Adaptive Re-Ranking as an Information-Seeking Agent

Sean MacAvaney[1], Nicola Tonellotto[2] and Craig Macdonald[1]

[1]*University of Glasgow, United Kingdom*
[2]*University of Pisa, Italy*

Abstract

Re-ranking systems are typically limited by the recall of the initial retrieval function. A recent work proposed *adaptive re-ranking*, which modifies the re-ranking loop to progressively prioritise documents likely to receive high scores based on the highest scoring ones thus far. The original work framed this process as an incarnation of the well-established *clustering hypothesis*. In this work, we argue that the approach can also be framed as an information-seeking agent. From this perspective, we explore several variations of the graph-based adaptive re-ranking algorithm and find that there is substantial room for improvement by modifying the agent. However, the agents that we explore are more sensitive to the new parameters they introduce than the simple-yet-effective approach proposed in the original adaptive re-ranking work.

Keywords

Neural re-ranking, Clustering hypothesis, Retrieval agents

1. Introduction

Neural techniques leveraging use pre-trained language models (PLMs) have been shown to be effective at various information retrieval (IR) ranking tasks, such as question answering and ad-hoc document ranking [1]. The most effective deployment of PLMs in neural IR is through a cascading architecture. The cascade is composed of a two-stage pipeline. The role of the first stage is to retrieve a candidate pool of documents from the collection using an inexpensive scoring functions such as BM25, while the second stage re-ranks the documents in the first stage sample using an expensive PLM. Typically, the first stage focuses on optimising the recall of the pool, i.e., the number of relevant documents in the pool, and the second stage optimises precision-oriented metrics such as NDCG@10.

A major limitation of this approach lies in the performance of the first stage. If the first stage fails to retrieve a relevant document, the second stage has no chance to re-rank it. Consequently, many techniques have been designed to resolve this first stage "recall" problem, such as query rewriting [2, 3] and dense retrieval [4, 5, 6, 7]. Recently, MacAvaney et al. [8] proposed a complementary approach for overcoming the first stage recall problem by enabling re-rankers to identify and score documents missing from the initial ranker. This is accomplished by using

First Workshop on Proactive and Agent-Supported Information Retrieval, 2022
sean.macavaney@glasgow.ac.uk (S. MacAvaney); nicola.tonellotto@unipi.it (N. Tonellotto); craig.macdonald@glasgow.ac.uk (C. Macdonald)

the incremental nature of re-ranking documents, typically done in batches to make the best use of specialised hardware like GPUs and TPUs. After each batch, information is gained about which documents get the highest relevance score, which can be leveraged to make a more informed decision about which documents to rank next (a process the authors call *Adaptive Re-Ranking*). By making use of the *clustering hypothesis* [9, 10, 11] and pre-computed similarity scores between documents (via a *corpus graph*), documents that are similar to the highest-ranked documents can be identified and prioritised, thereby allowing documents missed in the first stage to be retrieved. This particular formulation of Adaptive Re-Ranking is referred to as *Graph-based Adaptive Re-ranking* (GAR). The variant of GAR proposed by MacAvaney et al. [8] used a simple strategy for balancing (a) the scoring of batches of existing candidate documents (identified by the first stage retriever) and (b) discovering new candidates (by scoring documents similar to the top ranked results so far). Their approach simply alternates between (a) and (b), batch by batch. We refer to this strategy as `Alternate`.

We argue that the approach for re-ranking can be thought of as an information-seeking agent and explained in terms of user browsing strategies. For instance, an agent using the traditional re-ranking approach simply scans down a list of search results looking for items that are relevant to its information need. Meanwhile, an agent using the `Alternate` Adaptive approach alternates between exploring documents from the search results and similar documents to the most relevant ones it found so far. We observe that this may not be the process by which a reasonable human user would traverse search results; for instance, a user may only look at similar documents to those that are sufficiently relevant. Therefore, in this work we explore a variety of agent-based strategies for adaptive re-ranking, inspired by behaviours that a human may take when seeking information.

In summary, we explore new agent-based strategies for the recently-proposed Graph-based Adaptive Re-Ranking approach. We establish an upper-bound performance for smarter agents and explore several other strategies. In doing so, we demonstrate the potential breadth of possible strategies that can be applied to GAR.

2. Background and Related Work

Cascading architectures are commonly deployed in information retrieval. In principle, these are driven by an efficiency-effectiveness tradeoff – the most effective models cannot be applied on all documents in the index in a timely manner [12]. Hence, cheap ranking stages (with high recall) are typically combined with more expensive ranking stages (to obtain higher precision). Ranking cascades are common in web search using learning-to-rank, where an initial retrieval stage uses a cheap ranking function, commonly BM25, to identify a candidate set of documents. This is followed by a feature computation stage, and then application of the learned model [13]. Limiting the number of documents retrieved in the first stage – which we call the *budget* in this paper – has the effect of reducing recall, but enhances overall efficiency [14]. Later, Tonellotto et al. [15] proposed a selective approach that could adjust the budget on a per-query basis for efficient & effective retrieval.

More recently, multi-stage ranking has been commonly used for applying neural re-ranking models. For instance, *cross-encoder* neural models [16] must be applied jointly on the query

and the document to compute a relevance score. Since the cross-encoder scoring function can be computationally expensive, re-ranking is often limited to a predefined maximum number documents that the system is willing to re-rank for each query (i.e., a *re-ranking budget*, such as 1000). Hence, the application of a first (so-called sparse) retriever to obtain a candidate set is necessary for such models. However, this limits the recall of the candidate set, in that relevant documents not matching the query terms (and which may obtain a high score from the re-ranking cross-encoder) will not be retrieved.

To address this gap, several recent neural ranking models of have been used proposed: (i) dense retrieval models based on bi-encoders and (ii) learned sparse models.

In bi-encoder-based dense retrieval models [4, 5, 6, 7], the documents are pre-encoded into embedded representations, and nearest neighbour lookup is performed based on an embedded representation of the query [17]. However, online nearest neighbour search is more expensive than use of an inverted index-based sparse retrieval, hence inverted indexes are adopted for first-stage retrieval. Of note, ColBERT [4] uses a multi-stage dense retrieval architecture – an approximate nearest neighbour search, followed by an exact neural scoring.

Learned sparse models encompass the use of the PLMs upon the document representation to make a refined representation that can be indexed by existing sparse inverted indexing tools. For instance, doc2query [18, 19] enriches the sparse document representation with possible queries that the document could be retrieved for. DeepCT [20], uniCOIL [21, 22], DeepImpact [23, 24] and SPLADE [25, 26] all use a PLM to generate a new sparse representation of the document.

This work focuses on a recently-proposed complementary approach: Graph-Based Adaptive Re-Ranking (GAR) [8]. Rather than limiting the candidate pool of documents for re-ranking to those from the first stage, GAR augments the re-ranking loop to identify documents that are similar to the highest-scoring ones identified so far. In the following sections, we cover this process in more detail (Section 3) and identify that this process can be seen as an information-seeking agent (Section 4). We use this formulation to propose alternative adaptive re-ranking agents and compare their effectiveness (Section 6).

3. Graph-based Adaptive Re-Ranking

GAR is formulated as a re-ranking process. In a re-ranking process, the documents from an initial candidate set, R_0, are re-ranked by a scoring function, Score(), to obtain a final ranking of R_1 documents. The scoring function can be expensive to apply, for instance a cross-encoder neural model such as monoT5. This means that efficiency constraints may limit the number of applications of Score() that can be undertaken, which is called the re-ranking *budget*. In practice, batches of documents are scored at the same time, making effective use of GPU or TPU hardware.

A typical is non-adaptive re-ranking is shown in Algorithm 1, where an initial ranking (prioritised by rank) is scored in batches until the re-ranking budget is fully consumed.

In contrast, GAR proposed an *adaptive* approach to re-ranking, where highly scored documents are used as seeds for nearest neighbour retrieval, rather than continuing down the ranking. To perform this efficiently, a graph of document-document similarity (G) is created at index time, which records the nearest documents for each given document. Since this is precomputed

Algorithm 1 Non-Adaptive Re-Ranking

Input: Initial ranking R_0, batch size b, budget c
Output: Re-Ranked pool R_1

$R_1 \leftarrow \emptyset$
do
  $B \leftarrow$ Score(top b documents from R_0, subject to c)
  $R_1 \leftarrow R_1 \cup B$
  $R_0 \leftarrow R_0 \setminus B$
while $|R_1| < c$

and small, it can be used efficiently at ranking time to identify more documents to score.

In particular, after each batch B of documents is scored, the nearest documents to those in the batch are added to a "frontier" set of documents F. The frontier is prioritised by the scores of the re-ranked documents, so the neighbours of the highest-scored documents are explored first. The process then alternates between scoring the top documents from the initial ranking R_0 and scoring the top documents in the frontier F. Algorithm 2 shows this process.

We call this `Alternate`, due to the manner in which it alternates between the initial ranking and the frontier F. In the following, we link GAR with intelligent agents, and shows us how this allows further agent-based implementations.

4. Re-Ranking Agents

We argue that GAR re-ranking approaches can be interpreted as information-seeking agents, which peruse retrieved documents, and decide how to act based on the relevance of the documents found (such actions could be selecting more documents from the candidate set, or selecting similar documents to those already scored highly). In doing so, they act like a user who was deciding how to progress through a ranked list.

Algorithm 2 `Alternate` Re-Ranking [8]

Input: Initial ranking R_0, batch size b, budget c, corpus graph G
Output: Re-Ranked pool R_1

$R_1 \leftarrow \emptyset$
$P \leftarrow R_0$
$F \leftarrow \emptyset$
do
  $B \leftarrow$ Score(top b documents from P, subject to c)
  $R_1 \leftarrow R_1 \cup B$
  $R_0 \leftarrow R_0 \setminus B$
  $F \leftarrow F \setminus B$
  $F \leftarrow F \cup (\text{Neighbours}(B, G) \setminus R_1)$
  $P \leftarrow \begin{cases} R_0 & \text{if } P = F \\ F & \text{if } P = R_0 \end{cases}$
while $|R_1| < c$

We treat existing re-ranking strategies as information-seeking agents, and in the following we introduce new alternative agents. In all cases, we have the following: an *environment* which consists of an initial ranking of candidate documents, denoted as R_0, a graph of document-document similarities G, and the final retrieved documents list R_1; and a set of available *actions*, namely scoring documents from R_0 and placing in R_1, and scoring documents from G and placing into R_1. Some agents may also maintain a new *state*, such as a prioritised frontier F of documents collected from the graph. We argue that such a fully-capable agent can be classified as a *model-based reflex agent* within the classes of [27]. Indeed, a scoring model (e.g., monoT5) estimates relevance, which is only partially observable. It can choose actions based on the state, i.e., the current frontier.

In the following, we list four possible agents based on the GAR formulation.

4.1. `TwoPhase`

A simple alternative to `Alternate` is to operate in two sequential phases. In the first phase, all documents up to rank $k < c$ are scored. In the second phase, the remaining re-ranking budget $c - k$ is utilised by prioritising the neighbours of the highest-scoring documents. The behaviour models a user who first investigates numerous pages of search results to find the most relevant content, and then goes back to check for similar documents to the best ones identified. Algorithm 3 shows this process. We consider two variations of this `TwoPhase` strategy: one where the frontier continues to be updated based on the scores of the documents in the second phase (`TwoPhase-Refine`), and one where the frontier only considers the first phase documents (`TwoPhase-Fixed`).

4.2. `Threshold`

The `Threshold` agent only explores the corpus graph for sufficiently relevant documents. In this setting, the initial ranking pool is updated to include the neighbours of scored documents

Algorithm 3 `TwoPhase` Re-Ranking

Input: Initial ranking R_0, batch size b, budget c, corpus graph G, first phase size k
Output: Re-Ranked pool R_1

$R_1 \leftarrow \emptyset$
do
  $B \leftarrow$ Score(top b from R_0, subject to k)
  $R_1 \leftarrow R_1 \cup B$
  $R_0 \leftarrow R_0 \setminus B$
while $|R_1| < k$
$F \leftarrow$ Neighbours(R_1, G)
do
  $B \leftarrow$ Score(top b from F, subject to c)
  $R_1 \leftarrow R_1 \cup B$
  $F \leftarrow F \setminus B$
  $F \leftarrow F \cup (\text{Neighbours}(B, G) \setminus R_1)$ ▷ If "Refine" variant, otherwise skip this step
while $|R_1| < c$

Algorithm 4 `Threshold` Re-Ranking

Input: Initial ranking R_0, batch size b, budget c, corpus graph G
Output: Re-Ranked pool R_1

$R_1 \leftarrow \emptyset$
do
  $B \leftarrow$ Score(top b from R_0, subject to c)
  $R_1 \leftarrow R_1 \cup B$
  $R_0 \leftarrow R_0 \setminus B$
  $R_0 \leftarrow R_0 \cup$ (Neighbours(documents from B that satisfy r, G) $\setminus R_1$)
while $|R_1| < c$

that exceed a relevance score threshold r. By adding these documents to the pool with infinite priority, they are always scored before other documents in the pool. This agent is akin to a searcher who, as they traverse a ranked list, investigate similar documents to ones that they deem sufficiently relevant as they are encountered. Importantly, this agent considers the *absolute* relevance score of documents, rather than the *relative* ordering of retrieved documents so far, which is used by techniques like `Alternate` and `TwoPhase`. Algorithm 4 shows this process.

4.3. `Greedy`

We next explore a strategy that attempts to identify which pool of documents – the initial ranking or the frontier – is has recently given the most relevant document. We approach this as a `Greedy` strategy, by always choosing the pool with the largest maximum score from the most recent batch. This process is akin to a searcher who switches between searching through a ranked list and the documents close to the best ones they found so far based on the most relevant document they identified recently from each pool. Algorithm 5 shows this process.

Algorithm 5 `Greedy` Re-Ranking

Input: Initial ranking R_0, batch size b, budget c, corpus graph G
Output: Re-Ranked pool R_1

$R_1 \leftarrow \emptyset$
$P \leftarrow R_0$
$F \leftarrow \emptyset$
$best_{R_0} \leftarrow \infty$
$best_F \leftarrow \infty$
do
  $B \leftarrow$ Score(top b documents from P, subject to c)
  $best_P \leftarrow$MaxScore(B)
  $R_1 \leftarrow R_1 \cup B$
  $R_0 \leftarrow R_0 \setminus B$
  $F \leftarrow F \setminus B$
  $F \leftarrow F \cup$ (Neighbours$(B, G) \setminus R_1$)
  $P \leftarrow \begin{cases} R_0 & \text{if } best_{R_0} \geq best_F \\ F & \text{if } best_{R_0} < best_F \end{cases}$
while $|R_1| < c$

Algorithm 6 Adaptive Oracle Re-Ranking

Input: Initial ranking R_0, batch size b, budget c, corpus graph G, relevance assessments *qrels*
Output: Re-Ranked pool R_1

$R_1 \leftarrow \emptyset$
$F \leftarrow \emptyset$
do
  $P \leftarrow R_0$ or F, whichever will improve the ranking the most using *qrels*
  $B \leftarrow$ Score(top b from P, subject to c)
  $R_1 \leftarrow R_1 \cup B$
  $R_0 \leftarrow R_0 \setminus B$
  $F \leftarrow F \setminus B$
  $F \leftarrow F \cup (\text{Neighbours}(B, G) \setminus R_1)$
while $|R_1| < c$

4.4. Oracle

Finally, to explore whether there is further room for improvement in adaptive re-ranking, we apply an *oracle* strategy. This strategy makes use of the relevance assessments (*qrels*) to choose which batch of documents to score. At each step, both the initial ranking and the frontier are scored. Then, the algorithm proceeds by adding only the batch that improves the ranking effectiveness (e.g., through a measure like nDCG) the most at that step. Naturally, this algorithm is not useful in practice, given that it relies on ground-truth relevant knowledge.[1] However, it helps us establish whether there is further room for improvement in adaptive re-ranking agents. Algorithm 6 shows this process.

5. Experimental Setup

We conduct experiments using various adaptive re-ranking agents to answer the following research questions:

RQ1 Is there room for further improvement in adaptive re-ranking strategies?
RQ2 Can smarter adaptive re-ranking strategies outperform the `Alternate` approach?
RQ3 How sensitive are alternative adaptive re-ranking strategies to their parameters?

We conduct experiments using the TREC Deep Learning 2019 and 2020 passage ranking datasets [28, 29]. We use 2019 as our "development" data, leaving 2020 as a held-out test set. For both datasets, we report nDCG and Recall@1000 (with a minimum relevance threshold of 2) performance. We fix our study to a MonoT5 (base) scoring function [30], a re-ranking budget $c = 1000$, and a batch size $b = 16$. We explore four initial ranking functions: BM25, TCT-ColBERT [31, 32], Doc2Query [19], and SPLADE++ (`CoCondenser-EnsembleDistil` version) [26].

[1] Of course, a perfect agent would ignore the scoring function entirely and just return documents by their human assessments to get a perfect ranking. We limit the usage of the assessments for this agent to the decision of which batch to score to help us identify whether better decisions about which batches to score can further improve adaptive re-ranking effectiveness.

		TREC DL 2019 (dev)				TREC DL 2020 (test)			
		GAR_{bm25}		GAR_{tct}		GAR_{bm25}		GAR_{tct}	
Pipeline	Agent	nDCG	R@1k	nDCG	R@1k	nDCG	R@1k	nDCG	R@1k
BM25»MonoT5	Non-Adaptive	0.699	0.755	0.699	0.755	0.711	0.805	0.711	0.805
	Oracle	0.747	0.804	0.786	0.853	0.748	0.791	0.768	0.828
	`Alternate`	0.726	^{N}0.827	^{NO}0.743	^{N}0.839	^{N}0.743	^{NO}**0.874**	^{N}**0.749**	^{N}**0.892**
	`TwoPhase-Fixed`	^{N}0.729	^{N}0.815	^{NO}0.740	^{N}0.836	^{N}0.732	^{NA}0.838	^{N}0.742	^{NA}0.858
	`TwoPhase-Refine`	^{N}0.741	^{N}0.826	^{NO}0.743	^{N}0.841	^{N}0.743	^{NO}0.871	^{NA}0.744	^{NA}0.879
	`Threshold`	^{N}**0.742**	^{N}**0.829**	^{NOA}**0.751**	^{N}**0.849**	^{N}**0.744**	^{NO}**0.874**	^{N}0.744	^{NA}0.874
	`Greedy`	0.723	^{N}0.823	^{NO}0.737	^{N}0.839	^{N}0.743	^{NO}0.868	^{N}0.744	^{N}0.882
TCT»MonoT5	Non-Adaptive	0.704	0.830	0.704	0.830	0.693	0.848	0.693	0.848
	Oracle	0.793	0.891	0.766	0.846	0.762	0.874	0.754	0.861
	`Alternate`	^{NO}**0.733**	^{N}0.883	^{NO}0.724	^{N}0.866	^{NO}**0.719**	^{N}0.881	^{NO}**0.710**	^{N}**0.871**
	`TwoPhase-Fixed`	^{NO}**0.733**	^{N}0.874	^{NO}0.719	^{N}0.857	^{NO}0.717	^{N}0.877	^{NO}**0.710**	^{N}0.868
	`TwoPhase-Refine`	^{NO}**0.733**	^{N}0.882	^{NO}0.722	0.859	^{NO}**0.719**	^{N}**0.883**	^{NOA}0.707	^{A}0.866
	`Threshold`	^{NO}0.731	^{N}**0.886**	^{NO}0.720	^{N}0.866	^{NOA}0.711	0.871	^{NOA}0.705	0.862
	`Greedy`	^{NO}0.731	^{N}0.881	^{NO}**0.725**	^{N}**0.871**	^{NOA}0.713	^{N}0.873	^{NO}0.708	^{N}0.868
D2Q»MonoT5	Non-Adaptive	0.747	0.830	0.747	0.830	0.731	0.839	0.731	0.839
	Oracle	0.797	0.867	0.798	0.867	0.791	0.884	0.793	0.889
	`Alternate`	0.757	^{N}**0.880**	^{NO}0.766	^{N}**0.879**	^{NO}**0.748**	^{N}**0.880**	^{O}**0.748**	^{N}**0.895**
	`TwoPhase-Fixed`	^{N}0.765	^{N}0.866	^{NO}0.765	^{N}0.870	^{NO}**0.748**	^{NA}0.867	^{O}0.745	^{NA}0.870
	`TwoPhase-Refine`	^{N}**0.769**	^{N}0.875	^{N}**0.767**	^{N}0.878	^{NO}**0.748**	^{N}0.877	^{O}0.747	^{N}0.892
	`Threshold`	^{N}0.766	^{N}0.876	^{NO}**0.767**	^{N}0.877	^{O}0.746	^{NA}0.874	^{O}0.745	^{N}0.881
	`Greedy`	0.754	^{N}0.874	^{O}0.757	^{N}0.873	^{O}0.744	^{N}0.878	^{O}**0.748**	^{N}0.894
SPLADE»MonoT5	Non-Adaptive	0.737	0.872	0.737	0.872	0.731	0.899	0.731	0.899
	Oracle	0.807	0.898	0.783	0.859	0.777	0.886	0.781	0.899
	`Alternate`	^{O}0.745	0.893	^{O}0.737	0.875	^{O}0.737	**0.909**	^{O}0.734	**0.908**
	`TwoPhase-Fixed`	^{O}0.763	0.863	**0.764**	0.869	**0.748**	^{A}0.868	^{O}0.742	^{NA}0.867
	`TwoPhase-Refine`	^{O}**0.769**	0.875	**0.764**	0.870	^{O}**0.748**	0.877	^{O}0.736	^{A}0.869
	`Threshold`	^{O}0.766	0.871	0.759	0.857	^{O}0.746	0.874	^{O}**0.744**	^{NA}0.865
	`Greedy`	^{NO}0.747	^{N}**0.895**	^{O}0.740	**0.882**	^{O}0.734	0.903	^{O}0.734	0.906

Table 1
Re-ranking performance on TREC Deep Learning 2019 and 2020 using various agents. The best-performing (non-oracle) agent in section is listed in bold. Significant differences compared to the Non-Adaptive, Oracle, and `Adaptive` systems are marked with ^{NOA}, respectively (paired t-test, $p < 0.05$).

Following the original `GAR` paper, we use two corpus graphs, based on BM25 similarity and TCT-ColBERT.

As we will show in Section 6, `TwoPhase` and `Threshold` are rather sensitive to their parameters, so the results we present use the parameters that yield the highest nDCG performance on TREC DL 2019. For `TwoPhase`, we test first-stage cutoff k values ranging from 100 to 900 (by 100). For `Threshold`, we first identify the distribution of scores using the MS MARCO dev (small) dataset, re-ranking 100 BM25 results. We then pick threshold r values based on percentiles from this distribution, from 0.02[2] to 0.20 (by 0.02).[3]

6. Results and Analysis

Table 1 presents the main results of our study.

[2]Here, 0.02 represents the top 2% of scores.
[3]This range was chosen after an initial study of orders of magnitude from 0.001 to 0.3 identified it as the most promising.

We first explore whether there exists room for improvement by exploring alternative agents (RQ1). In all cases, the Oracle agent, which has knowledge of the relevance assessments when choosing which batches of documents to score, has a higher nDCG performance than all other agents. The difference in performance can often be considerable; for instance, the best-performing non-oracle agent over a the TCT pipeline using a BM25 graph has an nDCG of 0.733, while the oracle achieves an nDCG of 0.793. Differences between non-oracle agents and the Oracle are not always significant, however. We note that while nDCG is often improved using either oracle technique, recall often suffers compared to alternative agents. Perhaps this is unsurprising, given that the oracles aim to directly optimise nDCG (recall is only considered insofar as it would improve the nDCG). Significant improvements compared to the Oracle are found more frequently when measuring nDCG using TCT corpus graph (GAR_{tct}) than the BM25 graph (GAR_{bm25}).

In summary, to answer RQ1, by exploring an oracle agent, we find that there is considerable room for improvement in adaptive re-ranking. In particular, exploration of the semantic graph (GAR_{tct}) by non-oracle agents is likely sub-optimal and has room for considerable improvement.

Next, we look to answer RQ2 and RQ3. We find that each of the proposed alternative agents to be the most effective in at least one measurement. In one case, an alternative agent significantly outperforms the existing `Alternate` agent (`Threshold` when re-ranking BM25 results using GAR_{tct}). Further, there are 11 cases where alternative strategies outperform the non-adaptive, while `Alternate` does not. However, there are sometimes significant reductions in effectiveness when using the alternative agents (18 cases, all of which on the held-out DL20 data). There is also no clear pattern in which strategy is generally most effective. These results answer RQ2: while some strategies can outperform `Alternate`, we do not find a clearly superior strategy among the agents we test.

Given that significant reductions are present when using the held-out DL20 data, we dive deeper into the performance of `TwoPhase` and `Threshold` by plotting their performance for the parameters they introduce (`TwoPhase` adds the first stage cutoff k and `Threshold` adds the relevance threshold r). Figure 1 present the nDCG performance. We see that in all cases, performance varies considerably as the parameters are adjusted. Further, the optimal parameters for DL19 are often different than the optimal parameters for DL20, suggesting that the alternative agents are not very robust to the selection parameter selection. This is in particular contrast with the robustness of the `Alternate` approach to parameters, shown by MacAvaney et al. [8]. These results answer RQ3: `TwoPhase` and `Threshold` introduce parameters that are more sensitive than those present in the `Alternate` strategy.

7. Conclusions & Outlook

In this work we investigated the recently-proposed Adaptive Re-Ranking process from the perspective of information-seeking agents. We find that there is considerable room to improve adaptive re-ranking by changing the agent strategy. We further proposed several alternative strategies, and found that although they are promising, they are more sensitive to their parameters than the simple `Alternate` approach. This work sets the stage for future work that investigates additional agent strategies, such as those inspired by multi-armed bandit models.

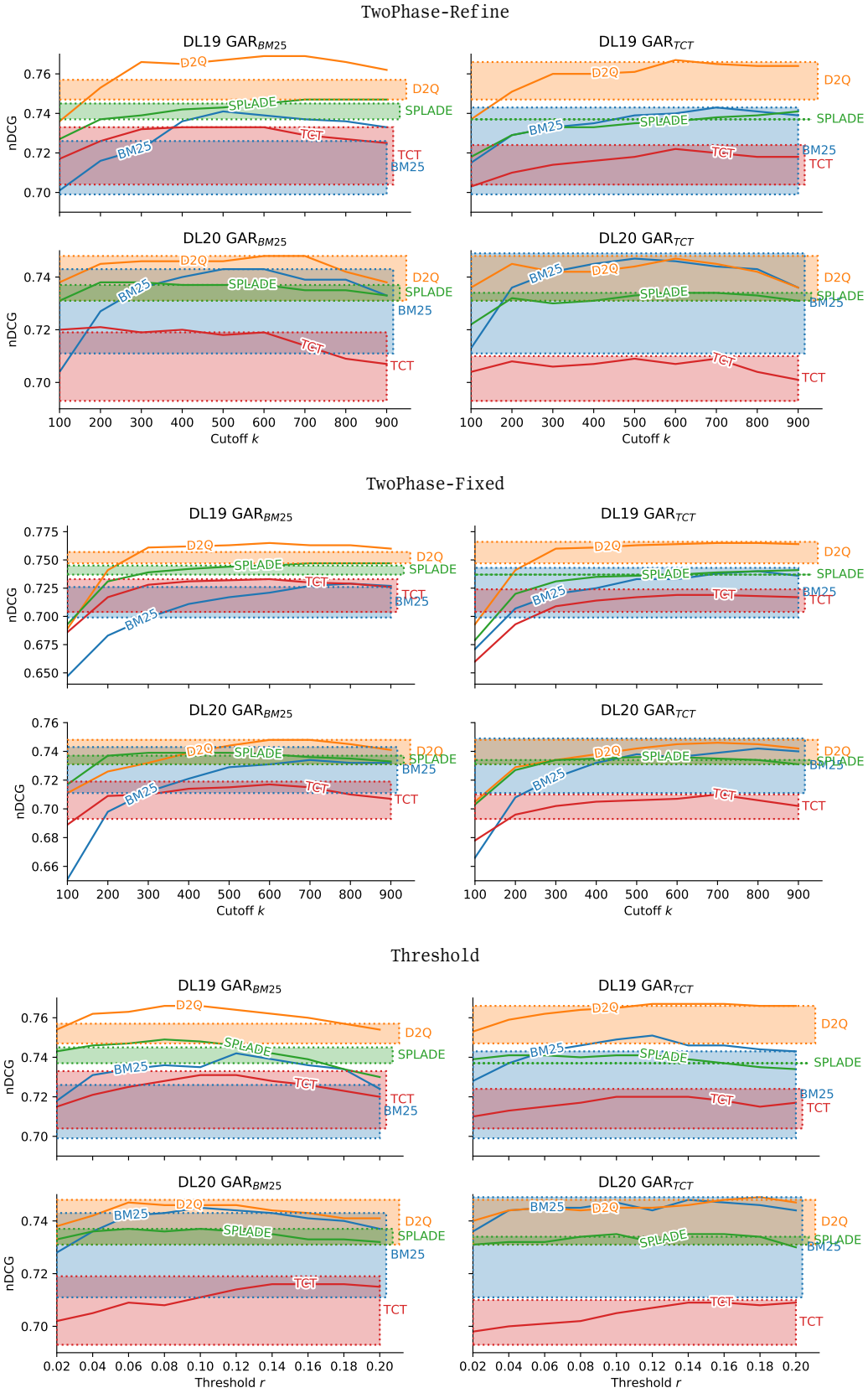

Figure 1: Performance of various proposed agents for various initial rankers and parameter values on TREC DL 2019 and 2020. The upper limit of each shaded area represents the `Alternate` strategy, and the lower limit represents the non-adaptive re-ranking performance.

Acknowledgements

Sean and Craig acknowledge EPSRC grant EP/R018634/1: Closed-Loop Data Science for Complex, Computationally- & Data-Intensive Analytics.

References

[1] J. Lin, R. Nogueira, A. Yates, Pretrained Transformers for Text Ranking: BERT and Beyond, Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers, 2021.
[2] R. Xiao, J. Ji, B. Cui, H. Tang, W. Ou, Y. Xiao, J. Tan, X. Ju, Weakly supervised co-training of query rewriting andsemantic matching for e-commerce, in: Proc. WSDM, 2019, p. 402–410.
[3] C. Macdonald, N. Tonellotto, I. Ounis, Efficient & effective selective query rewriting with efficiency predictions, in: Proc. SIGIR, 2017, pp. 495–504.
[4] O. Khattab, M. Zaharia, ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT, in: Proc. SIGIR, 2020.
[5] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, W.-t. Yih, Dense passage retrieval for open-domain question answering, 2020. URL: https://arxiv.org/abs/2004.04906. doi:10.48550/ARXIV.2004.04906.
[6] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. Bennett, J. Ahmed, A. Overwijk, Approximate nearest neighbor negative contrastive learning for dense text retrieval, in: Proceedings of ICLR, 2021.
[7] J. Zhan, J. Mao, Y. Liu, J. Guo, M. Zhang, S. Ma, Optimizing Dense Retrieval Model Training with Hard Negatives, in: Proc. SIGIR, 2021, pp. 1503–1512.
[8] S. MacAvaney, N. Tonellotto, C. Macdonald, Adaptive re-ranking with a corpus graph, in: 31st ACM International Conference on Information and Knowledge Management, 2022. URL: https://arxiv.org/abs/2208.08942. doi:10.1145/3511808.3557231.
[9] N. Jardine, C. J. van Rijsbergen, The use of hierarchic clustering in information retrieval, Information storage and retrieval 7 (1971).
[10] O. Kurland, The cluster hypothesis in information retrieval, in: Proc. SIGIR, 2013.
[11] E. M. Voorhees, The cluster hypothesis revisited, in: Proc. SIGIR, 1985.
[12] N. Tonellotto, C. Macdonald, I. Ounis, Efficient query processing for scalable web search, Foundations and Trends in Information Retrieval 12 (2018) 319–492.
[13] T.-Y. Liu, Learning to rank for information retrieval, Found. Trends Inf. Retr. 3 (2009) 225–331. URL: https://doi.org/10.1561/1500000016. doi:10.1561/1500000016.
[14] C. Macdonald, R. L. T. Santos, I. Ounis, The whens and hows of learning to rank for web search, Information Retrieval 16 (2012).
[15] N. Tonellotto, C. Macdonald, I. Ounis, Efficient and effective retrieval using selective pruning, in: Proc. WSDM, 2013.
[16] R. Nogueira, K. Cho, Passage Re-ranking with BERT, arXiv preprint 1901.04085 (2019).
[17] J. Johnson, M. Douze, H. Jegou, Billion-scale similarity search with gpus, IEEE Trans. Big Data 7 (2021).

[18] R. Nogueira, W. Yang, J. Lin, K. Cho, Document expansion by query prediction, Preprint: arXiv:1904.08375 (2019).
[19] R. Nogueira, J. Lin, From doc2query to docTTTTTquery, Online preprint (2019).
[20] Z. Dai, J. Callan, Context-aware sentence/passage term importance estimation for first stage retrieval, Preprint: arXiv:1910.10687 (2019).
[21] L. Gao, Z. Dai, J. Callan, COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List, in: Proc. NAACL-HLT, 2021, pp. 3030–3042.
[22] J. Lin, X. Ma, A few brief notes on DeepImpact, COIL, and a conceptual framework for information retrieval techniques, Preprint: arXiv:2106.14807 (2021).
[23] A. Mallia, O. Khattab, T. Suel, N. Tonellotto, Learning passage impacts for inverted indexes, in: Proc. SIGIR, 2021.
[24] A. Mallia, J. Mackenzie, T. Suel, N. Tonellotto, Faster Learned Sparse Retrieval with Guided Traversal, in: Proc. SIGIR, 2022, p. 5.
[25] T. Formal, B. Piwowarski, S. Clinchant, SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking, in: Proc. SIGIR, 2021, p. 2288–2292.
[26] T. Formal, C. Lassance, B. Piwowarski, S. Clinchant, Splade v2: Sparse lexical and expansion model for information retrieval, 2021.
[27] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 3 ed., Prentice Hall, 2010.
[28] N. Craswell, B. Mitra, E. Yilmaz, D. Campos, E. Voorhees, Overview of the trec 2019 deep learning track, in: TREC 2019, 2019.
[29] N. Craswell, B. Mitra, E. Yilmaz, D. Campos, Overview of the trec 2020 deep learning track, in: TREC, 2020.
[30] R. Nogueira, Z. Jiang, R. Pradeep, J. Lin, Document ranking with a pretrained sequence-to-sequence model, in: Proc. EMNLP, 2020.
[31] S.-C. Lin, J.-H. Yang, J. J. Lin, Distilling dense representations for ranking using tightly-coupled teachers, ArXiv abs/2010.11386 (2020).
[32] S.-C. Lin, J.-H. Yang, J. Lin, In-batch negatives for knowledge distillation with tightly-coupled teachers for dense retrieval, in: RepL4NLP-2021 Workshop, 2021, pp. 163–173.