We thank the reviewers for their thorough reviews, helpful comments, and for the various references and related areas they pointed us to. We will address all issues raised by the reviewers, add the missing references and clarify ambiguous paragraphs. Below we address the reviewers' main concerns.

Reply to reviewer 1 We thank the reviewer for suggesting additional applications for motivating this problem. We will put more emphasis on the alternative motivations and discuss their suggested applications.

The reviewer writes: *"Interviews in practice are used to learn more about the candidate and judge their fit for the position. In this case $v_i(c)$ is therefore noisily learned during the interview process, not when they arrive..."*

In our description of the hiring example (second paragraph of the intro), the first screening phase where candidates are retained/discarded corresponds to processing the candidates files **before** deciding who to summon for an interview. Thus, the values $v_i(c)$ are determined by the candidates application (CV, skills, reference letters, etc.) and not by the interviews. We agree that during the interview more information about the candidate is gathered, and that this information is likely affect the values $v_i(c)$. Indeed, in a more realistic model for the hiring problem the values $v_i(c)$ should be updated after the interviews and before the final recruitment is computed. In other motivating examples (such as internet advertising or job scheduling) the values $v_i(c)$ can be assumed to be given. We believe that our abstraction does capture the essence of the problem at hand, and is reasonable for a variety of applications.

Reply to reviewer 2 The reviewer writes: *"This question somewhat follows a new research topic called augmented-learning algorithms. Only in augmented-learning algorithms, there is usually also a fallback option?. I would be interested in seeing how they design an algorithm that performs quite well when the training set matches the actual hiring process, but does not perform arbitrarily bad when the training set is completely off of the actual candidates."*

Thanks for pointing out the connection and for suggesting the question. Indeed, our algorithm may fail when the thresholds computed in the learning phase are too large such that no item from the real-time data passes them. One simple fix is the following: after processing, say $n/2$, items from the real-time data, check whether the number of items retained by the threshold policy is too small than expected, and if this is the case then discard the threshold policy and apply the greedy algorithm from Theorem 1 on the remaining $n/2$ items. One can show that modifying the algorithm along these lines yields a "fallback" to the greedy oblivious setting from Theorem 1 in the (unlikely) case when the learned threshold policy is bad.

The reviewer writes: *" ... I don't see how is this the case. Both of the results are linear in k. Was is the idea to say that there is an exponential improvement in $1/\delta$?"*

We meant to refer in that sentence to the dependence on n - the total number of items: the bound in Theorem 1 is proportional to $\log(n)$ (which is tight, by Theorem 2), while in Theorem 4 this bound is proportional to $\log\log(n)$. We will clarify this sentence.

Reply to reviewer 3 Major concerns:

(1) *"...the authors considered d properties instead of one in previous research with the complex feasibility constraints. This makes the problem rather challenge and practical. However, the method the authors adopted for the d scenarios is relatively simple due to an independent assumption on the d properties, which may weaken the contribution here."*

We remark that we do not assume anything about the distribution of the d properties per item, and the joint distribution on the properties and value per item is arbitrary. (We do assume that the marginal distribution over the values is continuous, i.e., atomless). Additionally, note that our algorithm treats the d properties independently by design and that our method retains a near-optimal number of candidates (due to Theorem 5). We consider the simplicity of our solution as a strength rather than a weak contribution.

(3) *"How to set the number of candidate items, $k_1, k_2, ..., k_d$ with regards to each property in the first stages? And also, how to set the total candidate number k here?"*

The numbers of $k_1, \ldots, k_d$ and k are given as part of the problem formulation; they are not set up by the algorithm but rather are given as an input.

Minor concerns:

(2) We refer to the number of items retained by the **best** algorithm; i.e., the one that retains as few items as possible while satisfying the desired optimality guarantee. We will try to clarify.

(5) No, recall that our goal is to prove that T retains exactly the items in S and so we want to rule out the possibility that it retains an item outside S.

(6) The main purpose of this formulation is demonstrate the connection with matching problems.