Heuristic algorithm for ranking torrent search results

Steffan Norberhuis Quinten Stokkink

27 May 2013

1 Introduction

Our Tribler extension needs to handle (movie) torrent search result ranking based on some complex parameters. Given that there is no perfect way to order torrents based on more than just seeders and leechers (taking into account user preferences and such) we have produced a heuristic algorithm to handle this sorting. This document will describe the choices that were made in designing this algorithm.

2 Inputs

The inputs for our algorithm are the following ones:

- USER_RATING: A user rating for a provider, a floating point number ranging from 0.0 to 1.0
- NUM_LEECH: The number of leechers for a torrent, an integer ≥ 0
- NUM_SEEDS: The number of seeders for a torrent, an integer ≥ 0
- NAME_MATCH: The matching of the torrent name compared to the movie details, a floating point number ≥ 0.0
- SPEC_TERMS: The matching of special terms, set by the user, compared to the torrent name, a floating point num ber ≥ 0.0

USER_RATING

The user rating is given to a torrent search provider by the user. This allows the user to prioritize sites they trust. For example one might rank torrents from *www.website.com* better than torrents retrieved from *www.shady.com*. In this example the algorithm should choose a torrent from *www.website.com* over a comparable one from *www.shady.com*. The user can specify this preference using a floating point number on the interval from 0.0 to 1.0, corresponding to a certain trust percentage.

NUM_LEECH

The number of leechers for a torrent. This is an integer larger or equal to 0.

NUM_SEEDS

The number of seeders for a torrent. This is an integer larger or equal to 0.

NAME_MATCH

This number specifies the summation of all the fuzzy matches between the torrent name and the parameters of the movie we searched for. For example $(MovieName, Directors)$ matches "MovieNameDirectors" for a total of 100% and "MovieNamer" for 50%.

SPEC_TERMS

The user can also specify terms that signify some sort of quality. For example if we encounter "TheQualityReleaseGroup" in a torrent title and the user has marked this word as being an indicator of quality, the search algorithm should rank this result higher than other search results.

3 Algorithm

What the algorithm must do is provide some sort of rank per search result. We choose the format of "the higher the rank, the better the search result". To produce this rank we will calculate two sub-results and then use these sub-results to produce a final rank. The sub-results are:

- POT_SPEED: The potential download speed we could get from a torrent, a floating point number ≥ 0.0
- TECH_WANT: The combination of how much the torrent name matches the search and special term criteria, a floating point number ≥ 1.0

POT_SPEED

To estimate the speed of a torrent we look at the number of seeders and leechers it has. We also note that leechers inherently provide slower download speeds to us than seeders do. As a rule of thumb, we estimate a leecher gives us about half the speed of a seeder. What this actual speed is, is of no importance. The potential speed (expressed in units of avarage download speed from seeders for an arbitrary torrent) then becomes:
$POT\_SPEED = NUM\_SEEDS + 0.5 \times NUM\_LEECH$

TECH_WANT

To estimate the relevance of a torrent we base ourselves on the user's search criteria and special terms. We assume a good torrent is well named and specifies some quality identifiers in its name. This matching to our criteria will weigh any potential download speed. We assume that a name match is worth as much as a special term. Combining weights (and asserting we never end up with a weight of 0, because there are no special terms or name matches) we end up with the following formula:
$TECH\_WANT = (SPEC\_TERMS + 1) * (NAME\_MATCH + 1)$
Note that if multiplied with the download speed this will be equivalent to first weighing with the name matching rate and then weighing with the special term matching rate.

RANK

We can now finally calculate a rank for a torrent. If we multiply the relevance of the

torrent to the potential speed we can get we get an estimate for how much we want this torrent. For example a torrent with extremely high download speed but a slightly dodgy name will beat a torrent that "looks good" but provides little to no download speed. Lastly we also want to factor in how much a user cares for a certain provider. If a user knows that, however popular a torrent may be, a certain site produces bad torrents mainly, the algorithm should downgrade the search result. The final ranking algorithm then becomes:
$RANK = USER\_RATING \times TECH\_WANT \times POT\_SPEED$