APPENDIX

A PROOFS

Theorem 1. *Given a database of entities $\mathbb{D}$, two entities (t_i, t_j) where $t_i[m] \neq t_j[m]$ for a matching attribute m, t_j's ranking function f_j, the exact Shapley value algorithm (Algorithm 1) consumes $O\big(d\ 2^d\ n(d+C)\big)$ amount of time to obtain the Shapley values for all d attributes for the query PQ-NotMatch.*

Proof. Shapley value computation using Algorithm 1 hinges on formula 2 for any given attribute A_p. The formula relies on two key components to compute the Shapley value Sh_p, subset enumeration and the utility function v for each of the subsets. Calculating each of the subsets over d attributes takes a total of $d\ 2^d$ time. Computing the score for each entity involves constructing the masking function and then calling the ranking function. As there are n entities and each call to the ranking function consumes C time, the total time taken in computing the scores in $O(n(d+C))$. The k^{th} ranked item from the list can be computed in $O(n)$ time using the *median of medians* algorithm [4]. Given a set of attributes $S \neq \emptyset$, the time taken to calculate the utility function $v(S)$ is $O(n(d+C))$ As the subsets enumerated can be used to compute the values for all d attributes, the total time to compute Shapley value for all attributes amounts to $O(d\ 2^d\ n(d+C))$. □

Theorem 2. *Given a database of entities $\mathbb{D}$, two entities (t_i, t_j) where $t_i[m] \neq t_j[m]$ for a matching attribute m, t_j's ranking function f_j, the exact Shapley value algorithm (Algorithm 1) consumes $O\big(d\ 2^d\ n(d+C)\big)$ amount of time to obtain the Shapley values for all d attributes for the query PQ-Match.*

Proof. *(Sketch)* A similar proof to theorem 1 extends here to provide the time complexity of $O(d\ 2^d\ n(d+C))$. □

Theorem 3. *Given a database of entities $\mathbb{D}$, an entity t_i with the ranking function f_i, and match list l_i the exact Shapley value algorithm 1 consumes $O(d\ 2^d\ (n(d+C) + k\log(k)))$ amount of time to obtain the Shapley value of all d attributes for the query SQ-Single.*

Proof. As we have seen from Theorem 1, the Shapley value calculation is a product of two parts, with the subset generation consuming $d*2^d$ time. In each of these iterations, the utility function v needs to compute the top-k match list l_i^* for the masked attributes S. The new match list l_i^* with mask S applied can be computed in $O(n(d+C))$ time. The Jaccard similarity is then computed between l_i and l_i^*. To compute both intersection and union of l_i and l_i^*, first sort l_i and l_i^* in $O(k\log(k))$ time and then perform a linear $O(k)$ time scan to find both intersection and union. The time consumed in computation of the utility function v is $O(n(d+C) + k\log(k))$

Hence, the total time complexity using the exact Shapley value algorithm stands at $O(d\ 2^d\ (n(d+C) + k\log(k)))$ □

Theorem 4. *Given a database of entities $\mathbb{D}$ with ranking functions F and an entity t_i which is present in the match lists of a set of entities T, the exact Shapley value algorithm 1 consumes a total of $O(d\ 2^d\ n^2(d+C))$ amount of time to obtain the Shapley value for all d attributes for the query SQ-Multiple.*

Proof. As we have seen from Theorem 1, the Shapley value calculation is a product of two parts, with the subset generation part consuming $d\ 2^d$ time. As a first step, to compute the Shapley value for each subset, the utility function v needs to compute the top-k match lists $\forall_{1\leq i\leq n} l_i^*$ for the set of masked attributes S. The evaluation for n entities for one evaluation function f_i and a subset of attributes S can be computed in $O(n(d+C))$ time. The *median of medians* algorithm computes the top-k match list of an entity in $O(n)$ time. This yields a single match list l_i^* in $O(n(d+C))$. To compute T^*, all the match lists $\forall_{i=1}^n l_i^*$ need to be computed, increasing the runtime bounds by a factor of n. A simple scan of the match lists produces a set of entities whose match lists contain t_i with the masking based on the subsets. This consumes $O(kn)$ time. However, since k is bound by n, therefore kn is $O(n^2)$, and thus the factor of kn can be dropped. The time complexity for computing T^* is $O(n^2(d+C))$.

The Jaccard similarity is then computed between T and T^*. This is accomplished by computing both intersection and union of T and T^*, which can be done by sorting T and T^* in $O(n\log(n))$ time and then performing a linear time scan to find both intersection and union. The sorting step is dominated by the top-k match list computation step. Thus, the time consumed in computation of the utility function v is $O(n^2(d+C))$.

Hence, the total time complexity using the exact Shapley value algorithm 1 stands at $O(d\ 2^d\ n^2(d+C))$. □

Theorem 5. *The runtime for the randomized algorithm for PQ-NotMatch is bounded by $O(qd \times n(d+C)$.*

Proof. This is demonstrated by evaluating the runtime of the algorithm. First, it has to generate q permutations of the attributes, which can be computed in qd time. Then, for each of the q permutations, the contribution of each of d attributes must be computed. To compute whether an individual is in the match list, it is necessary to apply the masking function and to run the ranking function f_i on all n actors, each of these computations generating an evaluation function out of d attributes. The time of computing the match list, over all attributes, over all samples, is thus bounded by $O(qd \times n(d+C))$. □

Theorem 6. *The runtime for the randomized algorithm for PQ-Match is bounded by $O(qd \times n(d+C))$.*

Proof. The proof from Theorem 5 can be applied to show the same value for PQ-Match. □

Theorem 7. *The runtime for the randomized algorithm for SQ-Single is bounded by $O(qd \times (n(d+C) + k\log(k)))$.*

Proof. To compute this explanation, a Jaccard similarity must be taken between an original match list and a match list for each attribute and sample. To compute the match list, n individuals are masked over d attributes and run through an evaluation function taking C time. To compare two match lists, first the sets are sorted taking $k\log(k)$ time, and compared taking k time. This process is repeated over m samples, and for each of these samples the marginal value must be computed over d attributes. The time to compute the match lists and compare them over d attributes and m samples is accordingly $O(qd \times (n(d+C) + k\log(k)))$. □

Theorem 8. *The runtime for the randomized algorithm for SQ-Multiple is bounded by $O(qd \times n^2(d+C))$.*

Proof. The proof is the same as above with the following differences. Each of the n individuals being matched are computed against n evaluation functions instead of simply one. To find the sets to which an individual belongs, the k elements of n match lists are examined. Once this is computed, $n \log(n)$ calculations must be performed to compute the similarity between the sets in which an individual is present, but this value is dominated by n^2. Again, this process is repeated over q samples and d attributes. The total time taken is bounded by $O(qd \times n^2(d + C))$. □

B TABLES AND OTHER DETAILS

	Candiate Values	HR Function	Shapley Values
Python	1.0	0.002	0.0
R	0.0	0.005	0.09
Deep Learning	0.333	0.005	0.025
PHP	0.667	0.007	0.05
MySQL	0.667	0.007	0.075
HTML	0.667	0.007	0.035
CSS	0.0	0.005	0.085
JavaScript	0.667	0.005	0.065
AJAX	0.0	0.005	0.06
Bootstrap	0.0	0.006	0.07
MongoDB	0.0	0.005	0.045
Node.js	0.0	0.003	0.045
Reactjs	0.0	0.005	0.09
Performance_PG	0.791	0.06	-0.02
Performance_UG	0.7	0.06	0.015
Performance_12	1.0	0.06	-0.05
Performance_10	1.0	0.120	-0.05
Other Skills	['Algorithms', 'Data Structures', ...]	0.0769	0.065
Degree	Master of Science	0.0769	0.21
Stream	Computer Science	0.0769	0.05
Grad Year	2018	0.307	0.04
Current City	Bangalore	0.0769	0.005

Table 6: Candidate values, HR rankings, and PQ-NotMatch Shapley values.

C ADDITIONAL EXPERIMENT RESULTS

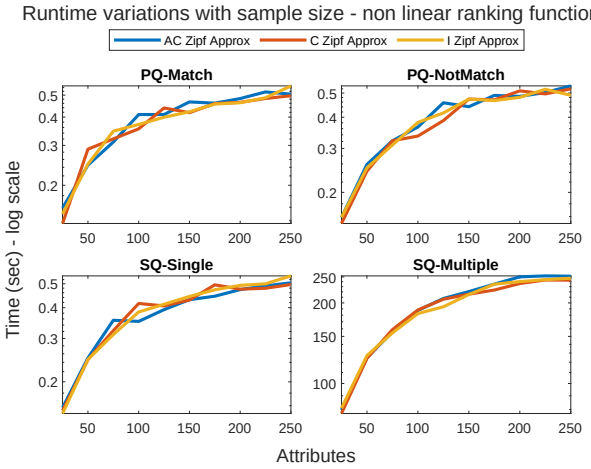

Figure 8: Runtime variations in approximate when varying number of samples for non-linear ranking functions

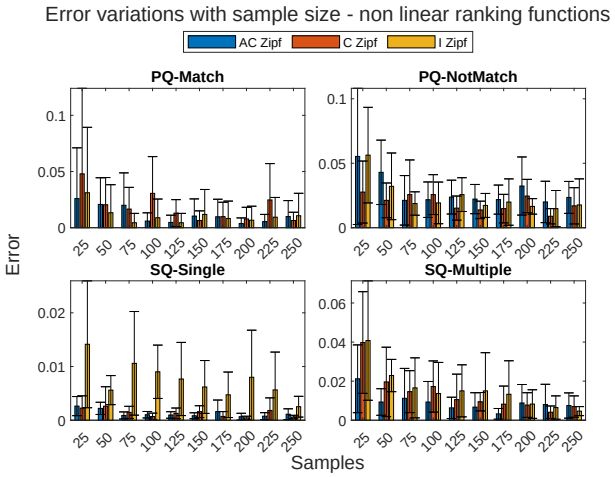

Figure 9: Error variations in approximate algorithm when varying the number of samples for non-linear ranking functions

Distribution	Correlation	Function	APX - Q1	APX - Q2	APX - Q3	APX - Q4	WT - Q1	WT - Q2	WT - Q3	SCR - Q1	SCR - Q2	SCR - Q3	SCR - Q4
Uniform	Correlated	Linear	1.0	1.0	1.0	1.0	0.9	0.7	1.0	0.2	0.6	0.1	0.1
Uniform	Correlated	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.5	0.9	0.5	.5	0.3	0.1
Uniform	Anti-Correlated	Linear	1.0	1.0	1.0	1.0	.8	0.7	1.0	0.5	0.7	0.1	0.2
Uniform	Anti-Correlated	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.4	1.0	0.1	0.4	0.3	0.1
Uniform	Independent	Linear	1.0	1.0	1.0	1.0	1.0	0.6	1.0	0.4	0.7	0.3	0.4
Uniform	Independent	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.2	0.5	0.5	0.2
Zipfian	Correlated	Linear	1.0	1.0	1.0	1.0	0.7	0.8	0.9	0.5	0.8	0.1	0.4
Zipfian	Correlated	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.2	0.6	0.2	0.2
Zipfian	Anti-Correlated	Linear	1.0	1.0	1.0	1.0	0.6	0.9	0.8	0.3	0.5	0.8	0.3
Zipfian	Anti-Correlated	Non-Linear	1.0	1.0	1.0	1.0	0.9	0.5	0.7	0.8	0.2	0.0	0.8
Zipfian	Independent	Linear	1.0	1.0	1.0	0.9	0.8	0.8	0.9	0.2	0.8	0.1	0.1
Zipfian	Independent	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.6	1.0	0.3	0.6	0.2	0.3

Table 7: The success measure of four methods in computing the same top value as Brute Force; APX=Approximate, WT=Weight, SCR=Attribute Score; and the four queries, Q1-Q4. For Q4, WT could not be used.