

Figure 17: Absolute max budget per line item.

10 CPV bidding.

In the current implementation, after we assigned to each campaign its own maximal budget, we estimate what CPV is needed to deliver this budget. It is done in method get_CPV_bid(), see Fig. 24. With Greedy option (see Section 6), we split each day into a few day parts (see method get_nhour_groups()), and then find CPV max values for each day part for 0-24 hours (In case of Optimal for Today, we do it only for the period [this hour, end of day].) We use those values to fit a dependence of CPV_{max} versus $costs$, please see Fig. 18. In such a way, we determine average CPV_{max} that is needed to deliver a given budget.

Along with a central fit, we also estimate $\pm\sigma$ corridor of uncertainties. These uncertainties are needed to regulate a choice of CPV_{max} value versus project budget delivery rate (DR). A relevant plot is shown in Fig. 19. Linear dependence is a default one. Specifically, if we deliver the project budget, we add just 1σ (since we determined only *average* CPV_{max} in the fit). Otherwise, we add more if DR<1 or less if DR>1.

If option Optimal for Today is chosen, we also correct for the day of week and time of day dependence (we cannot do it with "averaged bidding" while we use Greedy option).

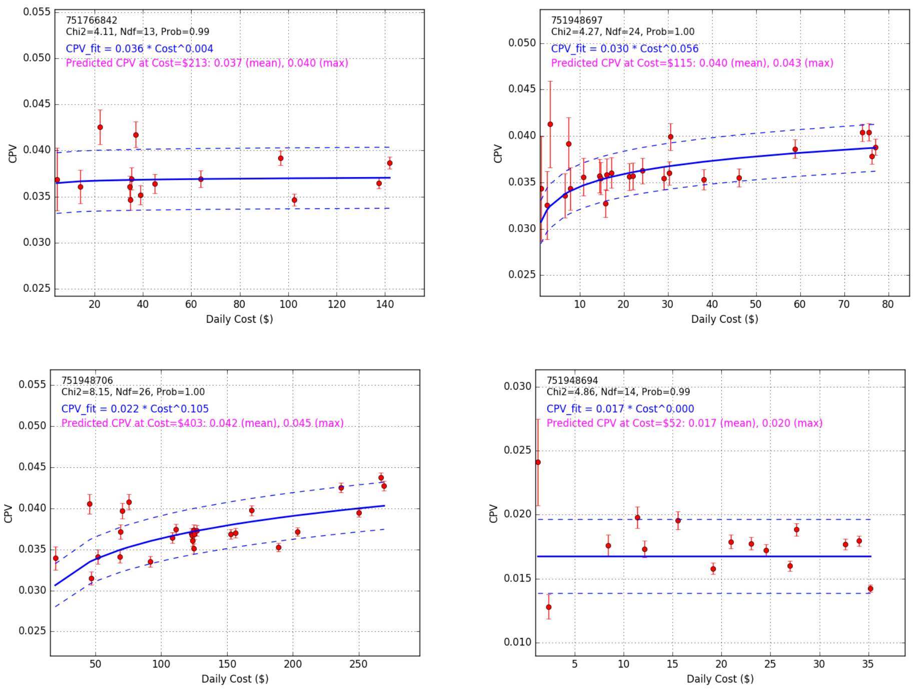


Figure 18: CPV vs daily budget for four campaigns. Solid line shows main fit and dashed lines show $\pm\sigma$ corridor of uncertainties.

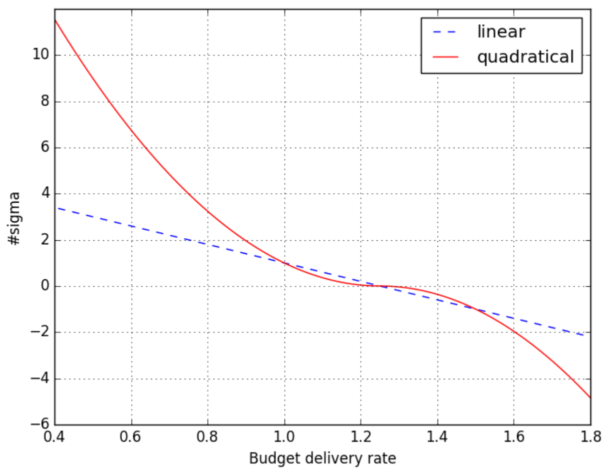


Figure 19: CPV vs budget delivery rate at a project level. Linear dependence is a default one.

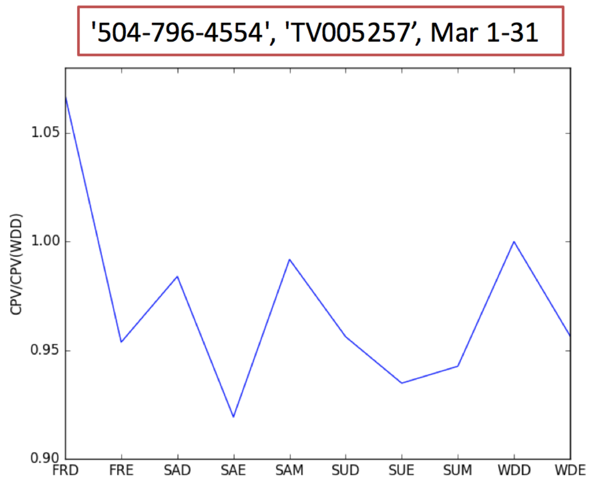


Figure 20: Bid modifications, calculated inside of the project, that reflect time-of-week dependence.

11 Probabilistic scores.

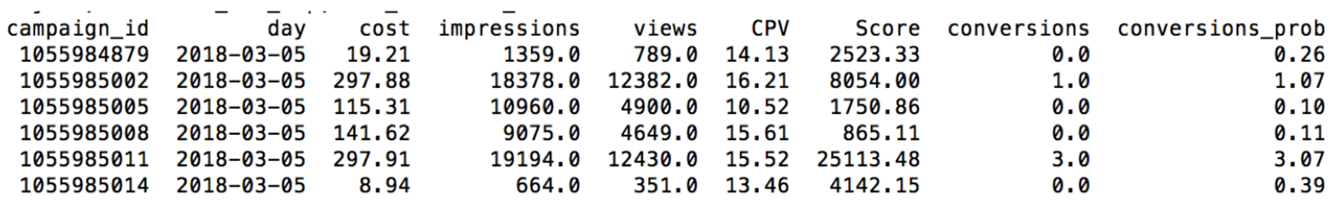
When a flight starts, we should follow some conservative approach in the scoring. There is some 'warm-up' period in the beginning, and we should follow a probabilistic approach. It is better to demonstrate it use rare events such as conversions. Instead of saying 'we see zero conversions', we should rather estimate upper bound on how many conversions we should see provided a given amount of views. That is we have to take into account that zero conversions with 10 views and with 10,000 views are different 'zeros'! Specifically, for scoring we use

$$Conv_{prob,i} = Conv_i + \varepsilon_i \tag{11}$$

where

$$\varepsilon_i = (C\ log(Views_{tot}/Views_i))^{1/2} \tag{12}$$

Here ε_i is a regularization term that takes into account total amount of views in a flight for a given moment $Views_{tot}$, and a desired confidence factor C (typically varies within $1-3$), and $Conv_{prob,i}$ is so-called upper Chernoff bound. For more details on mathematical foundations please see [5]. This regularization is significant at small numbers of views and disappears at large numbers of views. Example of the scores with upper bounds can be seen in Table 21.



```
campaign_id         day    cost  impressions    views    CPV     Score  conversions  conversions_prob
 1055984879  2018-03-05   19.21       1359.0    789.0  14.13   2523.33          0.0              0.26
 1055985002  2018-03-05  297.88      18378.0  12382.0  16.21   8054.00          1.0              1.07
 1055985005  2018-03-05  115.31      10960.0   4900.0  10.52   1750.86          0.0              0.10
 1055985008  2018-03-05  141.62       9075.0   4649.0  15.61    865.11          0.0              0.11
 1055985011  2018-03-05  297.91      19194.0  12430.0  15.52  25113.48          3.0              3.07
 1055985014  2018-03-05    8.94        664.0    351.0  13.46   4142.15          0.0              0.39
```

Figure 21: Actual and probabilistic conversions.

We see that conversions_prob are getting lower with higher views. That is we are getting more confident in the observed 0 conversions and can impose a tighter upper bound. We use conversions_prob for our final scoring.

We follow very similar approach for earned view and clicks. For regular views (complete views), instead of views in 12, we use impressions:

$$\varepsilon_i = (C\ log(Impr_{tot}/Impr_i))^{1/2}, \tag{13}$$

and all the other principles remain the same.

12 New scoring model.

In the currently default scoring model, we use day parts to predict what average score we should expect for a given campaign versus budget. However, a more appropriate question to ask is what score can be *delivered by end of day* for a given {hour, B_max (, CPV_max)}. That is (a) we are interested in a final (EOD) score, and (b) versus state (time) and actions (B_max (, CPV_max)).

So far, the score model is parametrized only versus hour, B_max. We have not included CPV_max yet since this variable itself is a function of the budget, in general (However, this opportunity must be better tested.). We should consider adding day of the week dependence as well.

Similarly to the budget model (see Section 9), for each hour the total cumulative score is a sum of score 'by now" and the expected score to get by end of day:

$$Score_{tot} = Score_{now} + Score_{[h,EOD]} \tag{14}$$

In its turn, $Score_{[h,EOD]}$ can be obtained from

$$Score_{[h,EOD]} = aB^{b}_{max\_h} h^{c}_{left}, \tag{15}$$

where $B_{max\_h} = B_{max} - cum\_cost$ (B_{max} is current maximum budget and cum_cost is a total cost accumulated so far), h_{left} is number of hours left till end of day, and a, b, c are free parameters, found from the fit to historical data.

Like a budget prediction model, the score model is supposed to be trained nightly on the passed days, and then used during a new day, hour-by-hour. Fig. 22 shows dependence of the remaining score, $Score_{[h,EOD]}$ versus hours of day for Dec 3rd after training on data from Dec 1,2. So far, with this model, after two-day training, we are doing mostly good after 6am, but fluctuations up to 20% may happen for hours 0-6.

Method get_budget_and_score_models_v2() shows how to deal with the new score model. Model is trained using train_models.py and score_model.py.

There is also an option to update the score model every hour (see method update_models_hourly()). It will take a bit longer to run the engine hourly, but we would be able to follow the "today's" trend a bit better. In this approach we follow the temporal difference (TD) method of Reinforcement learning (see [6]). Briefly it can be presented by this equation:

$$Score^{calc}_{[h,EOD]} = Score_{h+1} + Score^{pred}_{[h+1,EOD]} \tag{16}$$

That is, beign at hour $h+1$, we know score $Score_{h+1}$ for last hour, we predict $Score^{pred}_{[h+1,EOD]}$, and calculate $Score^{calc}_{[h,EOD]}$. This calculated value is compared to the predicted score for a previous hour $Score^{pred}_{[h,EOD]}$. Using a difference between them

$$\Delta Score = Score^{calc}_{[h,EOD]} - Score^{pred}_{[h,EOD]}, \tag{17}$$

we update all the coefficients as

$$c_i = c_i + \Delta Score(\partial Score/\partial c_i). \tag{18}$$

To be exact, in our code we update the coefficients using Adam algorithm [7], implemented in score_model.py.

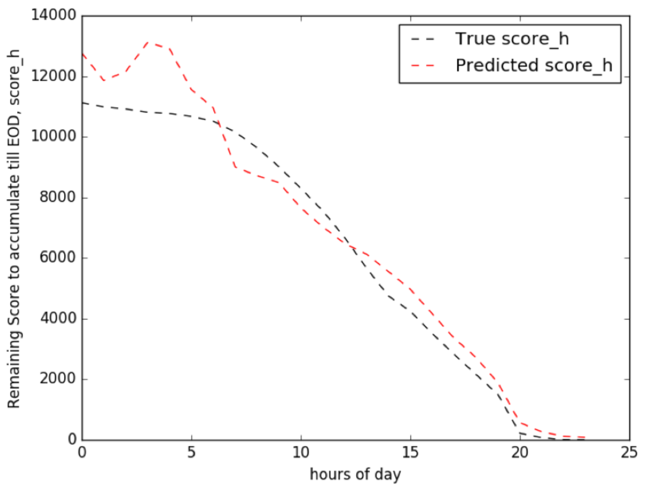


Figure 22: True score vs predicted in the new score model.

13 Discussion

Optimization engine described above currently exists as a stand-alone package and available from Git [2] in two main branches, research mostly for local media buyers, and prototype for the SaaS platform.

Most advanced version (v2) so far only exists in research branch in files read_actions.py, train_models.py, budget_delivery_model.py and score_model.py inside directory model. They are called from optimization_model.py via method get_budget_and_score_models_v2().

Fig. 23 shows a picture of the engine running within the SaaS platform. The function call tree is presented in Figs. 24, 25. Same files are also available from research branch in `https://github.com/strike-social/optimization_engine/tree/research/app` as oe_depth4.png and oe_depth4_v2.png.

Acknowledgement

We are very grateful to a team of media buyers for permanent testing of the optimization engine and providing useful feedback (especially to Jason Nesbitt, Natalie Jazdzyk), Mark Shore and data science team for permanent encouraging discussions, and helpful advices, software engineers (especially Jason Mossberg), and the whole Kevin team for making it running within the SaaS platform.

References

[1] `http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`

[2] `https://github.com/strike-social/optimization_engine`

[3] `http://docs.aws.amazon.com/lambda/latest/dg/welcome.html`

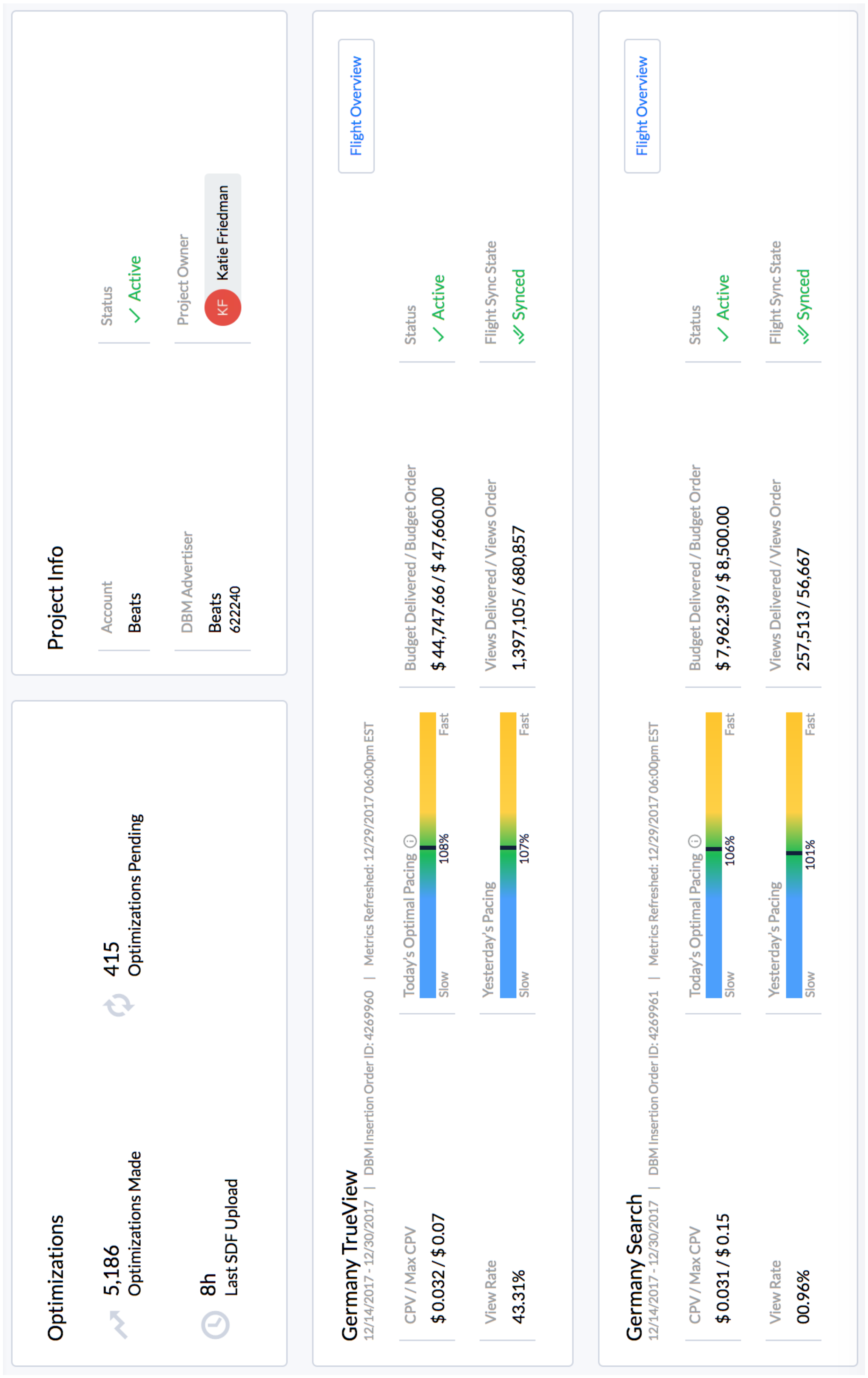
[4] `https://git.dev.striketech.pl/dbandurin/budget_bid_changes/tree/master`

[5] https://en.wikipedia.org/wiki/Chernoff_bound
http://math.mit.edu/ goemans/18310S15/chernoff-notes.pdf
https://courses.cs.washington.edu/courses/cse599i/18wi/resources/lecture3/lecture3.pdf

[6] "Reinforcement learning: An Introduction", R.S. Sutton, A.G. Barto.

[7] https://arxiv.org/pdf/1412.6980.pdf, "Adam: a method for stochastic optimization", D.P. Kingma, J.L. Ba.

Appendix



Optimizations

5,186
Optimizations Made

415
Optimizations Pending

8h
Last SDF Upload

Project Info

Account
Beats

Status
✓ Active

DBM Advertiser
Beats
622240

Project Owner
KF Katie Friedman

Germany TrueView
12/14/2017 - 12/30/2017 | DBM Insertion Order ID: 4269960 | Metrics Refreshed: 12/29/2017 06:00pm EST

Flight Overview

CPV / Max CPV	Today's Optimal Pacing	Budget Delivered / Budget Order	Status
$ 0.032 / $ 0.07	Slow 108% Fast	$ 44,747.66 / $ 47,660.00	✓ Active

View Rate	Yesterday's Pacing	Views Delivered / Views Order	Flight Sync State
43.31%	Slow 107% Fast	1,397,105 / 680,857	✓ Synced

Germany Search
12/14/2017 - 12/30/2017 | DBM Insertion Order ID: 4269961 | Metrics Refreshed: 12/29/2017 06:00pm EST

Flight Overview

CPV / Max CPV	Today's Optimal Pacing	Budget Delivered / Budget Order	Status
$ 0.031 / $ 0.15	Slow 106% Fast	$ 7,962.39 / $ 8,500.00	✓ Active

View Rate	Yesterday's Pacing	Views Delivered / Views Order	Flight Sync State
00.96%	Slow 101% Fast	257,513 / 56,667	✓ Synced

Figure 23: OE in action at SrtikeSocial SaaS platform.

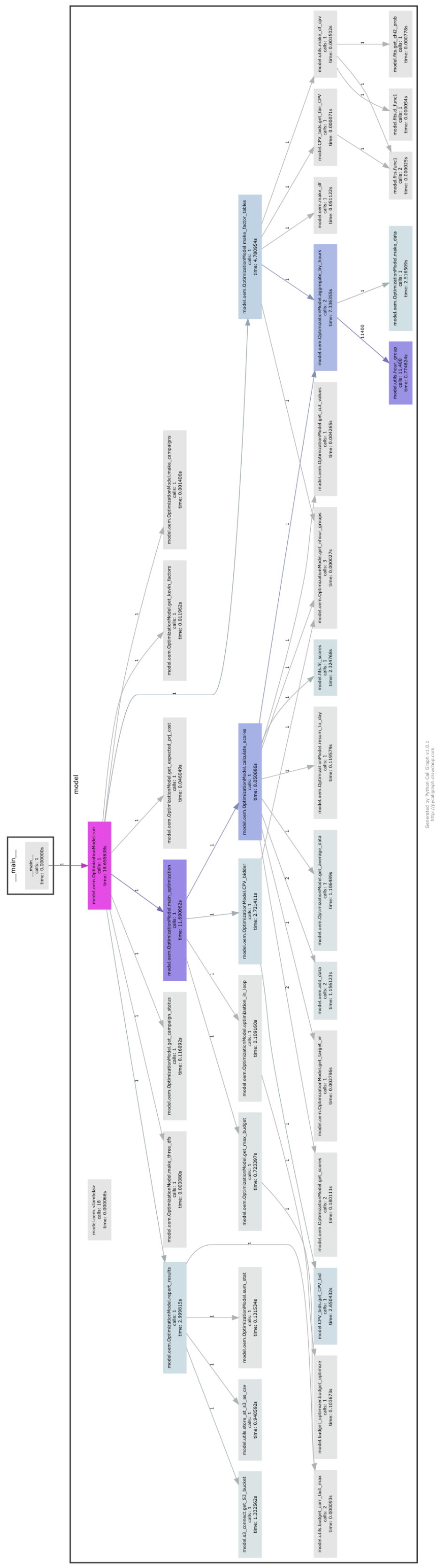

__main__
__main__ calls: 1 time: 0.000000s
model
model.oem.OptimizationModel.run calls: 1 time: 19.655838s
model.oem.<lambda> calls: 18 time: 0.000068s
model.oem.OptimizationModel.make_campaigns calls: 1 time: 0.001406s
model.oem.OptimizationModel.get_kevin_factors calls: 1 time: 0.011962s
model.oem.OptimizationModel.get_expected_prj_cost calls: 1 time: 0.046049s
model.oem.OptimizationModel.main_optimization calls: 1 time: 11.690962s
model.oem.OptimizationModel.get_campaign_status calls: 1 time: 0.116092s
model.oem.OptimizationModel.make_three_dfs calls: 1 time: 0.000080s
model.oem.OptimizationModel.report_results calls: 1 time: 2.999815s
model.oem.OptimizationModel.make_factor_tables calls: 1 time: 4.780954s
model.oem.OptimizationModel.calculate_scores calls: 1 time: 8.050086s
model.oem.OptimizationModel.CPV_bidder calls: 1 time: 2.721411s
model.oem.OptimizationModel.optimization_in_loop calls: 1 time: 0.109160s
model.oem.OptimizationModel.get_max_budget calls: 1 time: 0.723397s
model.oem.OptimizationModel.sum_stat calls: 1 time: 0.131534s
model.utils.store_at_s3_as_csv calls: 1 time: 0.940592s
model.s3_connect.get_S3_bucket calls: 1 time: 1.332562s
model.utils.make_df_cpv calls: 1 time: 0.001502s
model.CPV_bids.get_fair_CPV calls: 1 time: 0.000071s
model.oem.make_df calls: 1 time: 0.051122s
model.oem.OptimizationModel.aggregate_by_hours calls: 2 time: 7.336355s
model.oem.OptimizationModel.get_cut_values calls: 1 time: 0.004265s
model.oem.OptimizationModel.get_nhour_groups calls: 3 time: 0.000027s
model.fits.fit_scores calls: 1 time: 2.324768s
model.oem.OptimizationModel.resum_to_day calls: 1 time: 0.119579s
model.oem.OptimizationModel.get_average_data calls: 1 time: 1.106489s
model.oem.add_data calls: 2 time: 1.156123s
model.oem.OptimizationModel.get_target_vr calls: 1 time: 0.002796s
model.oem.OptimizationModel.get_scores calls: 2 time: 0.180111s
model.CPV_bids.get_CPV_bid calls: 1 time: 2.650432s
model.budget_optimizer.budget_optimize calls: 1 time: 0.103673s
model.utils.budget_corr_fact_max calls: 2 time: 0.000093s
model.fits.get_chi2_prob calls: 1 time: 0.000778s
model.fits.d_func1 calls: 1 time: 0.000054s
model.fits.func1 calls: 2 time: 0.000025s
model.oem.OptimizationModel.make_data calls: 1 time: 2.516509s
model.utils.hour_group calls: 11,400 time: 0.774824s
11400
Generated by Python Call Graph v1.0.1
http://pycallgraph.slowchop.com

Figure 24: OE: function call tree with depth=4.

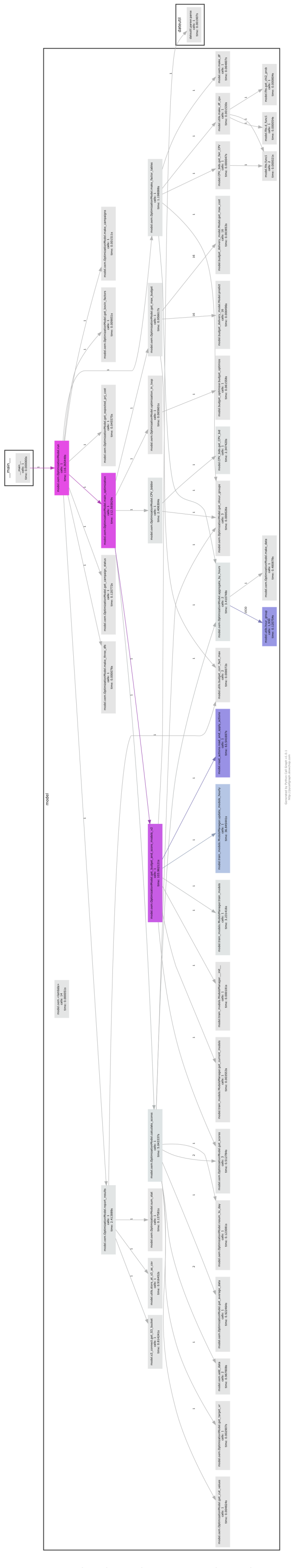


Figure 25: OE: function call tree with depth=4 with advanced (v2 model) methods added.