

Carousel for advertising

by Noesis Srl

<http://www.noesis-research.com>



CAROUSEL

Table of Contents

Basic example: one campaign, one placement	3
Many campaigns, many placements.....	6
Economics of campaigns	7
Adding behavioural targeting	7
Adding contextual targeting	8
Adding time targeting	9
The no-targeting case.....	9
Advanced topic: knowledge propagation.....	10
Advanced topic: beyond clicks	11
Carousel as ad server.....	12

Carousel for advertising

CAROUSEL



Basic example: one campaign, one placement

We have a pay-per-click campaign with 10 ads to be delivered on a slot in a web page. The campaign will run for two weeks. For each visitor we know the location, because we use the IP. We do not have behavioural or contextual information. The goal is to maximize the number of clicks over the whole period.

Each time a visitor lands on a page, we need to guess which ad has the greatest probability of gaining a click (the clickthrough rate). Carousel copes with this problem by building a targeting scheme like this at each moment

<i>Location</i>	<i>Ad</i>
City A	4
City B	2
City C	9
City D	4

Table 1

The meaning is obvious: when a visitor from city A lands on the page, we have to deliver ad #4, and so on.

Building such a scheme is a much more difficult task than you might expect at first glance. The problem is that we need to experiment combinations of locations and ads, to collect statistically significant samples, to derive targeting rules (as in the table) and then to continuously repeat this cycle in order to detect changes in the behaviour of visitors and performance of ads. The last issue is due both to statistical oscillations and to different life cycles of the competing ads. In the first phase we have to experiment enough to get reasonable confidence about which ad is the best for each location. When we get it, after some time there can be a change in the leadership (ad 7 overtakes ad 4 for city A), so we have to continue experimenting.

This process is not simply a statistical exercise: it is an investment. Experimenting is costly, because we have to try, say, ad 8 for visitors from city A though we know with reasonable confidence that ad 4 has a better CTR. We cannot afford to keep on delivering ad 4 for city A, because in so doing we risk missing a change in leadership, paying an *opportunity cost*. But nor can we afford to experiment too much in search



of new leaders, because we risk wasting impressions serving an inferior ad and losing clicks without any real benefit, so equally paying an opportunity cost.

We need knowledge about ad performance and targeting and this knowledge has a cost. In collecting not enough data we can fail in making the right targeting scheme. However, in collecting too much data we will manage to find a good scheme, but after having spent too many impressions (and clicks, and money). The problem is finding an optimal balance between *exploitation* of good targeting rules and *exploration* for even better ones. Moreover, we are required to continuously adjust this balance. This exploitation-exploration trade-off is the key to optimization.

Carousel solves the optimization problem by searching for the optimal exp-exp balance at each given moment, in an endless self-learning cycle. A useful metaphor can be the continuous rebalancing of an investment portfolio by observing the market behaviour and trying to anticipate its changes. Indeed, the deep algorithms of Carousel are really designed as a self-learning real-time investment system.

[Incidentally, a natural consequence of this approach is that Carousel optimizes the overall revenue in the long-term, while other systems are devoted to optimize here-and-now, which is a greedy and maybe myopic approach]

Now let us see how the example is implemented. Carousel is offered as a web service. The ad server calls the *GetPolicy* service. Carousel answers with a message containing table 1. From now on the ad server uses this table as delivery policy. When a visitor from city A lands on the page, it serves ad 4, from city B ad 2 and so on. After a while the ad server calls the *SetAnalytics* service. It provides Carousel with a table like this

<i>Location</i>	<i>Ad</i>	<i>Impressions</i>	<i>Clicks</i>
City A	4	2176	17
City B	2	849	5
City C	9	6730	71
City D	4	4098	39

Table 2

Carousel uses this new data in order to learn new knowledge about visitors' behaviour and ad performance and builds a new policy. When the ad server calls *GetPolicy* again, Carousel sends it the new table, e.g.



<i>Location</i>	<i>Ad</i>
City A	9
City B	9
City C	9
City D	4

Table 3

The process is then repeated as long as the campaign lasts.

Now for some technical details.

Messages are usually implemented as XML objects. Intervals between two successive calls are up to the ad server, depending on its ability to refresh analytics. Some ad servers can provide new data every 15 minutes, others every hour or day. With more frequent updates Carousel learns more quickly, of course, but the process remains the same.

Analytics will be generated by the ad server, extracting them from the reporting system in use.

Besides *GetPolicy* and *SetAnalytics* Carousel offers some accessory services for configuring campaigns and providing some other information, but these two services are the essential tools for running a campaign.

The scheme can be extended in many ways. For example, a policy can be represented with a slightly richer table:

<i>Location</i>	<i>Ad</i>
City A	4 in 68%, 9 in 32%
City B	2 in 86%, 4 in 12%, 9 in 2%
City C	9 in 100%
City D	4 in 53%, 9 in 38%, 7 in 9%

Table 4

In other words, if the visitor is from city A, deliver ad 4 with 68% frequency and ad 9 with 32% frequency and so on. This variant is very simple to implement and allows quicker learning.

This interaction protocol between Carousel and the ad server is really clear, elegant and simple to implement. It was designed to minimize implementation effort on the



ad server side. Moreover, it is quite robust. The traffic is handled by the ad server, not by Carousel. If for some reason the connection between the ad server and Carousel is temporarily suspended, the ad server simply continues with the same policy until the connection is established again, with only negligible loss.

The specific structure of messages is agreed between Noesis and the ad server provider taking into account specific applicative requirements and technical issues. In practice, the scheme shown above is immediate to implement in any realistic version.

Many campaigns, many placements

If we have many campaigns and many placements, the previous scheme still works unchanged. Indeed, a delivery policy will be something like

<i>Placement</i>	<i>Location</i>	<i>Ad</i>
1	City A	4
1	City B	8
1	City C	2
2	City A	9
2	City B	4

Table 5

The meaning is obvious. The previous one-campaign-one-placement case is really only a particular case of this one. Of course, analytics will be detailed for placement, too:

<i>Placement</i>	<i>Location</i>	<i>Ad</i>	<i>Impressions</i>	<i>Clicks</i>
1	City A	4	2,176	17
1	City B	2	849	5
2	City C	9	6,730	71
3	City D	4	4,098	39

Table 6

In this case of use many campaigns share many placements; at its most extreme, we can have a run-on-site scheme, where all ads run on all slots on all pages on the site.



Many channels: Carousel for agencies

The previous description is from the publisher point of view. Yet, when used by an agency, Carousel works following the same philosophy. In this case, ads are placed on channels (networks like Google or Yahoo!) instead of placements.

The optimization process is made more complex and richer by issues about prices and auctions, yet Carousel is able to handle that aspect with its algorithms. The idea is still to balance exploration and exploitation, searching for the right bid while searching for the right placement and the right target.

A policy is something like

<i>Channel</i>	<i>Bid</i>	<i>Ad</i>
1	123	4
1	186	8
2	97	9

Table 7

Economics of campaigns

Campaigns have their own microeconomics: cost budget, revenue goals, milestones, contractual bonuses and penalties. Carousel is designed in order to take into account such aspects. For example, it can privilege ad A over ad B, though B has a better CTR, because A needs to be “pushed” towards its goals for contractual reasons (or simply because the advertiser of A is preferred to the advertiser of B). What is interesting is that these factors are not absolute constraints (i.e. “prefer A to B”) but drivers of the exploration-exploitation dynamic balance (“B’s better CTR vs. need to push A because a long way from budget at the moment”).

Adding behavioral targeting

If we have a behavioural targeting system, we can add it to Carousel.

Suppose that visitors are classified in 4 profiles/segments. A policy will have this structure:



<i>Profile</i>	<i>Ad</i>
Profile A	4
Profile B	2
Profile C	9
Profile D	4

Table 8

The profile plays the same role as location in previous examples. Both location and profile can be used, even adding placements, as in

<i>Placement</i>	<i>Profile</i>	<i>Location</i>	<i>Ad</i>
1	A	City A	4
1	A	City B	8
1	B	City C	2
2	C	City A	9
2	C	City B	4

Table 9

Carousel is designed to use behavioural targeting schemes as additional resources. The BT engine is not part of Carousel, but an add-on for Carousel. It can be provided by a third-party or by Noesis itself. If by Noesis, it is a distinct, though coordinated, project in addition to the Carousel service. Noesis is indeed able to provide data mining, which is a distinct business area (as for the client Gucci). Anyway, the customer can use proprietary or third-party BT systems without any problem for Carousel.

Adding contextual targeting

Context, that is page profiles extracted with text analysis, are handled like visitor profiles. Previous considerations about behavioural targeting are still valid for contextual targeting. The contextual system can be provided by a third-party or by Noesis, as the customer prefers. Noesis has its own text mining technology, used for news recommendation, which is another business area for Noesis (as for the client La Repubblica).



Adding time targeting

Ad performance and visitor behaviour can be quite variable with respect to time, whether time of the day or day of the week. Thus it is reasonable to introduce time as another column in the policy table:

<i>Placement</i>	<i>Profile</i>	<i>Location</i>	<i>Time</i>	<i>Ad</i>
1	A	City A	Morning	4
1	A	City B	Afternoon	8
1	B	City C	Morning	2
2	C	City A	Night	9
2	C	City B	Evening	4

Table 10

The general mechanism still works in the same way.

The no-targeting case

If we have no targeting information, we can use Carousel in a very basic mode with policies like

<i>Placement</i>	<i>Ad</i>
1	4
2	8
3	2
4	9
5	4

Table 11

Here Carousel simply suggests an ad for each placement (remember that it could be preferable to suggest many ads at the same time, as in table 4, but the concept is the same).

If we have only one placement, here is a minimal policy example:



<i>Ad</i>
4 in 68%, 9 in 32%

Table 12

Carousel is giving only one suggestion to be used in the next cycle, until new analytics are available.

This case of use is not without practical interest in some operational environments (as in run-on-site models).

Advanced topic: knowledge propagation

[A reader not interested in scientific aspects can skip this section]

A careful reader with some statistical background could raise an issue: if we use many “situational” attributes, like location, behavioural profile, contextual profile, time, demographics etc., the policy table could explode in a huge number of rows. That phenomenon would cause trouble, both in the dimension of messages exchanged between Carousel and the ad server and in the fragmentation of samples used for generating policy rules.

Indeed, that criticism is well-founded: for just this reason Carousel is equipped with a *proprietary technology for learning in a high-dimensional space with sparse data*.

When learning something about a pattern, Carousel learns about similar patterns, too. Imagine the ad server calling the *SetAnalytics* service providing this table

Placement	Location	Profile	Ad	Impressions	Clicks
1	A	X	6	10,000	100
1	A	Y	6	20,000	230
1	B	X	6	100	2
1	B	Y	6	200	1

Table 13

The first row says that ad 6 has a CTR=1% in a certain situation (placement 1, location city A, visitor profile X). This CTR is computed from a sample of 10,000 impressions, which offers certain support for statistical inference. The second row says that the same ad 6 has a greater CTR with a visitor of profile Y, placement and



location being the same. The third and fourth rows (related to city B) offers negligible statistical support for inference, because samples are too small.

We would like to have more data about ad 6 vs. city B, but we cannot enforce traffic from B (of course) and anyway we do not like to experiment too much for economic reasons. The question is whether we can learn more about ad 6 vs. city B while saving investment in exploration thanks to exploitation of already available knowledge?

The answer is affirmative in principle, though far from simple in implementation. Patterns in rows 3 and 4 are similar to patterns in rows 1 and 2. On one hand, rows 3 and 4 say that ad 6 performs better on profile X than Y; on the other hand, rows 1 and 2 say the contrary *in a similar situation*. There is a conflict among exact but small samples and large but approximate samples. Intuitively we need to find a balance in inference driven by conflicting samples. Correct enough, but we really need another thing, much deeper and more subtle: we need *to use heterogeneous and possibly conflicting sources of knowledge in order to dynamically and economically drive the balance between exploration and exploitation*.

This is a formidable task, beyond the state of the art in industry and also challenging as a goal for academic research. Carousel exploits a proprietary breakthrough technology to cope with this problem. The point is vital: systems without such a technology are faced with a data-devouring problem, not even solvable with huge traffic on the site and huge data samples available. Hence, they are compelled to oversimplify the problem with strict hypotheses, which make the task seem more attackable, but not really the original task. In practice, a fine-grain visitor/page segmentation needs to be collapsed in a rough-grain one on the basis of human-chosen criteria. The effect is to reduce automation and to reduce self-learning abilities, which in the long run means more effort and less optimization, i.e. more cost and less revenue. Carousel makes a dimensionality reduction too, as unavoidable, but it does so with original algorithms designed over a long scientific research effort.

Advanced topic: beyond clicks



Until now we have been speaking about the pay-per-click revenue model. It is not the only reasonable model while the venerable CPM is declining. Other revenue models are gaining considerations as candidate for the near future of web advertising. We can speak about pay-per-action, pay-per-conversion, pay-per-time-spent and buzzwords of this kind. This is a real need. Carousel is designed to work with a general concept: *pay-per-value*. Any event or state bringing some kind of value can be used for optimization. What you need is a certain set of metrics and a reporting system providing analytics for those metrics. N.B. not only one metric, but a set of metrics, maybe many. You can assign value to clicks, actions, conversions, downloads, time spent, sales, even simple impressions (incidentally, in Carousel old-style CPM campaigns can be intermixed with more modern pay-per-something). Carousel optimizes the overall value in the long run taking into account all the value metrics simultaneously.

As one can easily guess, setting up metrics and reporting for metrics is not a trivial task. Once again, such a system is not part of Carousel. Carousel is designed to use it if provided by someone (the customer, a third party or Noesis itself).

Carousel as ad server

We saw Carousel working as a pure optimizer, while an ad server is in charge of traffic management and ad delivery. Carousel can work as a full-featured ad server, too. The mechanism is the traditional one: Noesis put a script on each page involved; when an user browser downloads the page, it calls Carousel which answers delivering the chosen ad.

In principle, this option is offered for customers preferring to have a single provider both for choosing and serving ads. In practice, often it is the best choice for a pilot project. The client may want to try Carousel's potential without engaging the ad server provider in the first stage, due to relational, contractual or operational reasons. In the next step the client will be fully confident in requiring the ad server provider to collaborate with Noesis.

A variant of this service model is Carousel serving not the ad itself, but a link to the ad, which will be delivered by a third-party ad server. Again, this option can be useful



as first-step: the campaigns to be optimized are still handled by the ad server, with Carousel choosing the ad and the third party serving it.

The web service protocol described in previous sections remains the most recommended option for a stable, long-run and large-scale partnership. Those described here can be sometimes useful to remove possible initial resistance and relational or contractual issues.



For additional information, please contact:

Noesis s.r.l.

Corso Italia, 89

I56125 Pisa

Italy

Telephone +39 050 9911080

Fax +39 050 9911588

Email info@noesis-research.com

www.noesis-research.com

Carousel for advertising

CAROUSEL

