Optimising Topical Query Decomposition

Marcin Sydow [1]* Francesco Bonchi[2] Carlos Castillo[2] Debora Donato[2]

[1]Web Mining Lab, Polish-Japanese Institute of Information Technology, Warszawa, Poland
msyd@pjwstk.edu.pl

[2]Yahoo! Research Barcelona, Spain
{bonchi,chato,debora}@yahoo-inc.com

ABSTRACT

Topical query decomposition (TQD) is a paradigm recently introduced in [1] , which, given a query, returns to the user a set of queries that cover the answer set of the original query. The TQD problem was studied as a variant of the set-cover problem and solved by means of a greedy algorithm.

This paper aims to strengthen the original formulation by introducing a new global objective function, and thus formalising the problem as a combinatorial optimisation one. Such a reformulation defines a common framework allowing a formal evaluation and comparison of different approaches to TQD. We apply simulated annealing, a sub-optimal meta-heuristic, to the problem of topical query decomposition and we show, through a large experimentation on a data sample extracted from an actual query log, that such meta-heuristic achieves better results than the greedy algorithm.

Categories and Subject Descriptors
H.2.8 [Database Management]: Database Applications - *Data Mining* H.4.3 [Information Systems Applications]: Communications Applications

General Terms Algorithms

Keywords Query logs, Query Decomposition, Query recommendation, Objective Function, Simulated Annealing.

1. INTRODUCTION

Search engines collect daily a huge volume of valuable information about the search activity of their users. Extracting behavioural patterns from this wealth of information is a key step towards understanding user needs and improving the services provided by search engines.

A common query log mining application is query recommendation: given a query submitted by a user, a list of other related queries is returned to the user. Such queries are extracted from query logs and ordered by relatedness to the one original query, or by frequency in the query log.

Topical query decomposition (TQD) [1] is a recently introduced paradigm that, similarly to query recommendation, assists users in finding the information they are looking for. Differently from the classical query-recommendation setting, the goal is not to offer the users with the *best* query, i.e., the query that is more likely to capture their information needs; instead the goal is to provide to the users a suitable set of queries that represent the different topical groups underlying a query. The main intuition is that, since the user could be not aware of all the possible facets related to a single topic, great advantage could be provided by a tool able to discover all the possible sub-topics subsumed in the user query.

The problem, originally stated and studied in the paper [1], was presented as a variant of the set-cover problem and it was solved adapting the greedy algorithm presented in [3]. In this paper we outline the limitations introduced by the original approach and extend it by introducing a new global objective function.

*The first author worked on this project during his research visit at Yahoo! Research Barcelona in November 2008.

2. BACKGROUND AND CONTRIBUTION

In this Section we recall the formal definition of the TQD problem in order to outline the main limitations of its previous formulation. Moreover we briefly list the main contributions we have done in order to address such limitations.

2.1 The problem Statement

For a given search engine, its current state of index X and current log L containing the queries submitted to it in a given period, let $docs : L \to 2^X$ be a function such that for any $q \in L$, $docs(q)$ denotes the set of documents returned by the search engine to the query q.

Each instance of the problem of query decomposition studied in this paper can be described as follows.

Problem INPUT:

1. the **original query** q_0 submitted with the set of returned documents $D_0 = docs(q_0)$. We will call all the documents in D_0 as *blue documents*. In addition, each document $d_i \in D_0$ has its associated weight or "importance" w_i. The weights are precomputed using the information contained in query log concerning the clicks on documents in D_0. Currently, the number of clicks on a document (i.e. simple document popularity) is used as the weight of importance. More robust characteristics than document popularity could also be used.

2. the set Q of "candidate" queries purposely extracted from the query log. Assume the candidate queries in Q are indexed by the set of indices $I = [1, |Q|]$ so that $Q = \{q\}_i$, $i \in I$ and, for each candidate query q_i, the set of documents $D_i = docs(q_i)$ it returned. The

set Q is chosen so that each query $q_i \in Q$ has the property $|docs(q_i) \cap D_0| \geq k$, for some parameter k , i.e. it overlaps with at least k returned docs from D_0. In our current settings, 2 is used as the value for k. In addition, each candidate query q_i has associated a value of measure reflecting the degree of not being "coherent", c_i (the lower c_i the more coherent is the query q_i). Currently, the values of c_i are computed as scatters of documents in $docs(q_i)$. Intuitively, if a query is "coherent" the distances among all of its documents in text-vector space are small.

Problem OUTPUT: a subset Q_d of the set of candidate queries Q which decomposes the original query q_0 wrt some desired properties, described below.

2.2 Quality of Decomposition

Assume that Q_d is indexed by the set of indices $D \subseteq I$. Further, for a given query decomposition problem and its solution Q_d, let U denote the union of documents covered by all the queries in Q_d, i.e.: $U = \bigcup_{q \in Q_d} docs(q)$. Then the quality of the decomposition Q_d is measured in terms of 4 factors:

1. high coherence of the selected decomposing queries, in terms of the values c_i;
2. low number of returned documents outside of the original returned set (i.e. low $|U \setminus D_0|$). We will call such documents as *red documents* ($U \setminus D_0$);
3. low overlap of documents returned by different decomposing queries (i.e. for all $p, q \in Q_d$, such that $p \neq q$, $docs(p) \cap docs(q)$ should be as small as possible);
4. covering as much of the documents in D_0 as possible, either in terms of number of documents (i.e. $D_0 \cap U$ should be as close to D_0 as possible) or in terms of total weight $\sum_{i \in D} w_i$ of covered "blue" documents.

2.3 The Greedy Solution

The problem was originally [1] presented as a variant of the set-cover problem and solved adapting the greedy algorithm presented in [3]. The greedy algorithm proposed there, in each step, added a new element to the decomposing set, until the predefined fraction of weight of documents was covered. Each step was chosen in a greedy manner according to the value of a "local" objective function. The function had 3 parameters: λ_1, λ_2 and λ_3 which controlled the importance of the first 3 out of 4 factors listed above. In addition, the stop condition of the greedy algorithm directly corresponded to the 4th factor above.

Despite this, there was no *global* objective function that the greedy algorithm optimised, so it was not easy to compare its results with the results of other approaches to the problem we subsequently developed, and which use the notion of the global objective function to be optimised, such as simulated annealing.

2.4 Contributions

In this paper, we strengthen the original framework by making it more formally defined. In particular, we introduce a *global* objective function and re-define the problem as a combinatorial problem of optimising (minimising) this function. This approach makes it possible to more naturally compare the results of various approaches to the problem. Finally, we apply simulated annealing, a sub-optimal meta-heuristic, to the problem of topical query decomposition and we show, through a large experimentation on a data sample extracted from an actual query log, that such meta-heuristic achieves better results than the greedy algorithm.

3. RELATED WORK

The original formulation of the topical query decomposition problem and the two variants presented in this paper are reformulations of the set cover problem. The set cover formulations and the adaptations of the greedy algorithm [3] we use are inspired by related variants of the set cover problem in the literature, such as the *red-blue* set cover problem [2, 7], and set cover with minimising the overlap of sets [5].

Simulated Annealing [6] is a randomised algorithm that attempts to simulate the physical process of annealing. The original motivation is to simulate the behaviour of molecules when a material is annealed into optimal crystal structure. Such method is a sub-optimal meta-heuristic for solving combinatorial optimisation problems based on the key notion of neighbourhood relation.

4. OBJECTIVE FUNCTION

In this section we introduce two variants of a global objective function, and we discuss their merits and limits.

4.1 Variant 1

The first variant that we introduce naturally extends the "local" objective function described in [1] which used only the first 3 factors described above (see "quality of a decomposition"). We add a fourth factor, directly controlling the coverage of original important documents D_0 by the decomposing queries. More formally, for a given decomposition D_q the factors are:

1. (cost) $cost = \sum_{i \in D} c_i / \sum_{i \in I} c_i$
2. (red fraction) $redfrac = |U \setminus D_0|/|U|$
3. (inter-query overlap). Let $nq(d)$, for any document d returned by any candidate query, denote the number of distinct candidate queries which returned this document. $iqover = \sum_{d \in (U \cap D_0)} nq(d)/|U \cap D_0|$
4. (uncover) $uncover = |D_0 \setminus U|/|D_0|$

The objective function is controlled by 4 parameters λ_i, $i \in [1, 4]$ which sum up to 1, and its value is a convex combination:

$$of_1(Q_d) = \lambda_1 cost + \lambda_2 redfrac + \lambda_3 iqover + \lambda_4 uncover.$$

4.1.1 Discussion

Parametrisation of the function allows to study the quality of decomposition for various different applications.

For example, in the case of query disambiguation the value of λ_3 should be relatively high since the decomposing queries should be non-overlapping with others, in case of query refinement, the value of λ_2 should be high so that the decomposing queries tend to narrow the search instead of making it broader, in case of clustering the results the high value of λ_1 additionally ensures that the clusters induced by the decomposing queries tend to be coherent, etc. Arbitrary linear combinations of the parameter values make it possible to flexibly emphasise various aspects of the particular application or even to optimise the decomposition in the context of some novel applications, other than mentioned above.

We additionally made all the parameters constrained to sum up to 1.This is different to the approach in [1], where all the (three) parameters could have any positive values. The new constraint makes the problem statement more elegant from the mathematical point of view without loosing the generality. Most importantly, adding the constraint avoids the problem of redundant settings. For instance, without the constraint, the quadruple of parameter values $\langle 40, 0, 10, 0\rangle$ would mean exactly the same as $\langle 4, 0, 1, 0\rangle$, despite the values are different. Both the settings are uniquely represented as $\langle 0.8, 0, 0.2, 0\rangle$ with the constraint. Due to this, the parameter space is now much simpler.

The variant 1 of the function was introduced mainly to directly compare the results of the previous greedy algorithm with the new results obtained by other approaches. However, the function has several properties which may be not desirable. Firstly, the value of *cost* potentially decreases as the set of candidate queries Q grows. On the other hand, it grows with the "breadth" of the original query q_0, independently on quality of its decomposition. In our opinion, it would be preferable, that any factor depends only on the quality of particular solution Q_d and is not very dependent on Q or $q0$. In particular, if *cost* measures the scatter of documents it should somehow relate to the original scatter of the query q_0. Secondly, in contrast to other 3 factors, *iqover* is not upper bounded. Furthermore, its lowest value is 1, while for the other factors the lowest value is 0. Also because of this, even after normalisation of the sum of the parameters, the value of the objective function is not upper-bounded. Finally, *uncover* does not take into account the importance weights of the documents in D_0 which are present in problem specification. The above issues are addressed in the second variant of the objective function which is described below.

4.2 Variant 2

Here we address the issues of Variant 1 in the same order as they were presented.

1. *cost* - to make this value independent on the size of Q the denominator should be different than in variant 1. One could propose the scatter of the original query as the denominator. However in this case the value of this factor could be greater than 1. To keep the value of the factor in the range $[0, 1]$, and at the same time, to make the factor maximally focused on the quality of decomposition instead on the particular problem instance we propose the following: $cost = (\sum_{i\in D} c_i/|D|)/maxCost$, where $maxCost$ is a constant representing the maximum value of cost for any query observed in our dataset. Due to this the value of this factor will be usually much lower than 1 but will always be in $[0, 1]$. The numerator is the average cost in Q_d which makes it possible to compare the value of this factor for two different problem instances, which was not possible in variant 1.

2. *redfrac* - the same as in variant 1. One could consider making this factor growing as importance weights of red documents grow, however, we considered that introducing new important documents to the union does not seem to be a bad property of the decomposition.

3. *iqover* - to fit the value of this factor into the range $[0, 1]$ we propose: $iqover = (iqoverOld - 1)/|D|$,where iqoverOld denotes the corresponding factor in variant 1. Notice however, that now the factor never can reach the value of 1, which does not seem to be a big problem.

4. *uncover* - to take into account the notion of importance weights we propose:

$$uncover = \frac{\sum_{1\leq i\leq |D_0|:d_i\in(D_0\setminus U)} w_i}{\sum_{1\leq i\leq |D_0|:d_i\in D_0} w_i},$$

   i.e. the ratio of weights of documents from D_0 that were uncovered by the decomposition to the total weight of documents in D_0

Finally, Variant 2 is defined as (similarly to Variant 1):

$$of_2(Q_d) = \lambda_1 cost + \lambda_2 redfrac + \lambda_3 iqover + \lambda_4 uncover.$$

4.2.1 Discussion

The variant 2 of the objective function, in our opinion, better than variant 1, reflects some intuitive properties and, in addition, has nice mathematical properties: each of the 4 factors is now in the $[0, 1]$ range and, hence, their convex combination too. Thus, the function value is always in the interval $[0, 1]$ where the value of 0 means "ideal" solution and the value of 1 represents a very bad solution.

5. SIMULATED ANNEALING

After introducing the objective function which maps each potential solution Q_d of a given query decomposition problem to a real number in $[0, 1]$ the problem can be formulated as follows: find any subset Q_d^* of Q that minimises the objective function: $Q_d^* = \arg\min_{Q_d\subset Q}(\mathrm{obj}_i(Q_d))$, for $i \in \{1, 2\}$. Although it deserves for a separate investigation whether the decision-version of this problem is NP-hard, the authors do not know any fast (polynomial in $|Q|$) method of finding the optimum subset Q_d in general case, and it is not unlikely that there is no such a method known, since the problem is similar to the weighted set-covering problem (which is NP-hard). Hence, it seems to be justified to apply to this problem any sub-optimal meta-heuristic for solving combinatorial optimisation problems. In this section we describe application of *simulated annealing* to our problem.

Simulated annealing [6] is an example of neighbourhood-based combinatorial optimisation sub-optimal meta-heuristic. Its name comes from a method in metallurgy for obtaining metal with a very regular crystal structure by allowing for a very slow cooling, which in turn allows the molecules to find a lower energy configuration. Although the analogy is rather far, this idea found application in combinatorial optimisation under the same name. The heuristic is repeatedly reported to be relatively successful, especially when taking into account its simplicity. While it does not guarantee finding a global optimum it usually outperforms most simple greedy sub-optimal solutions.

5.1 Neighbourhood Relation

Simulated annealing is a randomised algorithm for searching the space of potential solutions using a notion of *neighbourhood relation*. Assume that S is the set of all potential solutions to the problem and $f : S \to R$ is the objective function to be minimised. A neighbourhood relation is a binary relation $N \subseteq S \times S$ with some desired properties described below. The interpretation of $N(s_1, s_2)$ is that solution s_1 is a neighbour of solution s_2 in the search space of all solutions S. A neighbour-based heuristic proceeds in steps. It starts searching at some initial solution s_0 and in each step moves from the current solution to some neighbour (specified by the relation) according to some rules specific to the heuristic. The performance can strongly depend on appropriate choice of neighbourhood relation. Let's assume that variable n represents the size of the combinatorial problem to be solved. In our case it is the number of candidate queries: $n = |Q|$. A well designed neighbourhood relation should satisfy the following properties.

1. it should be (strongly) connected, to make it potentially possible to get to the optimum from any starting solution by moving from neighbour to neighbour.

2. for any solution $s \in S$ the size of its neighbourhood should be bounded by a (fixed) polynomial of n (i.e. $|\{s' \in S : N(s, s')\}| = O(P(n))$, where $P(n)$ is a polynomial of n). This property guarantees that each step, which involves choosing the neighbour to move to out of all neighbours, will have acceptable time complexity.

3. the diameter of the (directed) graph $G(S, N)$ should be bounded by a polynomial of n. This property guarantees that the total number of steps (getting to the best solution by moving from neighbour to neighbour) can be acceptably low.

4. the neighbourhood should be defined so that the values of objective functions of any two neighbours should differ as little as possible. This property is essential for searching for the optimum solution by making local, greedy-alike moves in the solution space.

We next describe how we define the neighbourhood relation for TQD. Actually, despite quite complex specification of the TQD problem, the goal is to find an optimum *subset* (Q_d in our case) of some given set of items (Q), which is a very common case in combinatorial optimisation. We define the neighbourhood relation as follows. Any two solutions $s, s' \subseteq S$ are neighbours $N(s, s')$ iff the following property is satisfied: $(s \neq s') \wedge ((\exists_{a \in s} s = s' \cup \{a\}) \vee (\exists_{b \in s'} s' = s \cup \{b\}))$, i.e. two solution-subsets are neighbours iff they differ by exactly one item (candidate query, in our case).

It is straightforward to check that such defined neighbourhood relation has all the mentioned desired properties:

1. given $S_1, S_2 \subseteq Q$ it is possible to move from S_1 to S_2 by exchanging all the elements in $S_1 \setminus S_2$ with all the elements in $S_2 \setminus S_1$;

2. for any solution $s \in S$ the number of its neighbours is exactly $n = |D|$. It is easily seen by the notion of *characteristic vector* – a binary vector of size n representing any subset. It has value of 0 on position $i \in [1, n]$ iff the i-th item belongs to the subset and value of 0 if it does not. Moving from a solution to a solution is equivalent to switching any of the n bits of the vector to its opposite value;

3. the diameter of the relation is exactly n – the most distant subsets are the complements to each other and it needs to change all the n bits to get from a subset to its complement;

4. the difference between values of objective function of any two neighbours is little, since the neighbours differ only by a single element.

```
SIMULATEDANNEALING($MaxIter, Gap$)
 1  $s \leftarrow$ RANDOMINITIALSOLUTION()
 2  $best \leftarrow s; step \leftarrow 0$
 3  $change \leftarrow step; t \leftarrow 1$
 4  while ($step < MaxIter \wedge step - change < Gap$)
 5  do
 6      $step = step + 1$
 7      $s' =$ RANDOMNEIGHBOUR($s$)
 8      if RANDOM$(0, 1) < \mathrm{P}(s, s', t)$
 9         then $s = s'$
10      if $f(s) < f(best)$
11         then $best = s; change = step$
12
13      $t = 1/\sqrt{step}$
14
15  return $best$
```

Figure 1: The version of simulated annealing algorithm used for optimising the query decomposition.

5.2 The Algorithm

We adapted the simulated annealing algorithm to our problem of optimising the query decomposition problem. The variant which was used in most of our experiments is shown on the figure 1. The idea of the algorithm is as follows. It starts with a random solution. Variable *best* keeps the best solution found, *step* counts the iterations of the main loop and *change* remembers the last iteration in which the algorithm improved the best solution. The main loop proceeds until the number of iterations exceeds $MaxIter$ or there was no improvement for the last Gap iterations. After some exploratory experimentation we set $MaxIter = 100,000$ and $Gap = 10,000$. In each iteration, the algorithm picks a random neighbour of the current solution s (line 9). The key idea in the simulated annealing algorithm is the function $P(s, s', t) : S \times S \times R \rightarrow [0, 1]$ which specifies the *probability* of accepting the move from solution s to a neighbour solution s', which also depends on so called *temperature* (t). The function P should satisfy the following conditions:

1. $P(s, s', t) = 1$ if solution s' is *better* then s in terms of the cost function f (i.e. $f(s') < f(s)$ in a minimisation problem);

2. if s' is worse than s the value of P is positive (i.e. it allows for moving to a worse solution), but decreases with $|f(s) - f(s')|$;

3. for fixed s and s', when s' is worse than s the value of P *decreases* with time and tends to 0.[1]

The function P used in our algorithm is given by the following formula, which is a common choice in simulated annealing: $P(s, s', t) = e^{-\frac{|f(s) - f(s')|}{t}}$ where t is the temperature. It is straightforward to check, that it satisfies all the conditions mentioned above. The temperature in our algorithm, after some exploratory experimentation, was chosen to be the following function of iteration (step): $t(step) = 1/\sqrt{step}$, and is updated in line 15 of the algorithm.

The best solution is updated in line 13.

[1]More precisely, it increases with temperature t and the temperature is a decreasing function of time (iteration)

	Variant 1		Variant 2	
	Grd	SA	Grd	SA
Min	0.005	0	0	0
Max	2.982	1	0.981	0.957
Avg	0.790	0.517	0.291	0.232
Won	11%	89%	24.4%	75.6%

Table 1: Experimental results.

6. EXPERIMENTAL RESULTS

The dataset we used comes from an in-house query log from early 2008. We sampled uniformly at random a set of 100 queries $Q' \subset Q$ out of the top 10,000 queries made by users. For each of the original 100 queries $q_i \in Q'$, we collected all the URLs seen by users that issued that query, $docs(q_i) \subseteq D$ where D is the set of all the documents in the search engine, and $d_j \in docs(q_i)$ if there was a user who issued query q_i and then was shown by the search engine the document d_j among the top 100 results for that query. Note that click information is not taken into account. The sizes of the sets $docs(q_i)$ vary between 10 and 472 in the sample, with a median of 45 URLs. Next given the original query q_i and its set of seen URLs $docs(q_i)$, we took the all the candidate queries in Q having at least two queries overlapping with the set of results seen by users: q_j is a candidate for query q_i iff $\exists d_k, d_\ell \in D \; s.t. \; d_k \neq d_\ell \wedge d_k \in docs(q_i) \wedge d_\ell \in docs(q_i) \wedge d_k \in docs(q_j) \wedge d_\ell \in docs(q_j)$. If there were more than 100 queries with an overlap of 2 documents or more, we picked only the top 100 by overlap, breaking ties arbitrarily. The sizes of the candidate sets vary between 0 (for 4 queries), and 100 which was the maximum, with a median of 27 candidate queries.

We compared the greedy and the simulated annealing methods on all the 96 non-empty queries, with 39 different combinations of $\langle \lambda_1, \lambda_2, \lambda_3, \lambda_4 \rangle$ for each query, and with the two variants of the global objective function. The 39 combinations were obtained by taking all the 13 combinations of the three parameters $\langle \lambda_1, \lambda_2, \lambda_3 \rangle$ reported in [1][2], and extending each of them with the fourth parameter, set to the value of 0, 1 or 10, and normalising to sum up to 1.

Aggregated results over these 3744 runs are reported in Table 1. The results confirm that the SA method is generally (89% of the cases for Variant 1, and 75.6% of the cases for Variant 2) outperforming the previous greedy method. The breakdown by combinations of parameter (Table 2)[3] shows that the settings under which SA methods outperforms greedy more evidently are for $\lambda_1 = 10$ and/or $\lambda_3 = 0$. It can also be observed that the average value of the optimisation function (Variant 2) grows with the value of λ_2.

In our future work we plan (1) to test other heuristic methods such as genetic algorithms, (2) to analyse how far from the optimum are the results (optimum obtained by brute force, when feasible, and integer programming), and (3) to set-up a human user-evaluation.

Acknowledgements. Thanks are due to Aristides Gionis for valuable discussions and for providing the output of the greedy algorithm.

[2]The settings in [1] were: (1,0,0), (0,1,0), (0,0,1), (0,1,1), (1,1,0), (1,0,1), (1,1,1), (10,1,0), (10,0,1), (10,1,1), (1,10,0), (1,0,10), (1,10,10)

[3]In this section, we report the parameter values before normalisation, for easier readability

λ_1	λ_2	λ_3	λ_4	Min	Max	Avg	Won
0	0	1	0	0	0.22	0.02	66
0	0	1	1	0	0.48	0.13	59
0	0	1	10	0	0.88	0.20	53
0	1	0	0	0	0.94	0.45	87
0	1	0	1	0	0.92	0.44	74
0	1	0	10	0	0.95	0.27	64
0	1	1	0	0	0.53	0.30	73
0	1	1	1	0	0.61	0.31	66
0	1	1	10	0	0.87	0.25	65
1	0	0	0	0	0.32	0.09	86
1	0	0	1	0.01	0.65	0.17	76
1	0	0	10	0	0.91	0.21	75
1	0	1	0	0	0.19	0.06	75
1	0	1	1	0	0.43	0.13	70
1	0	1	10	0	0.83	0.20	66
1	0	10	0	0	0.21	0.03	67
1	0	10	1	0	0.19	0.05	63
1	0	10	10	0	0.47	0.13	61
1	1	0	0	0	0.58	0.36	77
1	1	0	1	0.06	0.66	0.35	70
1	1	0	10	0.03	0.89	0.26	63
1	1	1	0	0	0.42	0.27	71
1	1	1	1	0.04	0.50	0.28	66
1	1	1	10	0.02	0.82	0.24	61
1	10	0	0	0	0.88	0.45	85
1	10	0	1	0.01	0.84	0.47	83
1	10	0	10	0.01	0.89	0.42	74
1	10	10	0	0	0.52	0.31	71
1	10	10	1	0.01	0.50	0.31	71
1	10	10	10	0.01	0.60	0.31	67
10	0	1	0	0	0.29	0.09	86
10	0	1	1	0.02	0.35	0.11	84
10	0	1	10	0.01	0.61	0.17	81
10	1	0	0	0	0.36	0.15	82
10	1	0	1	0.06	0.41	0.17	79
10	1	0	10	0.03	0.65	0.20	73
10	1	1	0	0	0.33	0.14	81
10	1	1	1	0.06	0.38	0.16	81
10	1	1	10	0.03	0.62	0.20	76

Table 2: Performance of SA on Variant 2, breakdown by the 39 different combinations of $\langle \lambda_1, \lambda_2, \lambda_3, \lambda_4 \rangle$. (Max. Won is 96)

7. REFERENCES

[1] Francesco Bonchi, Carlos Castillo, Debora Donato, and Aristides Gionis. Topical query decomposition. In *Proceeding of ACM SIGKDD'08*.

[2] Robert D. Carr, Srinivas Doddi, Goran Konjevod, and Madhav V. Marathe. On the red-blue set cover problem. In *Symposium on Discrete Algorithms*, 2000.

[3] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[4] Bernard J. Jansen, Amanda Spink How are we searching the World Wide Web? A comparison of nine search engine transaction logs *Information Processing & Management. Formal Methods for Information Retrieval.*, 1(42):248–263, 2006.

[5] David Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the ACM Symposium on Theory of Computing*, 1973.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[7] David Peleg. Approximation algorithms for the label-covermax and red-blue set cover problems. *Journal of Discrete Algorithms*, 5(1):55–64, 2007.