Unica Optimize

Version 8.0.x

Troubleshooting Guide

Publication Date: December 18, 2009

Copyright

Unica Corporation
Reservoir Place North
170 Tracer Lane
Waltham, MA 02451-1379

Examples and Data

All software and related documentation is subject to restrictions on use and disclosure as set forth in the Unica Software License and Services Agreement, with restricted rights for U.S. government users and applicable export regulations.

Companies, names, and data used in examples herein are fictitious unless otherwise noted.

Trademarks and Patents

Unica, the Unica logo, NetInsight, Affinium and MarketingCentral are registered trademarks of Unica Corporation with the U.S. Patent and Trademark Office. MARKETING SUCCESS STARTS WITH U is a trademark. All other trademarks are the property of their respective owners.

Portions of the software described in this document are covered by U.S. Patent Numbers: 6,317,752, 6,269,325, 6,542,894, and 6,782,390.

The NetTracker and Unica NetInsight products are licensed under the following patents and patent publications: US5,675,510, US6,115,680, US6,108,637, US5,796,952, US6,138,155, US6,653,696, US6,763,386, AU0701813, BR9609217, CA2223919, EP0843946, JP03317705, MX193614, NO09705728, AU735285, CA2246746, CN1174316, CN1547123, CN1547124, DK870234, DE69720186, ES2195170, AU727170, BR9808033, CA2284530, CN1251669, IL131871, JP2000514942, KR341110, NZ337756, WO9641495, EP0870234, EP1130526, EP1168196, US20040078292, WO9810349, US20050114511, US20040221033, WO9843380.

Markup functionality in Unica Marketing Operations is enabled through use of third-party software components from AdLib™ eDocument Solutions and Adobe® Acrobat®. "Powered by AdLib™."

Table of Contents

1 Troubleshooting Optimize sessions

Troubleshooting Optimize sessions

Many issues that you might encounter in Optimize sessions can be resolved by obtaining details from the session logs and then fixing the specific problems that are identified.

Error	Description of Problem	Action to Take
Failure to start Solver engine	This error usually indicates a missing or invalid license file.	Check for the license file in the /bin directory for xpauth.xpr

Error	Description of Problem	Action to Take
The LP solver was unable to find an optimal solution to the chunk problem	The outer algorithm (the part that handles the capacity rules) has found that the problem given to it for a customer sample is not solvable. This could mean that there is a logical contradiction in the capacity rules, or it could mean that a solution to those rules is not possible with the given data. This error can also occur if the scores used for the proposed contacts exceed the numerical precision of the floating point math used. In general, you should not exceed a range of 1.0 to 1.0e+11.	Look for problems with the logic of the capacity rules, and for mismatches with the rules and the proposed contact data. For instance, if a capacity rule requires a minimum of something, make sure the PCT has at least that many of that item. cAlso check for sample-related provisioning problems. If there seems to be no problem with the rules and data, use the `configtool`utility to change the `Optimize|Debug|SolverDebugEnable` item so it is no longer hidden. Then select this property in Configuration Manager, and set its value to TRUE. Now run the session again (making sure you are in single thread mode). In the log directory, some files will be created that start with the letters "ACO_". The rest of the filename is made up of two numbers, where the first is the customer sample number and the second is the LRE iteration for that customer sample. The file with the highest iteration number for the highest sample number contains the linear programming model that got the error. Send this file to Unica Technical Support so that they can analyze it.
The generation loop was unable to eliminate all slack and surplus variables	The outer algorithm (the part that handles the capacity rules) makes progress by creating alternative solutions to the per customer rules generated by the inner (core) algorithm. It does this by temporarily changing offers scores, and looking for solutions that have not yet been generated. If the outer algorithm cannot satisfy the capacity rules with any of its alternative solutions, and the core algorithm is not creating any new alternative solutions, this is the error you will see. This could mean that the per customer rules and the capacity rules are inherently in conflict, so no solution is possible, or it could mean the data is such that a solution is not possible.	Look for problems with the logic of the rules, and for mismatches with the rules and the proposed contact data. For instance, if you had a percustomer minimum of 1 on all offers, channels, and segments, this would result in at least as many offers as there are customers in the results. If you also had a capacity rule with a maximum value that was less than the number of customers, this would cause an inherent logical conflict between the per customer rule and the capacity rule, and you would see this error. Also check for sample-related provisioning problems.

Error	Description of Problem	Action to Take
Unprocessable customer	At the end of each session run, there are some log entries that summarize the results. One of the entries is: "Total # of Unprocessable Customers". This indicates that no solution could be found for the per customer rules for the number of customers shown. When this occurs, it is not a fatal error – the result is that the "unprocessable" customers receive no offers. You can get a separate log entry for each unprocessable customer by setting the `Optimize|logging|enableBailoutLogging` property to TRUE. An example of an inherently unsolvable problem is when you have a per customer minimum greater than zero, and an insufficient number of proposed transactions to meet that minimum.	Unprocessable customers can occur either because the rules and data make it impossible to get a solution, or because the number of tries has been exceeded. The number of tries is configured by the `Optimize|AlgorithmTuning|MaxAlternativesPerCustomerEvaluated` property. Setting the value of this property to a higher number lowers the likelihood of the customer being unprocessable (assuming that it is not inherently unsolvable), but when it occurs, it also makes the performance penalty higher. An example of an inherently unsolvable problem is when you have a per customer minimum greater than zero, and not enough proposed transactions to meet that minimum.
No offers	If a particular customer receives no offers, it is not necessarily an error. Unless a per customer minimum is found in the per customer rules, it is perfectly legal to reject all offers of some customers, as long as no rules are violated and the overall score is maximized. It can also be a side-effect of an error, as when there is no legal combination of offers given the rules, or when the customer is unprocessable.	Check whether a per customer minimum exists in the rules, ensure that given the rules, all combinations are legal, and check whether there are any unprocessable customers.
Invalid size provided to the init count table. (1,0): CODE 5: Internal Error 5	No channel offer attribute values have been defined.	You must define some channel offer attribute values.

Optimize session takes a long time to run

Here are troubleshooting steps you can follow if you believe your Optimize session is taking too long to run

Before you start

1. Make sure the session was not running with `Optimize|Debug|ExtraVerbose` enabled, as this setting causes slow run times.
2. Make sure you are using a DB loader, and that it is properly configured.
3. If you are using time intervals with your rules, make sure the contact history tables for your audience level are indexed.
4. Set the Optimize server logging level to MEDIUM or LOW.

Run a session to generate a clean log for troubleshooting

Run a session to generate a clean Optimize server log with the HIGH or ALL setting on. While your session is running, do not access any Optimize reports, as this will add data to the log that may confuse things.

When you have the generated log, you will check for 2 things:

- The amount of time spent accessing the database to set up the data needed for the session.
- The amount of time spent processing customer samples (chunks).

How to check the amount of time spent accessing the database

Using the clean log you generated, follow these steps to find out how long Optimize is taking to access the database to set up the data needed for the session.

1. In the Optimize server log, search for the string: `LRE Starting chunk: 0`
2. Take the timestamp of this entry, and subtract from it, the timestamp of the first entry in the log. The difference is the amount of time spent accessing the database to set up the data needed to run the session.

   If the value seems too high, look at the start and end timestamps for the queries that comprise the log section preceding `LRE Starting chunk: 0` to identify which one is taking too long.
3. Then, troubleshoot the task that took too long in the same way as you would any other database performance issue.

How to check the amount of time spent processing customer samples (chunks)

Using the clean log you generated, follow these stpes to find out how long Optimize is taking to process customer samples.

1. In the Optimize server log, subtract the time stamp from the line matching `LRE Starting chunk: 0` from the time stamp of the line matching `Run Thread terminated`.

This will tell you the total time spent in the CPU-intensive optimization section. If this is where the bulk of the time is being spent (typically, this is the case), you can get a better idea of what is going on by looking at the chunk iterations.

The optimal solution for each chunk is found by applying a set of scores to the offers in that chunk, finding the optimal solutions with those scores for the chunk's customers by using the core algorithm, then using the result in the outer algorithm to find a new set of scores to try. Each time this happens, it counts as one chunk iteration. The amount of time spent in the CPU-intensive section will be roughly proportional to the average number of iterations per chunk.

Troubleshooting Sample-related Provisioning Problems

To handle large volumes of data while not sacrificing the quality of the results, and at the same time getting the results in an acceptable amount of time, certain requirements are made regarding the makeup of the session's proposed contacts. One of the strategies Optimize uses is to break the proposed contact data into random subsets of approximately equal numbers of customers; it then optimizes the proposed contacts of each of these samples independently. If multiple threads are configured and supported by your hardware, these customer samples will be processed concurrently.

There is a class of problems which can result in errors or suboptimal results that are a side-effect of the customer sample approach. The number of customer samples that will be used for a session run is determined by dividing the number of customers in the PCT by the value of the configuration parameter `Optimize|AlgorithmTuning|CustomerSampleSize`. It is important that there are enough proposed contacts matching each capacity rule to allow each random customer sample to be statistically similar relative to every feature used by a capacity rule.

For example, say we have 1 million customers, and have a configured customer sample size of 1000. This implies that we will have 1000 customer samples. Imagine that we have a capacity rule that is set up as: minimum 1 email, maximum 5000 emails. What Optimize does in this example is to take the rule constraints and modify them to spread that rule across the customer samples. In this example, the maximum 5000 emails constraint is divided by the number of samples, so that each sample is processed with a maximum 5 emails constraint. But what do we do with the minimum 1 email constraint? We cannot have each sample requiring a minimum 1/1000 of an email!

Instead, we randomly pick one sample to process with a minimum 1 email constraint, while the other 999 samples are processed with no minimum email constraint. This all works fine, unless there are not enough proposed contacts using email, to make sure all 1000 samples get at least one email. If your proposed contacts only contain 500 contacts using email, there will be a smaller than 50% chance that a particular sample will contain an email. That means you have a greater than 50% chance that the session will be aborted with an error, because the minimum cannot be satisfied, even though 500 times that minimum were present in the proposed contacts. In order to avoid this situation, any feature used in a capacity rule should be well-represented relative to the number of samples.

2 General performance tips for Optimize

General performance tips for Optimize

Keep these points in mind when making data or configuration decisions, if you are concerned about performance.

- In general, larger PCTs will take longer to process than smaller ones, in both the IO-intensive data setup and CPU-intensive sections.
- Larger numbers of proposed contacts per customer makes the core algorithm work harder in the CPU-intensive section.
- A larger value of `Optimize|AlgorithmTuning|CustomerSampleSize` will take more memory and longer CPU-intensive processing than a smaller value. There is a tradeoff here, since larger values can give more optimal results. Also, smaller values increase the likelihood of encountering sample-related provisioning problems.
- If you use a time interval in your rules, this will add processing time in two ways:

1. Contact history will be queried, and this can be slow since those tables are often very large.
2. The number of rules is multiplied by the number of time windows required by the interval. This makes the CPU-intensive part do more work.

Multithreading

If you see from the log timestamps that much of the session run time is in the CPU-intensive section, and the Optimize server is running on hardware that supports data-intensive processing in multiple threads, the run time of the CPU-intensive section can be greatly decreased by setting up the configuration for multithreading.e

Use cases that can negatively affect performance

This section lists various use cases that can negatively affect the performance of Optimize.

Smart Offer Lists with Rules Using Offer Versions

If you use smart offer lists with rules that use offer versions, there are additional queries used in the IO-intensive data setup section. When the number of offers in the lists is large, and the number of attributes per offer is large, the time taken to execute these queries can be great.

High Maximum for Iterations per Customer Sample

The maximum number of iterations to use for each customer sample is configurable using the `Optimize|AlgorithmTuning|MaxIterationsPerCustomerSample` property.

Depending on the rules and data, this limit may not be reached by a customer sample. High values guarantee the highest level of optimality of the results, but often the the use of a greater number of iterations does not make a big enough improvement in optimality to justify the performance penalty. Typically, five iterations yields an acceptable degree of optimality, and it is unusual to see more than about a dozen or so iterations ever required.

To analyze the customer sample iteration behavior, search in the Optimize log for the string `Iteration:`. This will be followed by a number, indicating which iteration it is. Each chunk starts at iteration 1 and counts up. It helps to see what is going on if you get a count of each iteration number in the log, and use the results to construct a histogram.

High Number of Unprocessable Customers

Another major factor in performance is the number of unprocessable customers. If the value of the `Optimize|AlgorithmTuning|MaxAlternativesPerCustomerEvaluated` property is a large number (over 100 or so), the time penalty will be high whenever a customer is unprocessable.

When you have many unprocessable customers, you should look for logical errors in the rules or data, but it is possible (especially with large numbers of proposed transactions per customer) that the time needed to get some per-customer solutions is just very high. If this is the case, it may be best to reduce the value of the `MaxAlternativesPerCustomerEvaluated` parameter, accepting more unprocessable customers as a trade-off to improve performance.

In Optimize version 7.5.3 and later, there is more detailed logging to show the minimum, maximum, and average number of alternatives evaluated for each customer sample.

Solver Subroutine Calls

If certain combinations of per-customer rules are used, a major performance penalty may be seen in some cases. This can occur when there is at least one per-customer Min/Max # Transactions rule where the minimum constraint is not zero, combined with one or more package rules.

In versions older than 7.5.3, "Never A with B" counts as a package rule here.

In addition to having these two rules, their scopes must overlap so both are applied to the same proposed transactions. In addition, the scores have to be such that the preferred solution for a package rule causes the "Min/Max" rule to fall below its minimum. If all these conditions are met, the core algorithm cannot find the optimal results in an efficient way, and has to use a slower call to the solver engine. You know this is occurring if you see this message in the server log:`Solver subroutine parameters:`

If you are seeing performance issues from using "Never A with B" rules, the best way to improve performance is to upgrade to Optimize version 7.5.3 or later.

Many Cases Where Scores are the Same

If there are many cases where the scores are the same, decision-making in the LRE can sometimes get very inefficient. You can tell this is happening if you see this string in the server log: `Additional alternative generated:`

To avoid this situation, try assigning more varied scores to the proposed transactions.