Aiding Software Project Staffing by Utilizing Recommendation Systems

Aleksi Asikainen

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Helsinki 31.7.2020

Supervisor

Prof. Eero Hyvönen

Advisor

Dr. Michael Samarin

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Aleksi Asikainen

Title Aiding Software Project Staffing by Utilizing Recommendation Systems

Degree programme Master's Programme in Computer, Communication and Information Sciences

Major Computer science **Code of major** SCI3042

Supervisor Prof. Eero Hyvönen

Advisor Dr. Michael Samarin

Date 31.7.2020 **Number of pages** 56+10 **Language** English

Abstract

Both staffing and recruiting have become more sophisticated during the 2000s. The amount of data and different scenarios, that needs to be processed during recruitment and staffing, has become huge thanks to the introduction of different resource and recruitment management software to companies recruitment pipelines. Multitudes of different solutions have been proposed to assist decision making in recruiting and staffing through the use of recommendation systems. They have been proven to increase the accuracy and speed of recruitment decisions with multiple different setups. One of the key points arising from previous research is the importance of underlying data used in the recommendation process as the recommendation can be only as accurate as the data is.

This thesis starts with key concepts behind the recommendation systems that can be leveraged as well as ranking schemes that can be used. Then a short peek is taken into existing work on recommender system-aided staffing and the peculiarities that case Futurice imposes on the problem, Futurice's current staffing process is also introduced with company culture and existing practices and tools in mind. Also, different technical options are explored with their up and downsides explained for this particular task.

This thesis evaluates the feasibility of a recommendation system powered staffing assistant by implementing a recommender system-aided staffing finder for Futurice internal use and measure its feasibility and accuracy through expert interview and data analysis. The solution in the form of implementation is presented to aid Futurice's staffing workflow. The presented hybrid recommendation system produces results either based on given skill input or input in the form of another person by leveraging information about the employees already collected during Futurice's normal staffing process. The recommendation system's performance is evaluated through an expert interview as well as data analysis. The solution is shown to provide the desired functionalities that were requested by the staffers and proves to be feasible for the task it was implemented while emphasizing the importance of used data and careful consideration of the case it will be used for.

Keywords Recommender systems, staffing, assisted decision making, digital service consultancy , employee ranking

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Aleksi Asikainen	
Työn nimi Projektihenkilöresursoinnin avustaminen suosittelujärjestelmällä	
Koulutusohjelma Master’s Programme in Computer, Communication and Information Sciences	
Pääaine Computer Science	**Pääaineen koodi** SCI3042
Työn valvoja Prof. Eero Hyvönen	
Työn ohjaaja FT Michael Samarin	
Päivämäärä 31.7.2020 **Sivumäärä** 56+10	**Kieli** Englanti

Tiivistelmä

Niin projektiosaamisen etsiminen kuin työpaikkojen täyttäminen on muuttunut yhä edistyneempään suuntaan 2000-luvun alusta. Tätä ongelmaa on pyritty paikkaamaan lisäämällä henkilöstöstä ja hakijoista kerättävän tiedon määrää ja laatua, jotta henkilöstöhallinan prosesseja voidaan joko helpottaa tai täysin automatisoida. Useita eri toteutustapoja on ehdotettu eri tutkijoiden toimesta useisiin eri tilanteisiin räätälöitynä kuitenkin pohjimmiltaan niin, että niitä yhdistää pyrkimys helpottaa annettua tehtävää vastaavan henkilöstön valintaa ja löytämistä. Edellämainittu tutkimus on todennut, että suosittelujärestelmien käyttö sekä rekrytoinnissa että projektien henkilöresursoinnissa on tehokas ja nopea tapa helpottaa projekti- tai rekrytointivastaavien työtä resursoinnin osalta.

Tämän diplomityön aluksi selitetään teoria suosittelujärjestelmien avainkäsitteiden osalta ja lisäksi avataan tekstipohjaisen tiedohaun teoriaa ohjelmistotekniikassa sekä hyötyperäisen haun perusteita. Seuraavaksi esitellään suosittelujärjestelmien käyttöä henkilöstöresursoinnin apuna aiemmassa tutkimuksessa. Lisäksi työ valottaa Futuricen projektiresursoinnin nykyistä tilaa, tarpeita ja ongelmia sekä esittelee yrityksen kulttuuria ja käytänteitä sekä niiden vaikutusta projektien resursointiin. Teknisten toteutusten hyviä ja huonoja puoli punnitaan aiemman tutkimuksen valossa.

Diplomityössä esitellään Futuricen henkilöresursoinnin päätöksentekoa helpottava suostittelujärjestelmän implementaatio, jonka soveltuvuutta ja tarkuutta evaluoidaan sekä data-analyysin että asiantuntijahaastattelun kautta. Toteutettu hybridisuosittelujärjestelmä, joka hyödyntää Futuricen aiemman järjestelmien dataa, tuottaa haluttuja tuloksia sekä taitoperusteisella että ihmisperusteisella haulla. Tulosten perusteella järjestelmä sellaisenaan avustaa resursointiprosessia, mutta lisätarkkuudella datankäsittelyssä sekä tarjottujen rajaimien lisäämisellä voitaisiin tuottaa vielä tarkempia tuloksia.

Avainsanat Suosittelujärjestelmät, henkilöstöresursointi, avustettu päätöksenteko, ohjelmistokonsultointi

Preface

I want to thank Prof. Eero Hyvönen, Prof. Tuomas Aura and my instructor Dr. Michael Samarin for their guidance and help. I would also like to extend my thanks to Futurice Oy for providing me with the opportunity to conduct this research. Finally, I want to thank my family and friends for their continuing support throughout the writing process.

Otaniemi, 31.7.2020

Aleksi Asikainen

Contents

Abbreviations

CSP	Constraint Satisfaction Problem
NN	Nearest Neighbour
kNN	k-Nearest Neighbours
MAUT	Multi-attribute Utility Theory
B2B	Business to Business
FTE	Full-time equivalent
UTZ	Utilization
SVM	Support Vector Machine
SVC	Support Vector Classification
PoC	Proof of Concept

1 Introduction

Filling job openings with suitable talent has been a problem the society have been battling with since the dawn of commercialism. Even though the nuances have somewhat changed from the master-apprentice times, the base of the problem still remains the same: matching supply and demand in competences. Nowadays companies like Monster and Barona offer extensive machine learning powered web services that job applicants and employers can use to get personalized recommendations for job openings. Some companies have also taken this further and use their own software solutions to manage their internal human resources to their fullest extent.

Modern software development in Finland is powered by a quite extensive repertoire of software consultancy agencies. Most software projects both private and public have multiple agencies providing development services. Usually consultants are picked with a specific area of expertise in mind. For example, when deciding on a right developer for front-end work, small differences in experience on specific frameworks can be the deciding factor in winning the case or not. While this is not always the case, having that small upper hand compared to competitors in public cases almost always can guarantee you the place on the next round of tenders.

In this thesis, the main goal is to create a recommender system based on multiple internal data-sources that are already used partly in the staffing process. The research questions are as follows: how to streamline Futurice project staffing process with a recommender system and how it fares against the manual candidate search in terms of feasibility and accuracy. The goal is to create a tool for Futurice operative crew to use when searching people suitable for new project openings or rotations. The main focus is on feasibility and accuracy of results obtained when using a recommender system and how it compares to manual process of staffing.

The structure of this thesis after this Introduction chapter is as follows: First background and basics behind multiple attribute theory, different recommender systems and machine learning algorithms is presented. Also theory behind constraint satisfaction problems and k-nearest neighbours is explained. Next, related work on assisted staffing is explored and case Futurice as well as the results of the initial interviews are presented. Based on the findings in the previous chapter a solution model is presented in the next chapter. Also different technical approaches to implementing a recommendation system for staffing are explored in chapter 4. Chapter 5 presents an in-depth take on the two different implementations produced, a proof of concept and the final, to answer the first research question. Next the results to measuring feasibility and accuracy of the system implemented are presented for both functionalities that the system offer. Finally the whole thesis is summarized and further work discussed in the Conclusion chapter.

2 Background

2.1 Recommender Systems

Recommender systems have seen their rise in the late 20th and early 21st century through implementations like Netflix, Tapestry, and Grouplens [5, 40]. These systems swiftly revolutionized how content was filtered and offered to users, and today they power the sales of nearly every web store and online marketplace [16]. Partially thanks to recommender systems modern web stores can maintain sales on the "Long tail" of products. Thus meaning that the online sales providers can offer a huge catalog of products and thanks to recommender systems, customers can find even the most marginal products related to their interests [2]. Naturally, the same principle of vast selection can be applied to jobs, entertainment, and information, too.

The fundamental idea behind recommender systems is to find common characteristics between items and users either through other user's actions or through the information available on the item itself [40]. "A recommender system calculates and provides relevant content to the user based on knowledge of the user, content, and interactions between the user and the item" [16]. Recommender systems can be divided to three popular categories based on what they use to produce the recommendations: The systems based on actions and reviews of other users are called "Collaborative Recommendation Systems" [40], the systems that obtain the results through analyzing descriptions and properties of items are called "Content-based Recommendation Systems" [37] and lastly there are systems that combine aforementioned techniques called "Hybrid Recommendation Systems" [16]. There exists also a few more specific categories: demographic, knowledge-based, and utility-based. In demographic recommendation systems, the users are clustered to demographic groups and popular items inside the cluster are recommended to users [7]. Knowledge-based recommendation systems use case-based user input to determine relevant items. Recommendation systems that try to factor in non-product information like availability and delivery times to provide value to recommendations are called utility-based systems. Usually, utility-based systems employ constraint satisfaction problems to power their production of recommendations [6].

2.1.1 Content-based

"Content-based recommendation systems analyze item descriptions to identify items that are of particular interest to the user" [37]. In content-based systems the item representation, meaning the structure how data is stored, plays an important role. The item data can be stored in multiple attributes with each attribute conveying information about a single aspect of the item, for example, a column stating what kind of cuisine a restaurant serves [37]. In the aforementioned cases the data is structured. The relevant information about the item may also reside in the unstructured description texts, which means that it needs some form of preprocessing before it can be used in producing recommendations. One of these preprocessing techniques, known from search engines, is called stemming and *TF-IDF* (term-frequency times inverse

Technique	Pros	Cons
Collaborative Filtering	• Serendipity • Domain knowledge not needed • Adaptive: Learns over time • Implicit feedback sufficient	• Cold start • "Gray sheep" • Quality depends on extent of history • stability vs plasticity
Content-based	• Domain knowledge not needed • Adaptive: Learns over time • Implicit feedback sufficient	• Cold start • Quality depends on extent of history • Stability vs plasticity
Knowledge-based	• No cold start • Sensitive to changes of preference • Can include non-product information	• User input • utility function • Static suggestions
Utility-based	• No cold start • Sensitive to changes of preference • Can include non-product information • Can map from user needs to products	• Static suggestions • Knowledge engineering required

Table 1: Positive and negative aspects of each recommendation system technique [6, 16]

document frequency) values but techniques like LDA (latent Dirichlet allocation) are also used to complement shortcomings of others [37, 16]. First, the unstructured text's words are stemmed to their root forms and then analyses the stemmed text for topics. These common topics can be then used to produce recommendations of similar items that share some of the topics with the original seed [16]. TF-IDF and LDA are not the only possible algorithms to drive content-based recommendations. Once the data has been structured and vectorized, Pazzani and Billsus [37] list that an implementation can employ algorithms like NN (nearest neighbors), Decision trees, probabilistic methods like Naive Bayes, or Linear Classifiers.

To match these items to users for recommendations, a user model has to be constructed also. In content-based systems, the model is constructed from the user's history and input. The items the user has interacted already have their structure calculated and can be used as a group to produce similar items and lastly just rank them if such ranking data exists [16]. While content-based recommendation systems do not suffer from cold starts (lack of initial data) on the item side, new users still produce problems as the system has no information about their interests [16]. This kind of systematic approach to categorizing items also produces very similar recommendations but having a very deterministic recommendation system can also be a desired feature.

2.1.2 Collaborative

Collaborative recommender systems base their recommendations on the opinions and actions of other like-minded users [42]. Unlike content-based, collaborative recommender systems try to classify and cluster users based solely on their ratings or actions. Compared to content-based systems, collaborative filtering does not need any domain knowledge in the system itself as there is no need to analyze the content itself [6]. As shown in Figure 1 collaborative filtering can be divided into two basic types: item-based and user-based [16, 42]. These two differ in the way of obtaining recommendations from user data. Item-based collaborative filtering uses ratings the user has already given to products and is in that sense closely related to content-based recommendation systems. The crucial difference comes in finding similar items to recommend, as that is done using other users that have rated the item at hand and then the recommendations are calculated from the ratings of other items they have also rated [42].

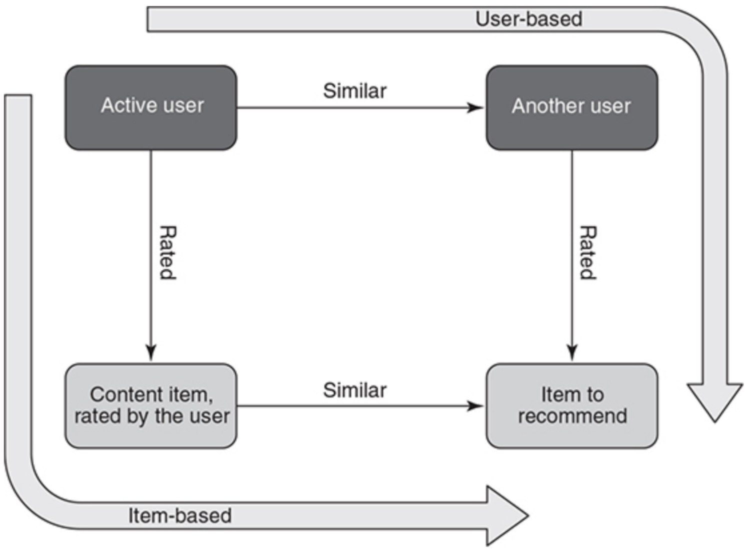


Figure 1: The two ways to utilize user data in collaborative filtering [16]

User-based recommendation systems take an even more social approach to produce recommendations by trying to find similar users among the userbase and recommending new items that the original user has not interacted with but similar users have liked [16]. One of the most lucrative attributes of user-based solutions is the ability to recommend completely unrelated but still relevant items, thus increasing the serendipity of recommendations [16]. These kind of "discoveries" are what makes every visit to the site unique, also increase the user's sense of satisfaction. While having serendipity in recommendations can increase user retention and overall user experience, it does increase the risk of proposing unsatisfactory results [22]. Traditionally collaborative recommendation systems have suffered from cold start problems as can be expected when relying on users supplying the training data. Multiple different strategies exist to combat this problem ranging from mandatory initial questionnaires to shadow profiles to flat out hiding recommendations until relevant data exists [16]. Also highly unique users, "gray sheep", might become problems as their niche tastes do not overlap with any other users making it impossible to produce recommendations based on social similarness [6]. Probably the most robust way of avoiding the cold start problem is to implement a hybrid recommendation system.

2.1.3 Utility and Knowledge-based

Utility-based recommendation systems try to model the user's needs through a utility function that produces weights for different item attributes. As the function modeling does not leverage any data, it can be quite cumbersome for the user to supply all the information for it [43, 6]. Utility-based systems do excel at circumventing cold starts and they can include various non-product attributes like availability into the recommendation process. Thus, they excel in professional systems, where users have expert knowledge of the items they are filtering [6]. To ease the burden of the user inputting every minute detail to produce a utility function, the systems themselves can provide some exemplary pre-configured functions that the user can select as basis or seed for their query. Thus, according to Burke [6] "This (utility-based recommending) might be feasible for items with only a few characteristics, such as price, quality, and delivery date, but not for more complex and subjective domains like movies or news articles." Also as the whole system is highly specific to the domain and even a small change in the catalog of items could force a rewrite of the query logic of the system.

Knowledge-based systems rely on knowledge on the nuances of similar items inside the domain [8]. "Knowledge-based approaches are distinguished in that they have functional knowledge: they have knowledge about how a particular item meets a particular user need, and can therefore reason about the relationship between a need and a possible recommendation" [6]. Burke [6] divides knowledge needed into three categories: catalog, functional, and user knowledge. Catalog knowledge is information about the items themselves, like in case of recommending restaurants, the system knows that Thai cuisine is a subcategory of the Asian kitchen. Functional knowledge maps users' needs and objects, for example, that 'quiet with ocean view' is a trait of a 'romantic' restaurant. Lastly, user knowledge is needed to actually provide recommendations, it answers the question of what this particular user wants [6, 8]. Like utility-based recommenders, knowledge-based ones are highly specific to their domain, and information acquisition is the biggest challenge along with serendipity of results [6].

2.1.4 Hybrid

As stated above all flavors of recommender systems have some kind of downside: content-based systems lack surprise and serendipity and also have problems with new users, collaborative systems suffer from both item and user cold start problems, knowledge-based need a lot of input and utility-based can be only used with specific data [6, 16, 22]. Naturally combining these different approaches to recommendation might ease the problems they face and usually for example collaborative filtering is combined with some other technique to circumvent cold start problems [6, 7]. While hybrid recommenders can alleviate problems that single technique recommender systems face they can also bring more stability and accuracy as Pazzani [37] states in his research of combining results of content-based, collaborative, and demographic recommenders. From Pazzani's time of just comparing ranks of produced results, the field of hybrid recommenders has taken leaps in coming up with different ways to aggregate and combine the results and parts of different recommenders. While selecting the correct variants of recommenders for the hybrid make a difference, the aggregation technique used also plays a big role in producing desired results.[37, 7] Burke [7] lists seven different implementation approaches to combining results of different recommendation system variants:

- Weighted: The score of different recommendation components are combined numerically.
- Switching: The system chooses among recommendation components and applies the selected one.
- Mixed: Recommendations from different recommenders are presented together.
- Feature Combination: Features derived from different knowledge sources are combined together and given to a single recommendation algorithm.
- Feature Augmentation: One recommendation technique is used to compute a feature or set of features, which is then part of the input to the next technique.
- Cascade: Recommenders are given strict priority, with the lower priority ones breaking ties in the scoring of the higher ones.
- Meta-level: One recommendation technique is applied and produces some sort of model, which is then the input used by the next technique.

Falk [16] does a little more abstract division into 3 different categories

- Ensemble Recommenders: Internally combines results of the recommenders contained through some form of aggregation
- Monolithic Recommenders: Borrow parts from other recommenders and combine them to produce hybrid results

- Mixed Hybrid Recommenders: Returns the union of all involved recommenders ordered by score if applicable

As Falk also divides ensembles into weighted and switching, these two authors seem to have quite similar views on different hybrid recommenders, so no new forms of hybrids do not seem to have arisen during the last ten years. While hybrids bring many benefits they are not a silver bullet as they introduce a large amount of complexity to the system, tend to need more effort to get them to work, and might introduce performance problems, as instead of running one recommender you might be now running three for each request on top the aggregation itself. Also, choosing the correct type of hybrid as well as the types of used recommenders inside the hybrid is a slow process as the correct combination is highly data specific. [7, 16]

2.2 k-Nearest Neighbours

The nearest neighbor (NN) rule is one of the simplest machine learning approaches to classification and regression [11]. The strongest features of nearest neighbors algorithms are lazy learning and simplicity. Thus, small data sets with high edit rates are where these kinds of algorithms shine as lazy learning means that most of the computation is done during consultation time [47] and simplicity ensures that changes to models do not cause major changes to the implementation itself. The possibility to keep the data structure as dynamic as possible also eases the development of the data sources as developers of those applications or databases do not need to worry about changes breaking things downstream.

Basics of nearest neighbour rule is defined as follows: let there be M classes numbered $1, 2, ..., M$. Let each pattern be defined in a N-dimensional feature space and let there be K training patterns. Each pattern is a pair (x^i, θ_t), $1 \leq t \leq K$ where $\theta_t \,\epsilon\, \{1, 2, ..., M\}$ and

$$x^t \,\varepsilon\, \left\{x_1^i, x_2^i, ..., x_N^i\right\}$$

is the set of feature values for the pattern. Let $T_{NN} = \{(x^1, \theta_1), (x^2, \theta_2), ..., (x^K, \theta_K)\}$. The nearest neighbour to value x is defined as follows:

$$\min d(x^i, x) = d(x_n, x) \quad i = 1, 2, ..., n. \tag{1}$$

Where distance function d can be any distance in the N-dimensional feature space. This distance decides the nearest neighbour of x and thus the class of an unseen point x will be set to the class of it's closest neighbour. [21, 10] The nearest neighbour rule defined above is actually a generalization of kNN called 1-NN as it only uses the closest neighbour to determine the class [21]. Modern use cases for nearest neighbours contain both classification and regression and usually they employ the kNN to do either voting or weighting based on their distance.

2.3 Support Vector Classification

Support vector machines (SVM) offer a way to classify data efficiently [24]. The point of classification is to predict the class of a previously unknown item based on its features. The basic idea behind support vector classification (SVC) is to find a suitable hyperplane between the known items of classes based on their attributes in an N-dimensional space, which then acts as a mapping function for the classes and can be used to classify previously unseen items [9, 24]. Formally as Keerthi and Lin [28] define, with label-instance pairs $(\mathbf{x}_i, y_i) i = 1, ..., l$, where $\mathbf{x}_i \,\epsilon\, R^m$ and $y \,\epsilon\, \{1, -1\}^l$ that SVMs are solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i\left(\mathbf{w}^T\phi\left(\mathbf{x}_i\right) + b\right) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, ..., l \end{aligned} \tag{2}$$

where function ϕ maps vectors $\mathbf{x}_i$ into a higher dimensional space with w being the normal vector to the resulting hyperplane, $C > 0$ being the penalty parameter of the error function and b determining the offset from the hyperplane along the normal vector w. Kernel functions are defined as $K\left(\mathbf{x}_i, \mathbf{x}_j\right) \equiv \phi\left(\mathbf{x}_i\right)^T \phi\left(\mathbf{x}_j\right)$. Multiple kernel functions have been proposed but most common are the following four:

- linear: $K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \mathbf{x}_i^T\mathbf{x}_j$
- polynomial: $K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \left(\gamma\mathbf{x}_i^T\mathbf{x}_j + r\right)^d, \gamma > 0$
- radial basis function (RBF): $K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \exp\left(-\gamma\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\right), \gamma > 0$
- sigmoid: $K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \tanh\left(\gamma\mathbf{x}_i^T\mathbf{x}_j + r\right)$

Choosing the kernel right will be decisive on the accuracy of the classification results as can be seen from Figure 2 obtained from Sci-kit learn's, a machine learning library for Python, documentation [1]. In the aforementioned case the aim is to predict the subspecies of the Iris based on the specimen's sepal length and width, using a 2 dimensional dataset of sepal lengths and widths of Iris' three subspecies from Fisher [20]. In each frame the kernel is trained with the same training data and then displayed with the classification areas of each of the subspecies to enable easy comparison. The kernel choice should be done on the basis of the data and usually either through testing and cross-validation or just visualizing the data can be enough to determine if one needs either a linear approach or a more complicated one [39].

2.4 Document Ranking

All recommendation systems do not use data that users provide as actions, namely content-based and knowledge-based systems. In some cases, the interesting features can be as part of long descriptive texts that need preprocessing before they can be meaningfully transformed into values and vectors for recommendation systems to leverage. Ranking long texts or documents based on a query of words has been a particular point of interest in computer science for some time as the number of documents

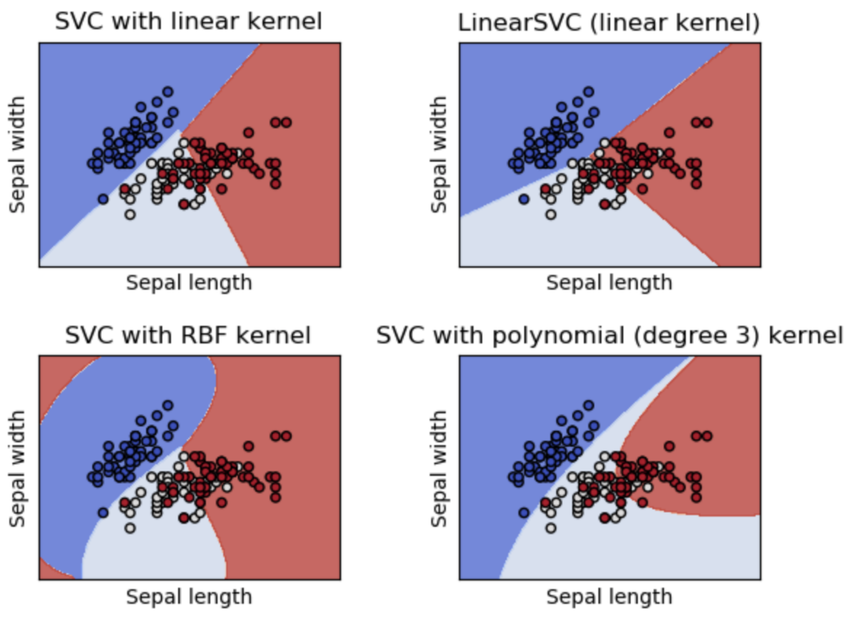


Figure 2: The decision boundaries that each Sci-kit learn's different- kernel implementations produce [1] for the Iris flower dataset by Fisher [20]. The sigmoid function does not make sense to be applied here as the data is not naturally divided in a circular manner.

and data has increased and finding the most interesting matches has become a priority.

The most rudimentary way of retrieving documents is to just perform a boolean search on the document: "do these terms appear in the document" [34]. Naturally, such a naive retrieval method produces a huge amount of results when the number of documents increases, and a more sophisticated system is needed for more granular searching and possibility to rank the results. The vector space model is an easy way to compare the similarity of documents by representing each document and the query vector in the same vector space. To represent each document as a vector, a way of transforming them is needed and term frequency-inverse document frequency, TF-IDF, is an industry-standard text term weighting scheme widely used in modern search engines [34, 4]. Luhn [31] who introduced the basic idea of term frequency as part of mechanized information retrieval, states that the term frequency should be, at its simplest, the frequency that each term appears in a document and as such could work as a basis of a score how relevant that term is for said document [31]. For example, a document that has hundreds of appearances of the word "car" probably contains a lot of information related to cars. Spärck Jones [45], to whom the inverse document frequency part of the equation is attributed to, learned that more could be deduced from the rare words that a document contains compared to the whole corpus than from the terms that are shared by the whole corpus. Specifically, Spärck Jones states that the weight of the term should be inverse to the frequency of its appearance in the whole document collection. Thus as Manning et al. [34] define it the TF-IDF weighting scheme assigns to term t a weight in document d given by

$$\mathrm{tf}-\mathrm{idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t \tag{3}$$

To acquire a full vector representation of a document a TF-IDF weight is calculated for each term that appears in the document ultimately forming the vector. Once the vectors are constructed there is no need to update the index anymore unless new documents are added to the corpus which affects the inverse frequencies of all the terms of the new document [34]. The same vectorization treatment is also done to the queries and after it, the similarity score can be acquired by calculating the vectors' similarities through for example cosine distance [34]. To find the score of each document d compared to the given query q, Manning et al. [34] define cosine distance as follows:

$$\mathrm{score}(q,d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|} \tag{4}$$

with $\vec{V}(q)$ denoting the vector representation of query q and $\vec{V}(d)$ the vector representation of document d. Furthermore, each vector is equalized to be of the same length to not take into account the length of the document when comparing similarity. Naturally more sophisticated methods of calculating vector space similarity have been designed but the basis behind these systems is still mostly based on the TF-IDF weighting.

BM25 Okapi

One ranking algorithm that has raised a moderate amount of interest is BM25, a state-of-the-art TF-IDF based ranking function, first introduced by Robertson et al. [41, 32] as a part of a City University of London's information retrieval system called Okapi. BM25 is based on a probabilistic retrieval framework due to the term weighting scheme it uses [41]. It scores a document D compared to query Q as defined in follows:

$$\sum_{q \in Q \cap D} \frac{(k_3+1) \cdot c(q,Q)}{k_3 + c(q,Q)} \cdot dtf(q,D) \cdot \log \frac{N+1}{df(q)+0.5} \tag{5}$$

Where $c(q,Q))$ is the count of the term q in Q, N is the total number of documents, $df(q)$ is the document frequency of q, and k_3 is a parameter [32]. By utilizing this parameter k_3 we can easily accommodate the saturation of terms in those documents by letting terms that occur multiple times on the document have a diminishing impact on the score of the document. This leads to ranking documents that include all the query values over documents that contain only few of them many times. BM25 also takes a probabilistic approach to the document frequency as it drops the significance of a term more rapidly the more it occurs across the whole document collection. Thus eliminating stopwords more efficiently than the traditional TF-IDF approach.

2.5 Constraint Satisfaction Problems

"Many tasks can be seen as constraint satisfaction problems. In such a case the task specification can be formulated to consist of a set of variables, each of which must be instantiated in a particular domain and a set of predicates that the values of the variables must simultaneously satisfy." [33] Kumar [29] states that a large number of AI and other computer problems can be modeled and solved as a constraint satisfaction problem. So once the problem can be expressed through a finite number of constraints and finite and discrete variable domains, then it can be solved or proved unsolvable using a brute-force method called generate-and-test in constraint satisfaction problem (CSP) context [33]. Of course, multitude of more efficient methods exist and one of the simpler ones is backtracking where values are chosen from domains and after each selection, the constraints are checked. The aforementioned process is repeated until a solution is found or a constraint cannot be satisfied. If such unsatisfactory assignment is encountered, the algorithm backtracks to the previous state known to be satisfactory [29, 33].

CSP's in Recommender Systems

Some cases of recommendation tasks can be seen as constraint satisfaction problems and especially knowledge-based recommendation systems can leverage the explicit nature of CSP-domains and -constraints [17, 19]. Depending on the UI choice, the majority of the constraints might come from either the user or the domain knowledge experts when building the system. Zanker et al. [48] propose an interface that essentially lets the user build the constraints from ground up by listing possible variables and domains. In the aforementioned solution domain knowledge comes from both the user and the implementers. Being quite complicated to use, this kind of system requires that the user is quite versed in terms of the domain at hand and as such is not that suitable for large scale use. Felfernig et al. [17, 18] propose a more user-friendly approach to building constraint-based recommender system interfaces through conversation-based dialogue between the user and the system. In the conversational model, the user is asked a series of questions with the intent to fill out the constraint criteria in terms of the bounds of the constraints. In this model, the domain expert has already defined the filtering and compatibility constraints which provide information for the system about relations between the attributes and aggregated multiple constraints under the same question [17]. For example, if a user wants a car that is good for the environment it can mean that it has a low fuel consumption rate, that it uses electricity, or that the production emissions have been low enough to offset the emissions that it will produce during use. The aforementioned could be also defined explicitly by specifying 'show me electronic cars, cars with low fuel consumption and low production emission models', which could be impossible for the system to understand.

Constraint based recommender systems are by nature quite restrictive and as such might not be able to produce any items that fulfill all the defined criteria which is called over-constrained model [48]. To battle this few strategies exist: weaken or relax some of the constraints or revise the preference model [48, 38]. For example the system can provide different lists of tradeoffs for the user to consider [38]. On the other hand Zanker et al. [48] propose a more automated approach by trying to deduce which attributes are not as important to the user and try to relax the constraints on those parts by ignoring them.

2.6 The Multi-attribute Utility Theory

Making decisions based on a complex list of attributes can be difficult and aiding this process of ranking the alternatives can vastly benefit the decision-maker. The Multi-attribute Utility Theory (MAUT) is a technique introduced by von Winterfeldt and Ward [13] to facilitate decision making in situations where the decision is to be made from a limited number of alternatives with distinct attributes that can be weighted. Jansen [26] states that all multi-attribute methods include five steps:

1. Defining alternatives and value-relevant attributes
2. Evaluating each alternative separately on each attribute
3. Assigning relative weights to the attributes
4. Aggregating the weights of attributes and the single-attribute evaluations of alternatives to obtain an overall evaluation of alternatives
5. Performing sensitivity analyses and making recommendations

During the first step, the collection of possible objects or decision choices should be gathered. Then among that collection, one should define the attributes that are the basis of the decision [26]. Keeney and Raiffa [27] state that the set of attributes should be comprehensive, measurable, and minimal. The scales of the attributes should be also natural if possible, meaning that they should be tied to their real-world counterparts, e.g., years, kilograms, meters. [13]. In the next step, each objects' each attribute should be evaluated separately [26]. It means that each attribute's value function should be determined in terms of how each attribute describes the desirability of the object and at what scale. For example, a car might be the most desirable if it has a 2-liter engine and least desirable if it has either 1-liter or 3-liter engine. Anything above 3 and below 1 liter would get 0 as score while 2 liters gets 100. Identifying these limits and determining the function between them is the point of this step. After determining the desirability scale of each attribute they should be assigned importance scores that best reflect the respondent's priorities [26]. Von Winterfeldt and Edwards [13] state that the weight of all the attributes should be then calculated relatively from all weights:

$$w_i = \frac{w_i^{'}}{\sum_{i=1}^{n} w_i^{'*}} \tag{6}$$

Where w_i' is the importance of the attribute i, n the number of attributes and w_i the resulting weight for the attribute. Once the weights have been determined the score of each object needs to be aggregated from the attribute values and weights. Von Winterfeldt and Edwards [13] state that weighted linear additive preference function is the most common way. The function states that the multi-variate attribute utility for object x:

$$v(x) = \sum_{i=1}^{n} w_i v_i \left(x_i\right) \tag{7}$$

where $v_i(x_i)$ is the value of the object x ith attribute, w_i is the weight of the ith attribute and n is the number of attributes. The only thing left is to perform sensitivity analysis and display the results based on the scores obtained in step 4.

3 Assisted Staffing & Case Futurice

In the context of this thesis, information technology consulting is discussed as services provided to client companies usually in forms of developers, designers, and management consultants. The basic case of a software consultant project is that the client needs digital service creation assistance which involves stages like sketching out new ideas, designing the user interface, developing a new software implementation, updating and upgrading existing ones, or maybe even wanting to change their organizational ways of working. Staffing these projects can be hard and especially in large organizations that offer consultancy, managing the talent can become quite a burden. For that reason, some companies have started to assist it by developing specific programs for it.

3.1 Related Work on Assisted Recruitment and Staffing

Recruitment and staffing have been targets of optimization through the use of recommender systems since the late '90s. For example, Namahoot and Brückner [35] propose a content-based RS as a decision support system for staffing, Gertner, Lubar, and Lavender [23] try to ease MITRE internal staffing through a collaborative recommender system and Lu, El Helou and Gillet [30] implement a hybrid recommender for matching recruiters and recruit candidates. Barreto, Barros and Werner [3] utilize constraint satisfaction problems in staffing decisions.

Faerber, Weitzel, and Keim [15] introduce one of the earliest recommendation systems for recruitment. Their hybrid recommender system used latent semantic analysis between users that have interacted with job postings and then combined that with content-based attributes from CVs. While it obtained promising results, they state that the collaborative data might have fallen behind the content-based in terms of impacting the final results. Also, their system did not take into account the reliability of the data it was handling and it could not assign weights to different attributes that the recruiter or job seeker might have valued more.

Lu, El Helou and Gillet [30] take the weighted approach even further by translating their content-based data and interaction-based collaborative data into a directed weighted graph. Their hybrid recommender system aims to offer a Swiss job hunting website a system that both recommends vacancies and employers to job searchers as well as employee candidates to the employers. Their approach is to create a weighted graph between people, job postings, and employers by first computing a similarity between them based on CV's and profiles and weight the connection on the matching score. Then they create more edges from user interactions like visiting a certain posting, liking one, or applying with each assigned an own weight. Lastly, they utilize a modified PageRank algorithm to produce ranking between the connections in the graph.
Namahoot and Brückner [35] approach the missing of weights trough a content-based ontology-driven solution for recruiting. In their implementation, they utilize

Thailand's Department of Employment's catalog of skills and competences as the attributes which define a candidate's compatibility with the employer's needs. The problem with this kind of system rises from its reliance on structured data. Changes to the ontology need a system-level change and thus are quite slow.

Gertner, Lubar, and Lavender [23] have solved the ontology problem by making the queries fully dynamic. Their goal is to staff MITRE Corporation internal projects through Apache Mahout collaborative filtering. Gertner, Lubar, and Lavender [23] also try to infer the relationships between different skills by calculating a Jaccard coefficient between the person sets that hold particular skill. This way they can propose people that do not necessarily possess the desired skills but have a very similar skill set compared to the query.

Isias and Casaca [25] describe a hybrid recommender system for Portugal's State Financial Administration, which acts as a tool for their HR to find optimal employees for their internal tasks. They used semi-structured interviews to determine the background, task assignment procedures, task attributes, and organization's knowledge about their employees. Their solution was a hybrid between collaborative filtering and case-based reasoning, a subtype of knowledge-based system. Isias and Casaca [25] state that the prototype they built for the PSFA was a success and added significant value to the department's HR efforts.

Barreto, Barros, and Werner [3] introduce a significantly different approach to assisted staffing compared to the aforementioned solutions through the use of constraint satisfaction. They modeled the problem so that each project is divided into activities which state how long it takes, how many hours per day needs to be allocated, and what kind of characteristics are needed from the employee to fare in the activity. In addition, the system has few global restrictions which state that an employee cannot be allocated over their availability and that the employee must possess the minimum requirements for the activity to be eligible for it. Then they use utility functions to drive the optimization from different angles. Barreto, Barros, and Werner [3] few utility function examples from the most qualified team to the smallest team. They state that their solution was able to increase staffing efficiency in a controlled environment however the needed accuracy of the system did not necessarily reflect the accuracy that the staffers were used to work with. This increased the need to input data more precisely to the system than staffers perceived was needed.

3.2 Case Futurice

To be able to produce software that offers staffing recommendations and acts according to one company's values and conventions, the current way of staffing must be dissected. Almost all of the research presented in Section 3.1 starts by introducing the company or need behind the actual implementation as it is tightly coupled with the desirability of the results obtained.

3.2.1 Company

Futurice is one of Finland's modern digital service consultancies founded in 2000. Its main focus is to sell and offer expert services in the Business to Business (B2B) environment commonly called IT-service consulting. During the last 10 years, Futurice has been growing quite rapidly and nowadays it employs around 600 people with the biggest office being in Helsinki at around 300 employees. It also has offices in Tampere, Berlin, München, Stuttgart, Stockholm, Oslo, and London out of which Tampere and Berlin being the most mature business-wise in addition to Helsinki. These offices are referred to as sites inside the company and also in this thesis.

The organization is founded on business units of 20-60 people. All other sites consist of one business-unit except Helsinki which is divided into 4 business-oriented units and 3 internal tribes that cover areas like business management, sales, IT, legal, and social impact initiative Spice. The units acquire their clients and do most of their recruiting on their own. Staffing the projects is done with an emphasis on members of their business unit but loaning is encouraged if suitable talent is not available in their reserves. The business units and transitively each unit's managers and account managers are also responsible for the profit and losses that the business units generate.

The most important resource that the company has is its people. All of Futurice's revenue comes from client work and most of it is hourly billable with exceptions being different kinds of training or design gigs usually spanning only one or two days. So the core behind the business is quite clear, people work for clients producing value and thus profit. If any of these parts mentioned before is at risk, the whole company is at risk. So the decisions and work ways need to be adaptable, fast, and autonomous. This is reflected in the internal decision-making rule that powers the self-organizing teams' mentality called the "3x2"-rule as showed in Figure 3. It states that "when making a decision a person should always think how it will affect the people around, the business numbers and the clients, both now and in future." If all these aspects seem to be positive then the decision should be made.

As people are the company's most important resource, taking care of their well-being is one of the core values in Futurice. Taking care of employees has multiple different aspects inside Futurice from providing meaningful and interesting work to more practical things like a free breakfast every Friday. Usually, the consultants working a client project spend four days of their workweek at client premises and then have an "office day" on Fridays meaning that then most of the company is at Futurice office. Reasons behind this range from increasing company cohesion and belongingness to easing schedule pressures for meetings. Fridays also include competence specific talks that aim to spread newly acquired knowledge of innovations inside their silo of expertise. Once every month a special Friday is held to sync between all sites and to offer a chance to ask questions from officers about company matters. Consequently, as most of these company alignment and training occasions happen during work time people will not be able to allocate 100% of their work time

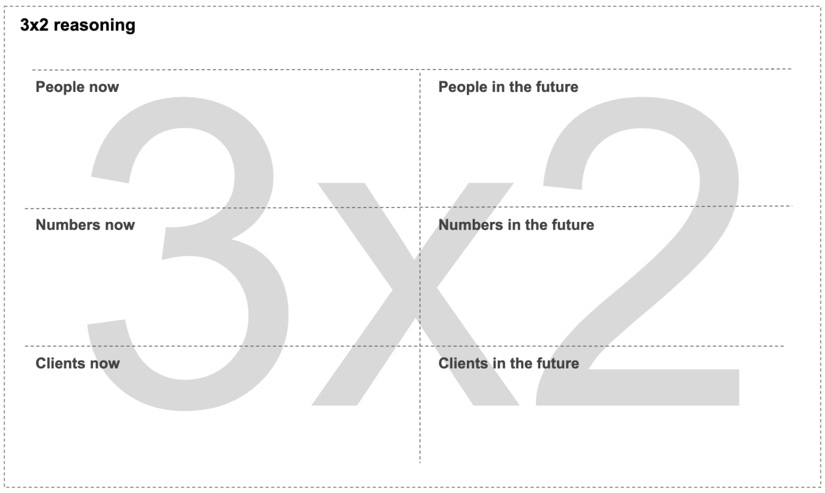


Figure 3: Part of Futurice 3x2 rule canvas used in internal decision making

to client work. Based on historical averages inside Futurice, a fully allocated person actually works 90% on billable client work and 10% goes into company internal events and tasks.

3.2.2 Power

Power is an in-house developed tool for forecasting numbers as well as doing resource planning in the context of the workforce. It has four main functions inside Futurice: resource planning, staffing, knowledge management tool, and financial forecasting. The resource planning side of Power is quite similar to tools like Harvest, Hub planner, and Float. In resource planning context Power is the only source of information about employee availability inside the company. As Figure 4 shows the view lists people with information about their business unit, main role, skills, and allocations. Allocations are shown as either blue or yellow bars, with yellow signaling that the allocation is at a proposal state and yet not been agreed with the client. The blue projects have already been signed and agreed on.

The allocation amount to a project is measured by full-time equivalent (FTE) with the basis being that 1.0 FTE equals to one person working 5 days per week 7.5 hours per day (8 hours per day in Germany). In addition to FTE, every person in the system also has utilization target (UTZ-target) which tells the percentage of the person's FTE that should be used for customer work. People with lower than 100% UTZ-target are expected to also participate in internal work alongside normal client work. Power combines pricing and account information to the allocation data producing approximate forecasts of the company's fiscal future. The forecasts are used to drive sales and as a benchmark of the current situation financially. Power provides few manual target values for sales and management to strive for.

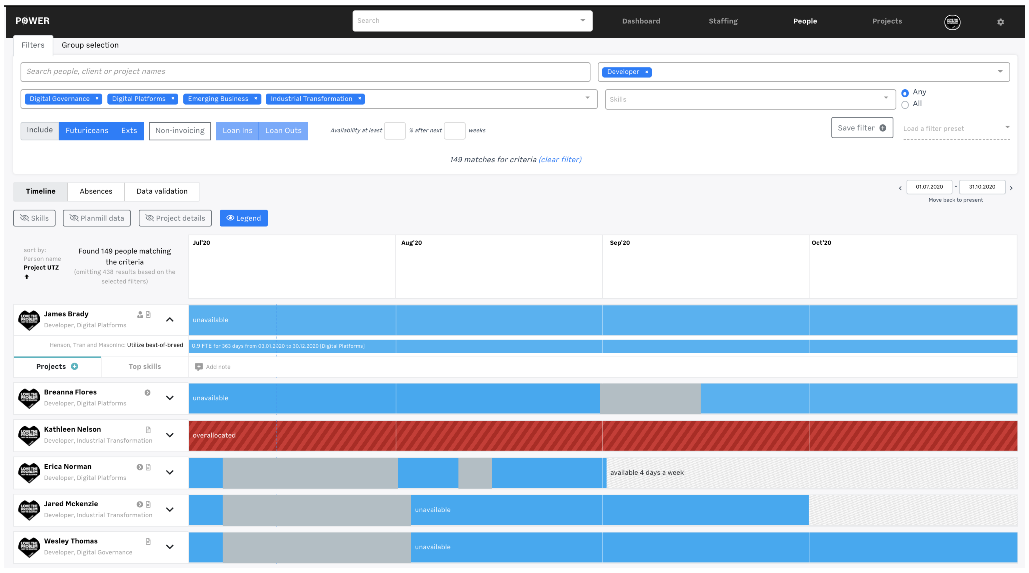
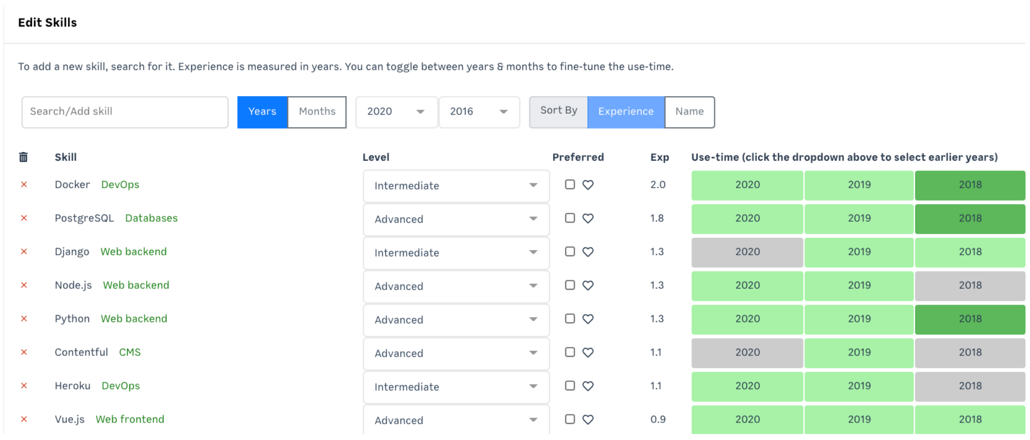


Figure 4: People view of Power, mainly used in staffing

Power stores information about people's skills and besides the tacit knowledge that staffers and salespeople have accumulated over the years, it also holds the most accurate representation of all employee's skills. The skills are inputted manually by employees themselves and should be updated at least twice a year in company-wide My Impact discussions. An example of the skill interface can be seen in Figure 5. The aforementioned My Impact discussions work as a checkpoint of the person's impact on the company's performance from the last 6 months, so they act as a natural point to recant which new skills have been acquired and how much the old ones have been utilized. The results of these discussions then drive the salary fairness meetings to decide who has earned a raise based on their actions and feedback received from co-workers and clients. While My Impact meetings keep the skills data somewhat up-to-date the incentive to update the skills is also personal. As in addition to just raw skill data and experience, the skill area of Power also contains information about which technologies or frameworks the person would prefer for the next project and also the period when these skills were last utilized can be stored as well. In addition to raw usage metrics people can also just define a proficiency level for themselves from a scale of Novice - Beginner - Intermediate - Advanced - Expert. The proficiency levels were added for people to better gauge their understanding in certain topics as time-based expertise does not reflect modern framework proficiency that well as technologies change quickly and for example, different design tasks do not scale on a linear time-based scale that well. By listing their competences and skills as accurate alongside preferences people can greatly affect how they show up in internal searches as well as reduce the back and forth chatter in staffing situations by offering the needed information beforehand. Power's skills are not limited to technological or professional experience only. For example, competences acquired outside of work, for example in hobbies, are encouraged to be inserted as well. This way even the skills unknown to the coworkers are uncovered and provide relevant information about the person's interests.



Edit Skills

To add a new skill, search for it. Experience is measured in years. You can toggle between years & months to fine-tune the use-time.

Search/Add skill | Years | Months | 2020 | 2016 | Sort By | Experience | Name

	Skill		Level	Preferred	Exp	Use-time (click the dropdown above to select earlier years)		
×	Docker	DevOps	Intermediate	☐ ♡	2.0	2020	2019	2018
×	PostgreSQL	Databases	Advanced	☐ ♡	1.8	2020	2019	2018
×	Django	Web backend	Intermediate	☐ ♡	1.3	2020	2019	2018
×	Node.js	Web backend	Advanced	☐ ♡	1.3	2020	2019	2018
×	Python	Web backend	Advanced	☐ ♡	1.3	2020	2019	2018
×	Contentful	CMS	Advanced	☐ ♡	1.1	2020	2019	2018
×	Heroku	DevOps	Intermediate	☐ ♡	1.1	2020	2019	2018
×	Vue.js	Web frontend	Advanced	☐ ♡	0.9	2020	2019	2018

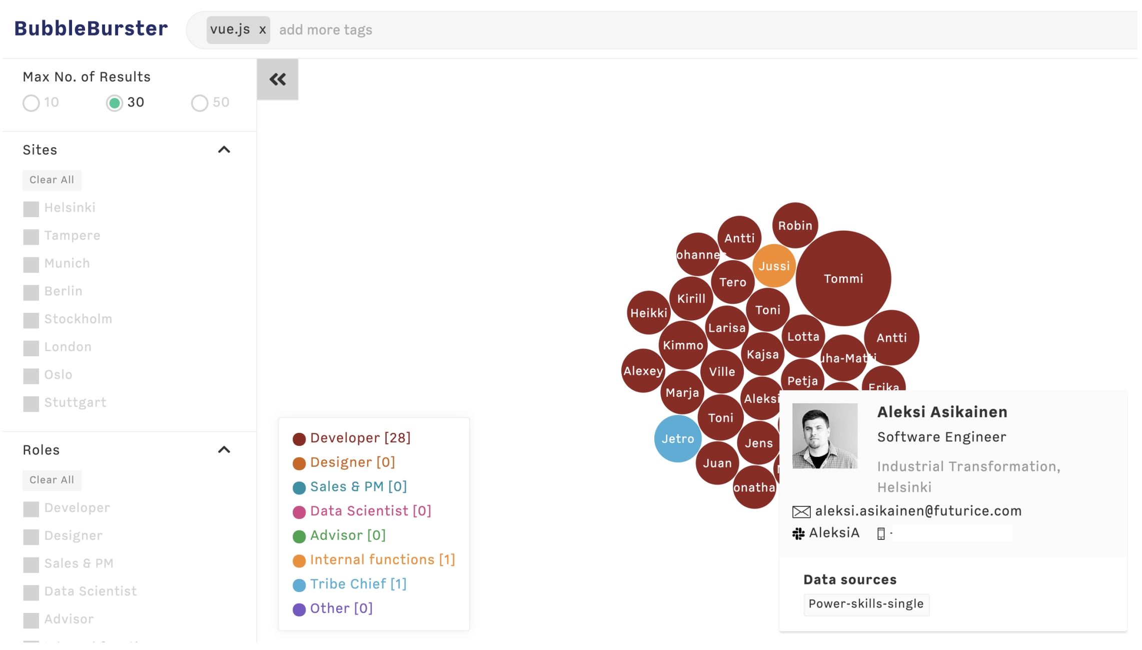
Figure 5: Skill input page of Power with annual experience separated from proficiency

The skill database combined with availability information, that Power provides, offers staffers and account managers most of the information needed to make staffing decisions. In addition to structured skill information Power also has a free text staffing note, that people can use to convey their desires and preferences to the sales and staffing. Usually, these comments note the most interesting technologies, desired customers as well as teammate preferences the person at hand would like to work with in the next project. It is also pretty normal for people to express what kind of social impact, for example, the client should have, should the new client be from the public or private sector, and what they would like the team composition to be like. It is not uncommon to also express the specific role they are after in the new project to strengthen some specific sector of their expertise next. Also, people usually list their latest projects to give the staffer a clear view of what they liked or disliked about their previous projects.

3.2.3 BubbleBurster 3000

Knowledge management is hard and in an organization that relies on people as its main resource, finding the know-how is crucial and still many companies struggle with it. Futurice took a more experimental route into knowledge management and tried to approach the problem with a more dynamic solution compared to manual knowledge banks like Power. Figures 6 and 7 show the user interface of Bubble Burster 3000 (BB3000 or BB3k), which is an internal search tool for finding people with desired knowledge. Unlike Power, which specializes in structured exact data, BB3000 analyses free form text produced in the company's Slack, work-time reports, calendar entries, and project names. The theory behind this is that people talk about technologies and subjects that interest them or they already work with them.

Work-time reports in Futurice have a small description part to them that describes what kind of work was done and in what relation. Usually, these contain main points of employee's workday such as "Refactoring the React front-end" companied with customer company. BB3k tries to capture the tacit knowledge that Futurice employees exchange daily by tapping into the text content they produce during their work. The technical implementation is done on top of a Bayesian reasoning platform provided by Futurice based company Aito.ai.


BubbleBurster

vue.js x — add more tags

Max No. of Results: 10 [✓ 30] 50

Sites
Clear All
- Helsinki
- Tampere
- Munich
- Berlin
- Stockholm
- London
- Oslo
- Stuttgart

Roles
Clear All
- Developer
- Designer
- Sales & PM
- Data Scientist
- Advisor

● Developer [28]
● Designer [0]
● Sales & PM [0]
● Data Scientist [0]
● Advisor [0]
● Internal functions [1]
● Tribe Chief [1]
● Other [0]

Aleksi Asikainen
Software Engineer
Industrial Transformation, Helsinki
aleksi.asikainen@futurice.com
AleksiA

Data sources
Power-skills-single

Figure 6: Bubble Burster 3000 search interface.

As the company produces more data for this tool to leverage every day, there is no way to actually go through and verify the accuracy of the added data. Naturally, this kind of data is by no means reliable for short term usage, but when enough interactions, calendar markings, and conversations are introduced to it, the sheer volume will start to give correct results on average. While some messages or calendar events might give hints about the employees' interests and expertise, not every discussion or work-time report does so. This produces false-positive matches especially because of how some people are much more active in Slack and others tend to only communicate with the team they have at the client premises or inside a client provided messaging platform that BB3k cannot leverage as a data source. A person talking about a subject might be also them asking for people who know about the subject or the words they used just happen to be homographs to the keywords used in the query. For example, sometimes staffers show up in industry-specific technical search terms queries as they tend to be the ones asking for experts who know about them in Slack. Fortunately the aforementioned is rare and can be circumvented in this staffing use case by filtering all the internal staff outside the group of candidates after querying.

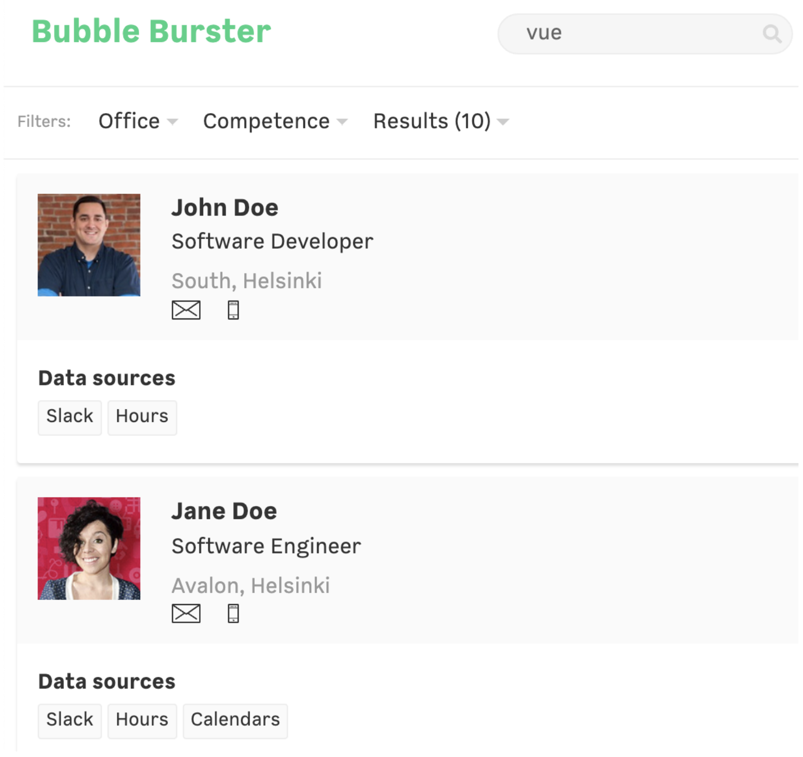


Figure 7: Bubble Burster 3000 search results in list mode, showing the data sources behind the results in detail

3.2.4 Staffing

The staffing process in Futurice consists of 4 parts: identifying the need, finding the candidates, discussing with the possible candidates, and deciding on the people that will be part of the offer. Usually, the salesmen identify the actual traits that the candidate should have and ask the business units' operations managers who would be the most suitable match for this particular case. Once a shortlist of candidates has been found, the salesmen proceed to discuss the case with the candidates to find a good match. At this point in the process, the person's preferences come to play. Once a candidate has been found it is proposed to the client. Normally the need comes from requests for proposal (RFPs) but it might also be that the client asks for extra manpower for an existing team. At this stage, the client already might have certain skill and competence requirements in mind especially when they are looking for a single addition to an existing team. It is also possible that when starting a greenfield project the client expects Futurice to propose a technology stack or design framework to use alongside the team of experts that know how to use them. It is also possible that when starting a greenfield project the client expects Futurice to propose a technology stack or design framework to use alongside the team of experts that know how to use them.

As requirements from clients can be so fluid, also the tools and processes for finding the talent need to adapt, too. It has to be possible to find available people with specific talent as well as to find people with broader expertise. Preferably both of these needs would be fulfilled through an internal software but as mentioned in Section 3.2.2 Power, the most utilized tool when staffing, only offers a constraint-based workflow. Basically when staffing a project with Power the user selects which site is preferred, what competencies are needed, and how much available FTE is needed for this allocation. Power then lists people matching the search criteria with all skills listed, not providing much outside the constraint-based approach.

In addition to Power results, also the staffer's tacit knowledge plays part in selecting suitable candidates for a vacancy in a project. He might know that the person has in their previous assignment done something similar to the upcoming project. There might be also some skills or competencies that the candidate has but has not supplied them to Power yet. Also, discussions with the candidates about the possible upcoming assignment are always held. These discussions aim to ask the employee about the suitability and appeal of the assignment. Lastly, the top candidates are usually offered to the client for consideration and then the client can pick the best fits.

3.2.5 Insights & Problems

The current staffing process has few problems identified in the first interview rounds shown in Appendix A. First of all the information available in our systems is mostly non-dynamic, with the exception of BB3K. Relying on manual updates on the skill information can be quite volatile as is only relying on the BB3K information. Thus a careful combination of these two could prove to beneficial. Some relevant staffing information might be also buried in the staffing note as free text. While staffers read these it usually happens later during the process when the pool of candidates has already been filtered to some extent. In such situation, a preference for relevant technology might be missed. In addition to information being hidden, also not all sources are utilized due to information being scattered into multiple systems. Some staffers might not even be aware of such possibilities as BB3k or just might feel that they do not offer accurate enough information to be useful.

The current search user interface (UI) is quite cumbersome to use and needs constant tweaking when trying to find different skill combinations with different availability. Especially soft searches where different skills are optional are impossible. While the people view in Figure 4 controls of Power are quite specifically tailored for only staffing purposes, the view itself does not contain all the needed information for staffing decisions. For example, it is missing the staffing comment, which provides crucial information about the person's current situation and aspirations for the future. Also, the visibility of people that should be doing customer work but appointed to internal work for the duration of their bench time is poor. It is customary at Futurice that people between projects are utilized to some extent in internal projects. As the system used staffing internal projects is the same used in client project staffing these people show up as allocated and thus will be wrongly ignored in staffing client projects. On the other hand, some employees only work in internal projects and should not be shown as viable candidates during the staffing process. The aforementioned overlaps of people's areas of work create confusion, but at current company size, it has not been found to be too problematic. But with the company growing it might become more imminent as staffers would not be able to know their candidates so well.

Few, more process than tool related problems, were also uncovered during the interviews. Most of them were related to communication during the period when a person has been planned to be proposed to a project. Usually, this happens when discussions with the top candidate have been had and they are not still marked into a proposal. Other times the staffers might have agreed face-to-face on an allocation decision but the information has not reached any internal channels yet.

From outside the interviews, two interesting aspects were noticed: project visibility and rotations. Rotations are employee switches usually initiated by the Futurice employee in a client project where they express their desire to change projects. As the contracts with clients usually promise that Futurice provides a certain amount of professionally talented people for each task, naturally a replacement has to be found. The person rotating in preferably should match the skill set of the person leaving, so providing a tool that eases the search would decrease the amount of manual labor needed to make a rotation happen. Project visibility was mentioned as an employee's ability to see upcoming projects, their clients, and technologies that are most likely going to be used. With this information, people could be more proactive about their own staffing already expressing their interests toward certain clients or techs. It would also give a better understanding of what kind of work is available in the market at the current time. As a result, people would not be waiting for that perfect match project saying no to projects with just mediocre offerings because they would know that there are no perfect projects coming up the sales pipeline.

4 Solution & Tech options

4.1 Proposed Solution

Based on the problems listed in Section 3.2.5 I identified a need for concentrated staffing tool inside the Futurice organization. The idea is to implement an recommender system that

1. Tries to predict what the staffer needs to make the search process more faster
2. Does initial ranking for the staffer so that finding the best candidates is faster
3. Combines free text information to have staffing comment information leveraged earlier in the staffing process
4. Uses BB3K as data source as it is not yet leveraged in the conventional staffing processes

The aim is to treat the recommender system as a decision assist tool that does more than conventional constraint-based filtering. To further map out the needs and shortcomings of the current system and process a set of semi-structured interviews were held with the current staffer workforce, they can be found in Appendix A. Based on the people interviewed the time used to view current systems results could be reduced by showing more accurate results or at least more relevant results near the top. Also, functionality showing meaningful alternatives to available people by recommending people around the given time range. Rotations can also be eased by offering recommendations using another person as the search query to find similarly skilled people.

4.2 Data

The two main data sources available are Power database and Bubbleburster 3000 API. The Power database can be accessed directly and with BB3k the API has to be used to leverage the aggregation power it offers. The Power database can provide two kinds of interesting information relating to staffing: allocation and skill data. The skill data is stored in two different mapping tables called 'preferences' and 'personcompetence' as shown in Figure 8. The existence of a 'preference' entry tells that the person would be interested to work with said technology or area of skill next. 'Personcompetence'-entry indicates that a person has experience with the skill it points to. A 'personcompetence' entry is inserted for each year the person has experience as can be input in Figure 5. The year information is voluntary and the skills can be submitted without any information about attained experience. Also, some remnant information from old systems is missing the experience information. The 'competence' table consists of competences categories and has a hierarchy built into it through the 'parent_id' foreign keys.

As mentioned in Section 3.2.2 Power acts also as an availability planner. As can be seen in Figure 9 availability data is stored in the database as allocations

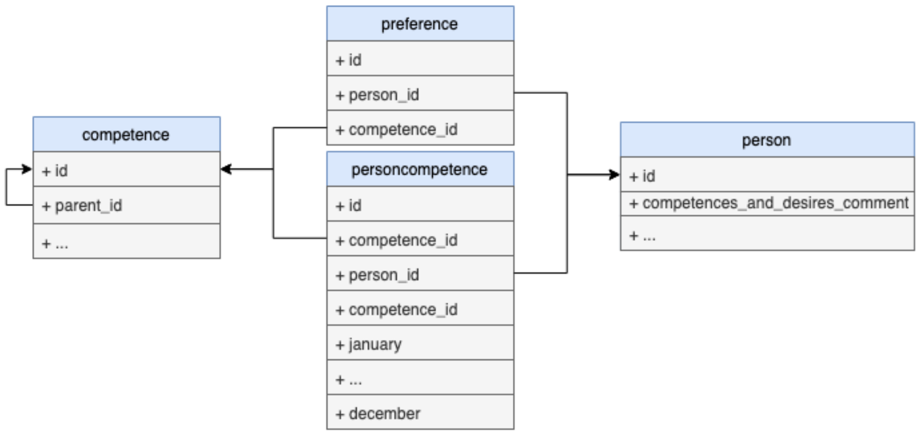


Figure 8: Skill related data structure in Power.

that link a project to a person. The 'total_allocation' -field contains the FTE-amount of the allocation and combining these values over time tells the person's availability at that a certain point in time. The 'proposed' field indicates whether the allocation is still in a tentative status with the client or if it is already agreed upon.

Power data is manual data that people themselves have submitted into it and thus it is quite reliable in the sense that the people do really have those skills and allocations. The only unreliable aspect of the database data is that it might be out of date as was mentioned in the initial interviews in Appendix A. This happens relatively often as mentioned in Section 3.2.2 when allocations change during negotiations.

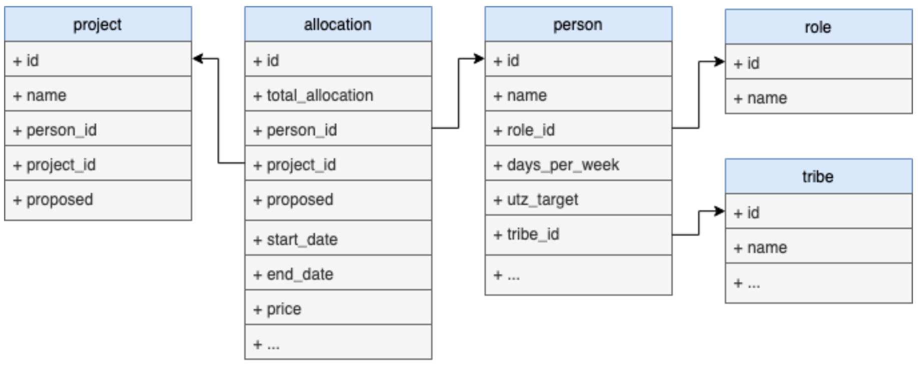


Figure 9: Allocation related data structure in Power.

4.3 Filter Prediction

As the Section 3.2.5 states, one of the problems of the existing tools is the long listing of irrelevant candidates that the primary staffing tool, Power, provides for each query. For example, many of Futurice's data scientists have skills that most of the developers do but they have strong preference and skillset towards data science and machine learning projects, so staffing them to a traditional software project is not preferable. Still, they show up on regular searches. Transforming as much of the search parameters of Power into autofill values will have at least a little impact on the ease and speed of use of the system. Unfortunately apart from the competence role of the person the other values proved to be very difficult to predict.

Predicting competence roles can be done with a classification algorithm. With SVM's being one of the most basic and widely used machine learning methods, it seems like a natural place to use it. As the documentation of Scikit learn [1] states that SVMs are a good choice of ML method if the amount of features is high and they perform well even if the amount of training data is somewhat low. At current time Futurice's systems have 1077 entries for people and 246 distinct skills which creates quite sparse vectors for each person. SVMs fortunately also handle sparse data well.

4.4 Recommender System

Choosing a suitable 'flavor' of a recommender system is crucial to produce meaningful results in a case-specific staffing scenario. The most important driver behind this decision is the availability and nature of data but also the procedures and conventions in the company staffing process play a significant role.

4.4.1 Collaborative filtering

The collaborative filtering based approach has at least worked for multiple recruitment recommender systems [15, 23] with promising results. As stated by [40] the basic idea behind collaborative filtering is the interaction between the user and item or in this case staffer and employee which produces the first problem in Futurice's case as there is no data collected about peoples interaction with internal tools. Also, collaborative filtering works well in situations where the aim is to recommend items to users, but in the case at hand, the purpose is to recommend people to item with the item being, in this case, the representation of an opening that the staffer has made.

One of the main reasons the needed data to utilize collaborative filtering is missing is that the Futurice's staffing pipeline is quite unidirectional at it's beginning. Meaning that the staffers contact people without projects and not the other way around. Based on my findings Futurice would need more visibility to the dynamic interests of people on the bench through a system that would allow unallocated employees to browse possible upcoming projects. Using the behavioral data the system could recommend projects to people much like Faerber et al. or Gertner et al[15, 23] have done. But as the purpose of this research is to support the current staffing process the implementation of such a system is left out of the scope of this thesis.

One option would be to recommend people who other staffers have chosen in similar situations but gathering data for this solution would take probably near two years before the cold start problems of such a system could be overcome. As Futurice's average project allocation lasts about half a year, the data would be reliable in 3 allocation cycles meaning that 1.5 years after starting to gather the data, the first recommendations could be made. Another aspect that needs to be taken into account is that the decisions made based on queries made a year ago might not be relevant anymore in terms of interests and desires of the person. Another feasible option could be to leverage the similarity data of people's skills as Gertner et al. [23] have done and then utilize that information in searching similar candidates to the ones that are already unavailable to provide options.

4.4.2 Content-based Recommender System

Content-based recommender system has also been leveraged heavily in existing research Faerber et al. [15] use it as part of the hybrid recommender they built, so do Lu et al. [30] and Namahoot and Bückner [35] utilize content-based recommender system as the only source of their recommendations. The content-based approach seems to be exceptionally well suited for recruitment and staffing purposes as most of the existing systems already have some kind of structured or semi-structured data available. Based on the literature review the most common source of data for the recommender system was CV or similar data entry in the database of the system. This works especially well in recruitment scenarios where also the 'items' or in this case job openings have also a structured model usually with similar attributes as the CVs have. For example, Namahoot and Brückner [35] utilize the ontology that Thailand's Ministry of Labor has already defined as the basis of their content-based system where the ministry uses an ontology based on an international standard that is used to describe job positions.

For Futurice's purposes leveraging a content-based recommender makes sense. The systems that are in use collect relevant data all the time and thus keep it up to date. The data is already structured into a format that can be easily converted into meaningful vector representations with numeral scales. The current system already offers to filter based on skills but it does lack ranking though.

Content-based recommender approach share some of the problems with collaborative filtering in Futurice's case: representation of the 'user' or in this case the criteria for the perfect candidate is always first missing when the staffer starts to use the system. To battle this a plain content-based approach will not suffice and some kind of query interface will be needed. The rotation part of the recommender system is probably the closest to plain content-based recommender system as it uses already present items as the basis of the recommendations.

4.4.3 Knowledge-based Recommender System

Knowledge of how staffing works inside Futurice is one of the key aspects to produce a useful recommender system for it. In recruitment as well as in internal staffing handling knowledge like how different skills relate to each other or how different query terms should be weighted. Isias and Casaca [25] have used Case-based reasoning as part of their hybrid recommender system for Portugal State's financial administration's resource management tool. Knowledge-based approach also sits well with the idea that the system has a query-based interface [8]. Many aspects that are at the moment used in staffing or recruitment need a professional evaluation of their significance to the staffing or recruitment process already during the implementation of the system.

For the case Futurice, at least some basic principles behind knowledge-based recommendation systems will be used. When implementing the system opinions and conventions of the staffers and current process can be transformed into requirements, limits, and weights. Information like how many absence days can exist during a month-long project before it becomes a problem for the client can be baked into the filtering process. This information has been extracted from the staffers themselves through the interviews.

4.4.4 Utility-based Recommender System

While, according to search findings in the literature review, using utility-based recommender systems is quite rare as a basis for staffing or recruiting recommendations it is not unheard of. For example, Barreto et al. [3] use utility functions as part of their constraint-based recommender system for staffing a software project. In their implementation utility functions offer variations of what should be optimized. Depending on the function chosen their system can propose teams that are the cheapest to teams that have the most experience inside the restrictions the user has defined.

For Futurice use, utility functions are probably over-engineering as the amount of possible suitable candidates is usually within the range of tens. Still some aspects of utility functions can be used during the staffing process. For example, the importance weights that MAUT [13] uses can be collected during the query building phase and later used during ranking or filtering. For the most part, these weights could be inferred from the experience requirements of the query. The problem with utility functions for this case use is the availability of the information related to the project being staffed. Optimization possibilities related to prize or old projects with similar technology stacks are in the current Futurice environment impossible as there is no data about which project use which tech. As for the pricing optimization each project is sold separately and the old project allocation might not implicate the real price of the person at hand.

4.4.5 Constraint-based Recommender System

Constraint-based recommender systems in staffing software projects have been proposed by Barreto et al. [3], with a contained study involving university staff. Their approach to staffing is to handle each problem as an optimization problem with each aspect of software project optimization having its own utility function as mentioned in Section 4.4.4. Modeling an employee's attributes in a way that constraint-based system could produce meaningful results is not an easy task and certain fluffiness when handling such data can be useful, which can be achieved with the utility functions and optimizations.

Parts of the current staffing flow in Futurice are already constraint-based. When searching for a candidate staffer usually has some strict restrictions regarding the location where the person works, the candidate's tribe, how much availability the candidate needs to have and some times even some skills are a must. Still the current constraints in Power do not offer too many options to score or order results and fine-grained ranking is missing completely.

Also, the unconventional nature of defining and coding constraint satisfaction problems might prove to be problematic for the maintenance and upkeep of the software. As constraint satisfaction problem solvers are quite rare in the current state of software development.

4.5 Free Text Analysis

The staffing comments mentioned in Section 3.2.2 provide interesting extra information about the candidates and to utilize them a way of analyzing natural language is needed. Based on an extensive look into the comments that have been submitted they seem to include technologies and frameworks that people have either worked with in the past or would want to work with next as can be seen from this example:

> "Previous experience with frontend development: Node.js, React, Redux, Typescript, Webpack, Docker. Preferences for next project: Something AWS, Something Scala, Some backend project (preferably not lead role yet), Some Futulabs/smart office improvements if on bench, IoT projects, Can do frontend too (e.g. Elm), AR/VR projects, Would be nice to choose tech stack."

As people do not tend to mention things they would not be willing to do, there is no need to analyze the intention of the text itself and get away with just a keyword-based search. That makes a TF-IDF based bag-of-words approach good enough for at least the current situation. As nothing prevents people to list unwanted techs or frameworks in the future, the preference or skill information acquired from the staffing comments can become quite unreliable. That has to be taken into account when aggregating the free text search results as part of the results. Another aspect that needs to be taken into account is that not all of Futurice's employees have submitted a staffing comment and not all of them are as in-depth as others. The comments also include information that is not tied to skills or frameworks like preferred clients or preferred areas of industry that people would like to work with next. While this information will be ignored during the recommendation calculations it will provide good extra info for the staffers during the decision making. The comments are hidden in Power's default view as seen in Figure 4 but many have expressed a need to access them easily in staffing situations.

4.6 Ranking

As the system produces three different sets of recommendations (one from skills, one from the free text and one from BB3K results) it needs a scheme to aggregate and rank the outputs based on their trustworthiness. As mentioned in Section 3.2.5 Power does not offer any kind of ranking and BB3k which does, is not reliable enough. As the recommendation system will be a combination of constraint-based pre-step with nearest neighbor powered knowledge-based recommendation part the aggregation of the results has to happen after the recommendation phase of the pipeline. There exists two different approaches to combining the results as visualized in Figure 10: Method 1 would calculate the ranking of all the people from the free-text, BB3k and skill point of view separately then take all the scores of those individual recommendations of each people, calculate a weighted average from them to find out the final ranking. The other approach, Method 2, would calculate the scores and ranks from comments and BB3k, pass the scores to the person vectors as features. Then just use weighted Minkowski as the metric for the Nearest Neighbours to negate the predominance of the number of skills that are in the query.

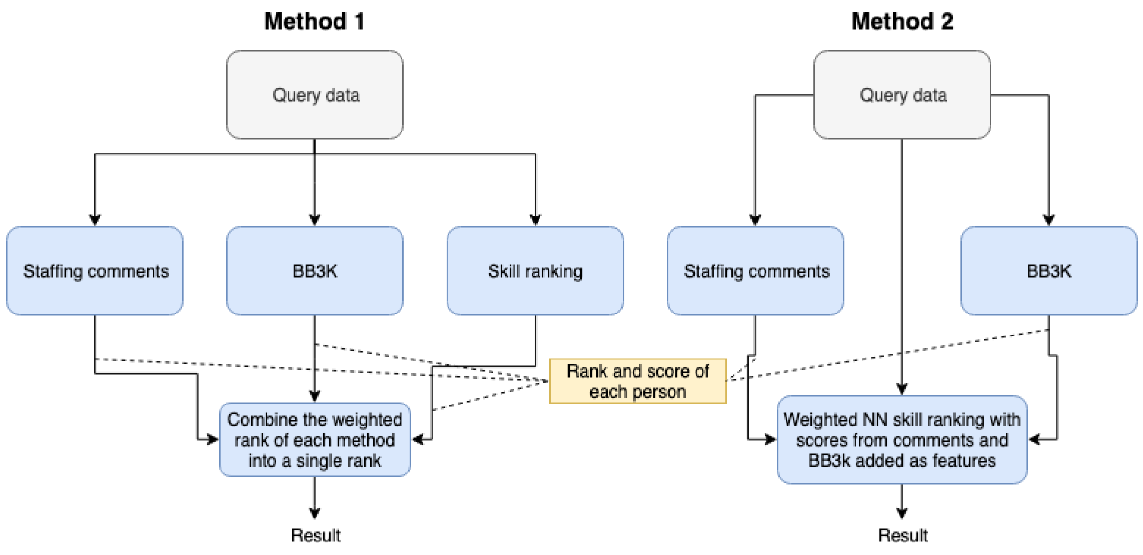

Method 1
Query data
Staffing comments
BB3K
Skill ranking
Combine the weighted rank of each method into a single rank
Result
Rank and score of each person
Method 2
Query data
Staffing comments
BB3K
Weighted NN skill ranking with scores from comments and BB3k added as features
Result

Figure 10: With three sources of rankings, this figure shows the two possible approaches to aggregating the results into on rank and score.

From these two methods, method 1 provides an easier and more flexible approach to balancing the weights as the weights are subject to change from time to time when the underlying systems or content in them changes. This approach also eases the MAUT weighting of the query parameters as they are separated from the weighting of entire recommendations.

5 Implementation

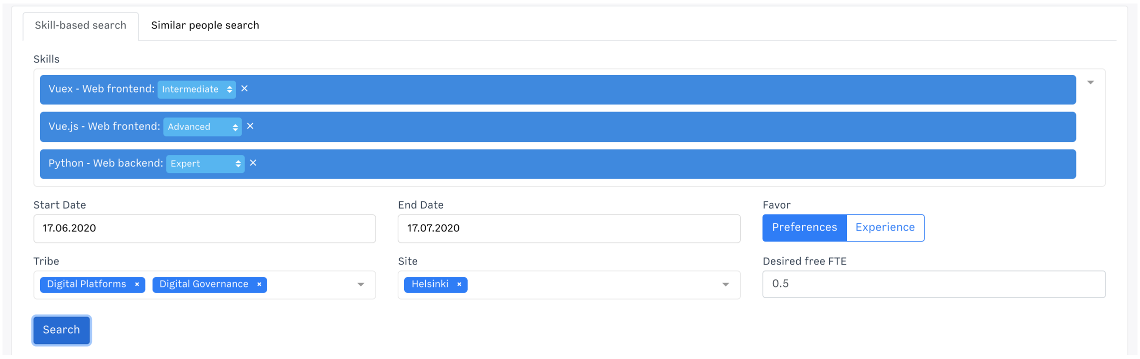
The first research question was "how to streamline Futurice's project staffing process with a recommender system" and this section is aimed to answer that. Here one possible way of implementing a recommendation system to aid out staffing process is introduced and how it was implemented as part of Futurice's internal personnel resource management tool called Power. The implementation was done in two parts, first a quick proof of concept was done completely separate from Power and once the main functionalities were proved to be working it was introduced to Power as part of the staffing toolset. The PoC was mostly aimed to iron out the kinks in the data ingesting, recommendation results and general back-end performance and not as much in terms of UI or usability. Once the PoC was proved to be viable the functionality was merged into Power and presented to users as beta release to be tested.

5.1 Front-end

As mentioned above the long term plan was to integrate the UI as part Power so the PoC front-end was implemented as a web-page heavily leveraging Vue.js framework as Power is built on it. Due to this shared framework behind, a good amount of parts from PoC front-end implementation could be transferred to Power without having to re-write too much code. The rudimentary PoC implementation of the query interface can be seen in Figure 12 where the query interface contains two parts, the constraint section and the utility/knowledge section. 'Sites'-field allows to define which sites are allowed locations for the candidates and 'Tribes'-field acts as a similar filter o tribes. The site rule can be relaxed with 'Relocation also ok' -tickbox, which also allows results from other sites and tribes if the candidate has expressed that they are willing to move to a different city or even country for their next project. Below the site and tribe filters is the desired FTE filter. It defines the minimum availability that the presented candidates must have during the whole allocated period, which can be defined under FTE with 'Start Date' and 'End Date'. The final UI, seen in Figure 11, which was integrated as part of Power contains largely the same inputs as the PoC with few differences in the skill input itself as well as the addition of selector for weighting scheme of the results and removal of Relocation selector.

Most of the constraints are carried over from old systems like Power and staffing habits and are treated as hard constraints except for the time frame which was also had a focus question in the initial interviews, in appendix A. Hard constraints mean that they act as a pre-filters and not as part of the recommendation flow and filter possible candidates out of the recommendation pool before the recommendations themselves are calculated.

First input in the upper left corner in Power UI (Figure 11) or lowest input in PoC Figure 12) is the skill input interface which offers a type-ahead selection box for adding skills into the query. With the typeahead functionality, the user can select



Skills

Vuex - Web frontend: Intermediate

Vue.js - Web frontend: Advanced

Python - Web backend: Expert

Start Date: 17.06.2020

End Date: 17.07.2020

Favor: Preferences | Experience

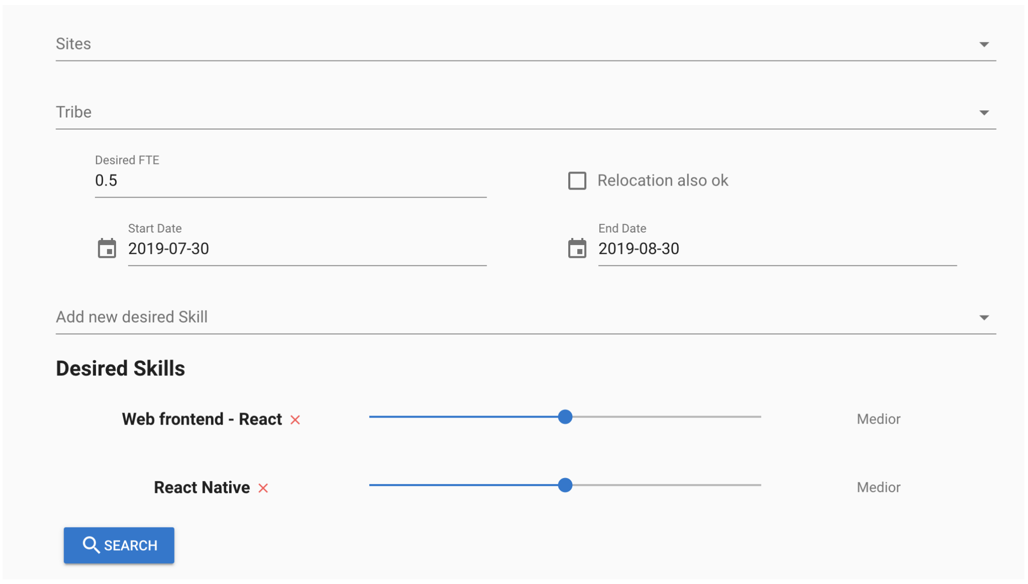
Tribe: Digital Platforms, Digital Governance

Site: Helsinki

Desired free FTE: 0.5

Figure 11: The integrated query interface in Power with the skill-based interface active.

the desired skills that the candidate should have. These selected skills act as the driving force for ranking the candidates for recommendations. In both versions of the UI, the skill selector has a selector for the desired experience for the competence selected. As Von Winterfeld and Edwards [13] when defining MAUT state that the scales should be natural units of measurement so the used range in defining the desired seniority is the same that is used in Power to express proficiency in skills. So the UI has a scale from 1 to 5 with each value corresponding to a list of experience terms used in Power: Novice, Beginner, Intermediate, Advanced, and Expert. Once the user has determined the filter conditions, all the skills that are needed and the amounts of experience desired clicking the search button sends the request to the backend REST-API and the results are then displayed below.



Desired FTE: 0.5

Start Date: 2019-07-30

End Date: 2019-08-30

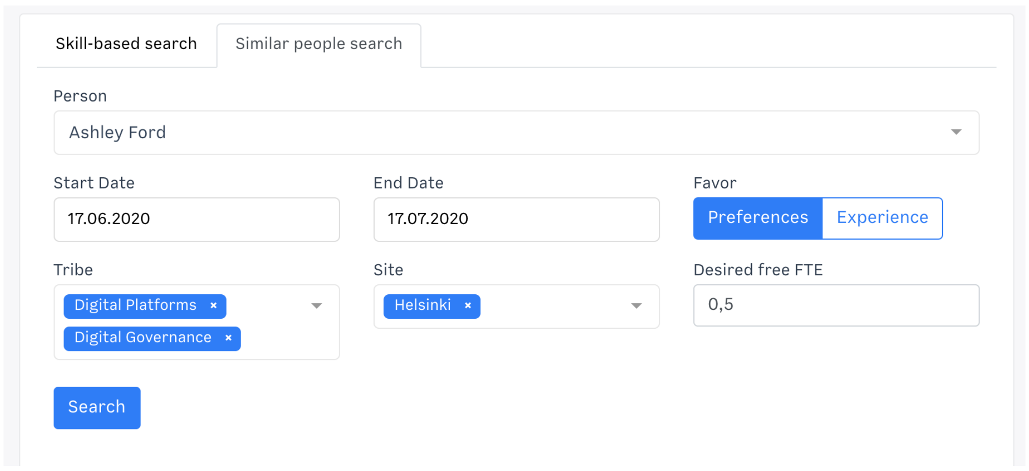
Desired Skills

Web frontend - React: Medior

React Native: Medior

Figure 12: The Futustaffer query interface in PoC.

In addition to skill-based search, that was part of the PoC, a person based search was also implemented. This functionality was implemented in entirety during the merge to the production environment and as such did not exist in the PoC version. The UI itself has all the same components as the skill-based UI has except the skill selector has been replaced with an auto-complete input with all the people of the company acting as the range of selection as can be seen in Figure 13. The idea is simply to offer a way to easily find available people that are similar to the person selected in the query as possible to combat the problems of finding suitable replacements for people wanting to rotate out of projects.

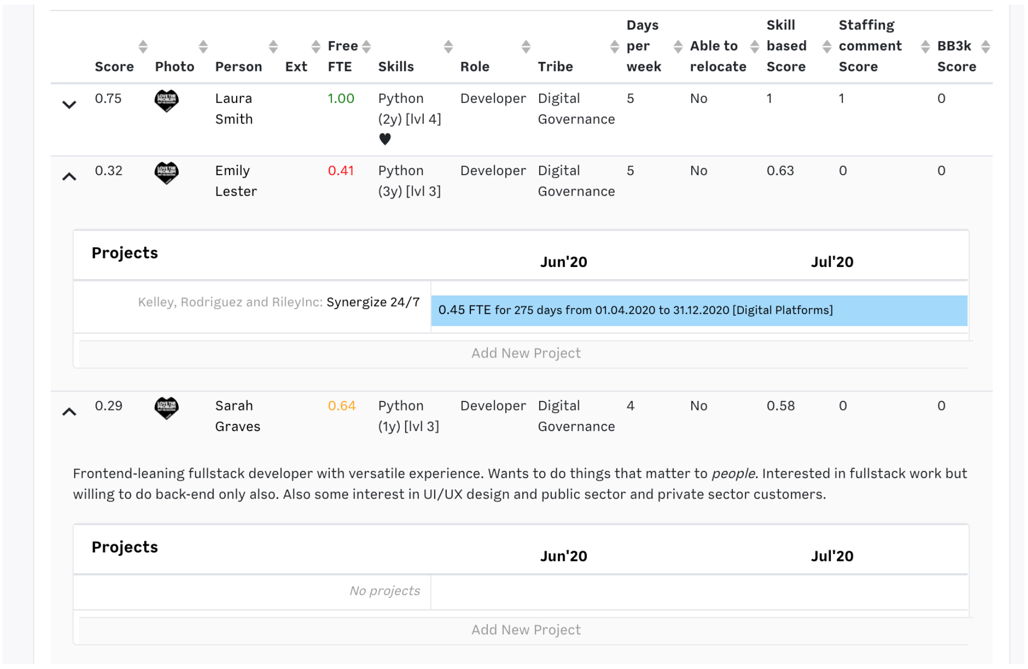


Person: Ashley Ford
Start Date: 17.06.2020
End Date: 17.07.2020
Favor: Preferences
Tribe: Digital Platforms, Digital Governance
Site: Helsinki
Desired free FTE: 0,5

Figure 13: The person query interface in Power.

The results are displayed in table format under the query interface as displayed in Figure 14. The listing shows the person's aggregated score for the query, small photo, name, indicator if the person is an external contractor, free FTE for the selected time, top 5 skills from the query ordered by experience. Additionally, it shows the person's role, Business unit (still named tribe in the Figure 14), how many days per week the person works, person's relocation status, the individual scores for the three data sources: Power skills, Power staffing comments, and the BB3K score. Each row can be also expanded to see the person's full staffing and desires comment and timeline inside the range chosen in the query. The timeline itself displays all the projects that the person has been allocated during that range in blue, and all proposed allocations as yellow. As the purpose of the timeline is to offer an easy glance check to the person's availability it also includes any absences the person might have during the selected date range.

The individual scores are only displayed during the beta phase of the functionality and once the correct weighting can be figured out, they can be left out of the results. Mainly the weighting depends on the reliability of the results that each data source produces, which will need user input. The "Favor"- weighting selector is closely related to this input as it in its current form it just weights the individual results differently.



	Score	Photo	Person	Ext	Free FTE	Skills	Role	Tribe	Days per week	Able to relocate	Skill based Score	Staffing comment Score	BB3k Score
⌄	0.75	♥	Laura Smith		1.00	Python (2y) [lvl 4] ♥	Developer	Digital Governance	5	No	1	1	0
⌃	0.32	♥	Emily Lester		0.41	Python (3y) [lvl 3]	Developer	Digital Governance	5	No	0.63	0	0

Projects	Jun'20	Jul'20
Kelley, Rodriguez and RileyInc: Synergize 24/7	0.45 FTE for 275 days from 01.04.2020 to 31.12.2020 [Digital Platforms]	

Add New Project

	Score	Photo	Person	Ext	Free FTE	Skills	Role	Tribe	Days per week	Able to relocate	Skill based Score	Staffing comment Score	BB3k Score
⌃	0.29	♥	Sarah Graves		0.64	Python (1y) [lvl 3]	Developer	Digital Governance	4	No	0.58	0	0

Frontend-leaning fullstack developer with versatile experience. Wants to do things that matter to *people*. Interested in fullstack work but willing to do back-end only also. Also some interest in UI/UX design and public sector and private sector customers.

Projects	Jun'20	Jul'20
No projects		

Add New Project

Figure 14: The results listing in Power implementation with obfuscated data.

5.2 Back-end

Due to the nature of the system Python was the most convenient choice of programming language for the back-end. Mostly because Python boasts many libraries that can be leveraged in the implementation of recommendation system, such as scikit-learn [44] for Nearest Neighbour and Classification tasks, Natural Language Toolkit [36] for stemming and vectorization of natural text and BM25 implementation [12] for free form text search and ranking. Additionally, the back-end of Power is partially implemented in Python, thus the new implementation will not bring extra baggage to technical stack. For hosting Futurice has an internal Docker [14] swarm that enables easy hosting of small services and it was used to host the recommendation system as well. With Docker, it was easy to bridge connections with existing APIs and databases without implementing proxies while keeping the system setup easy and reproducible.

Listing 1 shows the data format that is received from First when the back-end receives the query data, it constructs a query object suitable for back-end use which is depicted as the phase 1 in Figure 15. The aforementioned listing shows two optional parameters whose existence defines if the query is skill-based or person based. If the query contains person_id it will need some extra work to transform into the same format as the skill-based query is. The pre-formatting fetches the persons all skills and transforms them into the `{skill_id : proficiency}` -pairs that the skill-based search also uses. The proficiency parameter is calculated from an average between persons marked proficiency and experience values that have been

```
{
    startDate: "2020-06-22",
    endDate: "2020-07-22",
    fte: 0.5,
    preferred: "preference",
    tribes: [1,3],
    sites: [1],
    skills?: {
        66:3
    }
    person_id?: 1
}
```

Listing 1: Data structure of the query sent to back-end by the query interface. Optional values denoted by a question mark.

normalized against the company's maximum values. If either of the values is missing the defined one will act as full value for the proficiency. Also, there exists some cases where both of the values are missing and the person has just indicated that they have worked with this technology before, just either not comfortable sharing their experience levels with it or it is so long ago that they don't feel it is relevant. In those cases, the system uses a base value of half a year which is quite near the beginner levels.

Phase 2 is the role prediction which is implemented on sci-kit learn's SVM. The aim is to predict the desired role of candidates based on the given query parameters. The kernel type and relevant parameters for the kernels needed to be tuned to achieve the best performance as the training set of around 600 items with an average of 20 defined features out of 213 possible was quite small and sparse. Choosing a kernel is a task that can be eased with observing the data first to define the overall shape, thus helping to narrow down the pool of possible kernels [39, 24]. Scikit-learn's SVM implementation offers four kernel types: linear, polynomial, RBF, and sigmoid. As the data at hand is very high-dimensional, visual evaluation of it will be near impossible. Thus the testing of suitable kernel needs to be done with a wider scope. Scikit-learn offers a convenient tool for tuning SVM factors called GridSearchCV which enables an exhaustive search through user-defined parameters for most of the Sci-kit learn's ML-methods [1, 24]. After running the GridSearcCV with 5 fold cross-validation with linear, polynomial and rbf kernel, gamma $\epsilon\{0.01, 0.1, 1, 10, 100\}$, C $\epsilon\{0.01, 0.1, 1, 10, 100\}$ and degree $\epsilon\{1, 2, 3, 4\}$. The best configuration was found to be $\{C : 10, gamma : 0.1, kernel :' rbf'\}$ producing a 70% training accuracy which in itself is not enough to give reliable results. Fortunately, the SVC-instance provides a `predict_probability`-function which given a vector, it gives a probability of that vector belonging to each class present in the training set. The result of the probability function can be utilized to even out the unreliability originating from the small training set. The used solution was to define a threshold probability for a single class to be included in the query. With Power at the moment having 12 different roles available to each person worst case in predictions would be a probability of 8.33% for each class. Also with manual testing using the more common use cases including skills related to developing and design indicated that the trained SVC instance usually has 80% to 90% confidence in predictions towards the correct class. With the aforementioned measures, the probability threshold for a role to be added to the filter was set to be 15% with the produced prediction always added. If the SVC comes across such rare skill combination that it isn't able to produce roles that have over 15% probability, the process is repeated with decrements of 3% to the threshold until at least 1 or ultimately all roles are included in the filter. The query parameter construction also duplicates the recommendation calculation between three different branches with a temporal difference. The initial interviews brought up a possibility that some clients would be willing to shift the start and end dates of the projects especially when they are a long time client. The objective is to offer alternatives if the given start and end date of the project could be shifted either two weeks ahead or before the given values. Consequently, query objects with start and end dates shifted with 2 weeks offsets to both direction are created to find alternative candidates and the whole recommendation process is executed with those shifted values.

After phase 2 all the constraints have been defined and the constraint-based filtering is executed during the database fetch of candidates in phase 3 of recommendation. Listing 3 is the database query with all the constraints from phases 1 and 2 inserted into the braced expressions with string formatting. Lines 10-25 find the maximum simultaneous allocation FTE amount for each person during the queried time. That number is then later used to filter people who are not available for the query at hand. In addition to allocations, information like the person's site, absence days, and of course skills are also joined into the result. Allocations are not the only thing affecting the person's availability as also longer absences can prove to be problematic especially in shorter allocations. Therefore before the database query a `absence_days_max`-value is calculated from the time span of the query with formula $max(1, 0.1 * allocation\_length\_in\_days)$. The idea is that if the person has over 10% of the proposed allocations marked as vacation or absence, it might prove too much for the client to accept. The figure of 10% itself comes from the average time a person uses inside Futurice to non-billable work during a week. The exact number will still need some tweaking and will be monitored in use or just opened up as a configurable value part of the query interface. Phase 3 also contains the tokenization of both the skill names used in the query as well as the tokenizing the staffing comments of all the people left after constraint filtering. In this case, tokenization contains splitting up words based on punctuation and also transforming it into lowercase. Once the target documents (in this case the staffing comments) are tokenized they are inputted as the corpus for the BM25 Okapi.

Phase 4 of the Figure 15 contains all the actual executions of the recommendation score data sources. They are executed in synchronous order due to the nature of the back-end but still fully isolated from one another. First, the BM25 Okapi free text

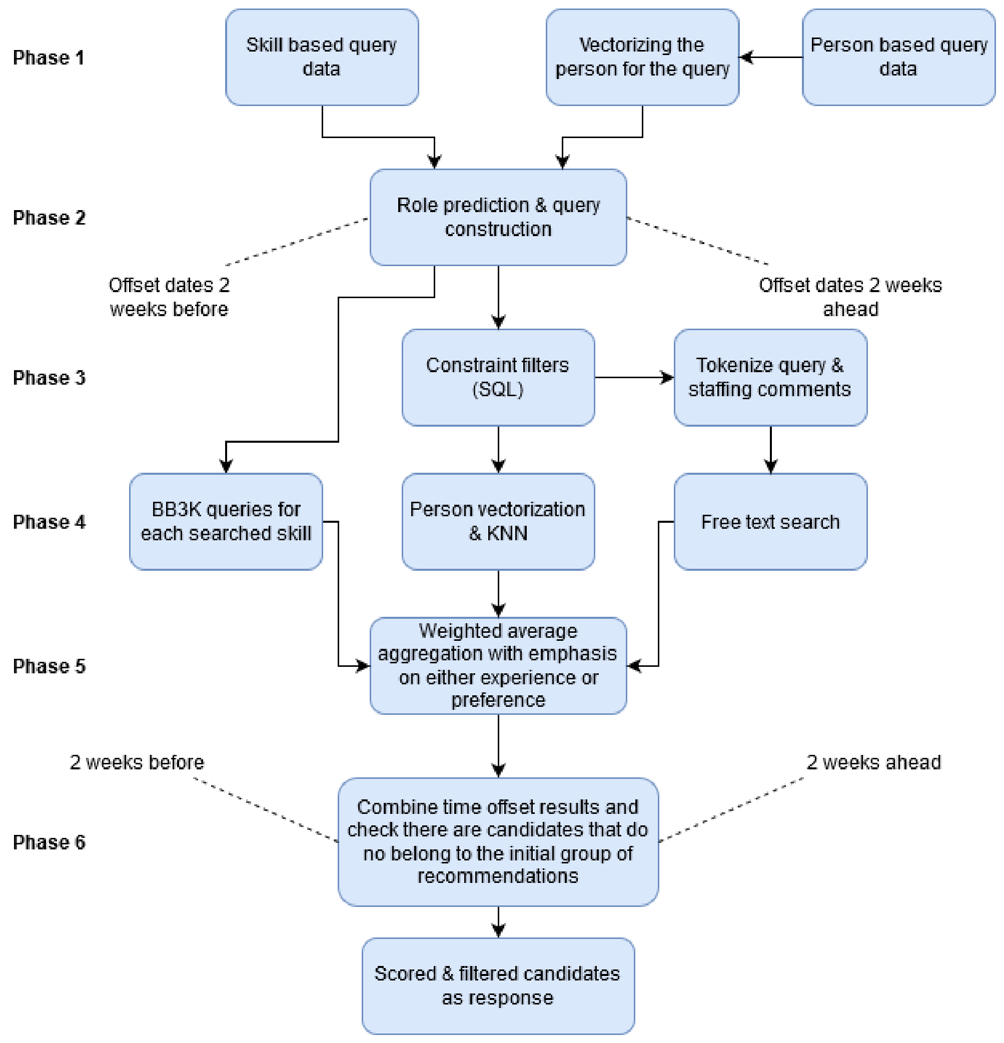

Phase 1
Skill based query data
Vectorizing the person for the query
Person based query data
Phase 2
Role prediction & query construction
Offset dates 2 weeks before
Offset dates 2 weeks ahead
Phase 3
Constraint filters (SQL)
Tokenize query & staffing comments
Phase 4
BB3K queries for each searched skill
Person vectorization & KNN
Free text search
Phase 5
Weighted average aggregation with emphasis on either experience or preference
2 weeks before
2 weeks ahead
Phase 6
Combine time offset results and check there are candidates that do no belong to the initial group of recommendations
Scored & filtered candidates as response

Figure 15: The different phases of recommendations and data flow inside the recommendation system

search is executed on the tokenized query array using the Dorian Brown's [12] which are based on the Trotman, Puurula, Burgess [46] research on the performance of different BM25 ranking methods. When the query is run it outputs a score for each "document" or in this case each person. Thanks to using the same preprocessing on each query parameter and staffing comment these scores are comparable between each other without extra steps.

After free text score calculation, the skill data is handled and the NN instance itself is built from the people obtained in phase 3 filtering. As the skills are already part of the query it is easy to discard all those skills that are not needed in the query while keeping the hierarchy of the skills in mind. This keeps the vector space clean as the zeros that would be part of the query without the filtering would produce misleading results as the query would favor those people who do not have experience

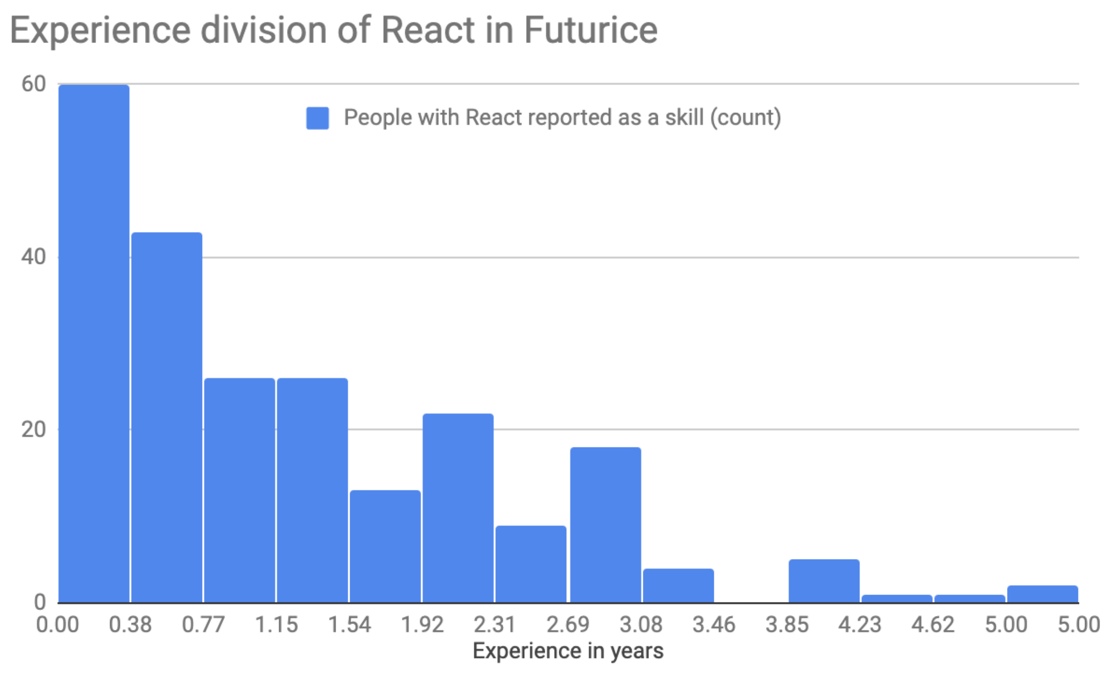


Figure 16: The skill experience distribution in years as reported in Power by Futurice employees (7/2019).

in those even though they are meaningless in the sense of the query. The vectors of each person are constructed from the experience and preference -value pairs. The value pairs for each skill are calculated with the following formula:

$$\left[\frac{\frac{x_e^i}{max(x_e)} + 2 * \frac{x_p}{5}}{3}, \; p_x^i * f \right] \quad x \in N, \; p_x^i \in [0,1], \; f \in [0.2, 0.5] \tag{8}$$

where N is the selected group of skills in the query, x_e^i is the amount of experience the person has in skill x in years, $max(x_e)$ represents the maximum experience that any one person has inside Futurice on skill x in years, x_p is the proficiency mapped to a $(1, 2, 3, 4, 5)$ range where beginner is 1 and expert 5, p_x^i denotes if the person has marked skill x as preferred with 1 representing true and 0 false and f is the weighting parameter of the preference. by The equation is weighted to prefer proficiency information as this information is consciously chosen and provided by the people inside the system and thus more accurate than pure time-based experience. Equation 8 is only valid if the person has both proficiency and experience defined for the said skill so if either of them is missing only the defined value is used. As can be seen from Figure 16 the experience distribution is heavily skewed to the year and under in the case of React, which itself is one of the most utilized technologies in projects undertaken at Futurice. The aforementioned slant towards the 0 to 0.4 years of experience can be partially explained by the number of people that have not listed any experience amounts but still, the distribution is by no means equal. Transitively this means that having the most experience on one of the skills or technologies is a trait that heavily makes you stand from the crowd. Thus those who have not shared any experience info while still having marked said skill as something they possess cannot be highly valued when defining a person's vector. Thus it will be given a small default experience of 0.2 years if no other information about the proficiency of experience of the person is not available for the skill at hand. As mentioned earlier the skills a hierarchy built inside Power where each skill has 0 to 1 parent skills usually referred to as categories, for example, React is part of the Web Fronted category. While the categories itself are quite wide they still provide some insight into each person's preference over wider areas of interest like for example whether a developer prefers front-end or back-end work. While building the NN vector space, if a skill with a parent is encountered on a person, the skills experience value is also added to the parent's experience. The aim of the chosen equation is to navigate the gaps and outdated values in Futurice experience data by averaging between all available experience sources. The equation itself was obtained in collaboration with Power developers and people managers through a testing session with different weights.

The distance function used in NN is weighted Minkowski distance with $p = 2$ also known as weighted Euclidian distance. The weights are determined during the NN build process purely by how much each value should affect the results. So the skill experience - proficiency average value gets a weight of 1, a parent category value gets a weight of 0.2 and the preference value gets either 0.5 weight when favoring preferences or 0.1 when favoring experience. This means for example that experience in a top category will impact the distance to query vector 5 times fewer than the original skill's experience value thus lowering its overall importance in the query. Finally, as the person vectors are obtained as experience value - preference boolean values they are flattened as one vector ordered by skill id and the same flattening is done on the array that contains the weight pair vectors as well. This produces a matrix with $|N| * 2$ columns and $|P|$ rows, where N is the skills desired in the query and P is the list of people obtained from the filter operation.

Finally, BB3k results are queried from the BubbleBurster 3000 server as the last part of phase 4. In the sense of this thesis, BubbleBurster 3000 is handled as a black box. Its main reason to be included in the search is to provide vision to data that is not part of Power at the moment. So the query itself is a simple GET request to appropriate URL, which in this case is inside the docker swarm:

http://bb3k_api/search?searchQuery=react&searchQuery=vue&limit=50

The search parameters will be the skills defined in the query and passed as URL query parameters. The BB3K response contains a JSON list of people with each person having a list of sources under attribute "sources" as can be seen in Listing 2. Three of the optional sources are used as part of recommendations: "flowdock", "calendar-events" and "timereports" and none of the sources are guaranteed to be in the response. "Power-skills-single" will be ignored as that data is also used to

```
[{
"id": 123,
"login": pena,
...,
"sources": [
      {
        "source": "flowdock",
        "score": 713.3883013725281,
      },
      {
        "source": "calendar-events",
        "score": 420.5162792201337,
      },
      {
        "source": "timereports",
        "score": 80.4251203135834,
      },
      {
        "source": "power-skills-single",
        "score": 15.857952117919922,
      }
    ],
    "score": 1284.5542998313904
}]
```

Listing 2: Example BB3K API response to query with keyword "react"

determine NN scores and using the single score would just give unnecessary weight to Power experience data which is not the aim here. Once the results are obtained each person's valid and defined "source-score" is added to a temporary BB3K score for that person. Every person will not receive a score from BubbleBurster 3000 every time for two reasons: firstly the 50 person limit on BB3K is much fewer than Futurice has versed employees in the most popular techs or when the selected skill is a parent category and secondly not every person is connected to the query at hand via Flowdock or Slack messages, time reports or calendar markings. On the other hand, 50 results are more than enough when implementing a recommendation system as the results probably deteriorate quite rapidly after the top 10 candidates when using FTE restrictions thus keeping the number of candidates low, to begin with.

At the start of phase 5, we have three arrays of scores amounting to 3 scores per person respectively. At this stage the scores are comparable only between the data sources but not they can not be cross-compared. For example, BB3k produces scores that are in 10^0 to 10^4 range, the NN instance has a theoretical maximum distance of $|N| * 2$ where N is the group of desired skills in the query and BM25 having a maximum score of 1 and minimum of 0 for each document. To make the scores comparable and aggregatable they need to be normalized. The most straight forward way of normalizing them is to just feature scale all the three values individually with simple min-max scale equation:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{9}$$

This action has two effects: the data becomes comparable but it will also become relative to the maximum and minimum values of each distinct data sources' score. This means that the scores can be used to rank and order people inside one specific query but the scores obtained are not comparable between queries as the smallest score of the query will always be 0 regardless of how close the last candidate was to the first candidate in pure score sense. Another point related to the aforementioned normalization of the scores is that if there are only two candidates left after the initial filtering and the other one will always have 1.0 score and the other one 0.0. Once the scores have been normalized an aggregated score can be calculated from them for each candidate. The formula used is a weighted average with different weights depending on whether the querier wants to emphasize expertise over preference or the other way around using the "favor" option in the UI. The formula used is the following:

$$s_p = \frac{w_e * s_e^p + s_c^p + w_b * s_b^p}{1 + w_e + w_b} \tag{10}$$

where s_p is the aggregated score for the person p, s_e^p is the skill experience score of person p, s_c^p is the skill comment score for same person, s_b^p is the BB3K score for that person, w_e the weight for experience score and w_b the weight for BB3K score. Weights $w_e = 3, w_b = 1$ are used for queries that favor preference as the most accurate information about the person's preferences can be found in the Power's structured data. The skill comments could be weighted also higher in this case but the free text nature introduces a good chance that people might also be listing things they do not enjoy in the comment which would skew the results quite a bit and in the worst case be recommending people for project openings which they are trying to avoid. Weights $w_e = 8, w_b = 4$ are used when the user has chosen to favor experience over preference. With aforementioned weights combined with experience-based queries assigning smaller weights to preference the query will naturally favor people who have high proficiency and experience values in queries skills but also get high scores from BB3K with said skills. BB3K scores high weighting in experience favoring queries can be explained by its data sources, either the person has talked a lot about the topics or they have reported a lot of hours with the skill mentioned in the comments or have calendar events with the topic mentioned. These all indicate that the person is very active around the topic which also indicates that the person at hand is probably quite experienced or at least has advanced knowledge about the topic.

After the scores have been normalized the whole query process is then repeated with two different time-offsets of 2 weeks forward and 2 weeks backward (if the start date is more than two weeks in future). Unfortunately, this feature is yet to be implemented in front-end and as such will be ignored in results. These temporal offset results are filtered with the people that are already in the existing results being removed which is depicted as phase 6 in Figure 15. The idea is to provide options for the users if they would move the start date of the project forwards or backward. Once the temporal offset results have been obtained the result payload is sent to the front-end and displayed there to the requester.

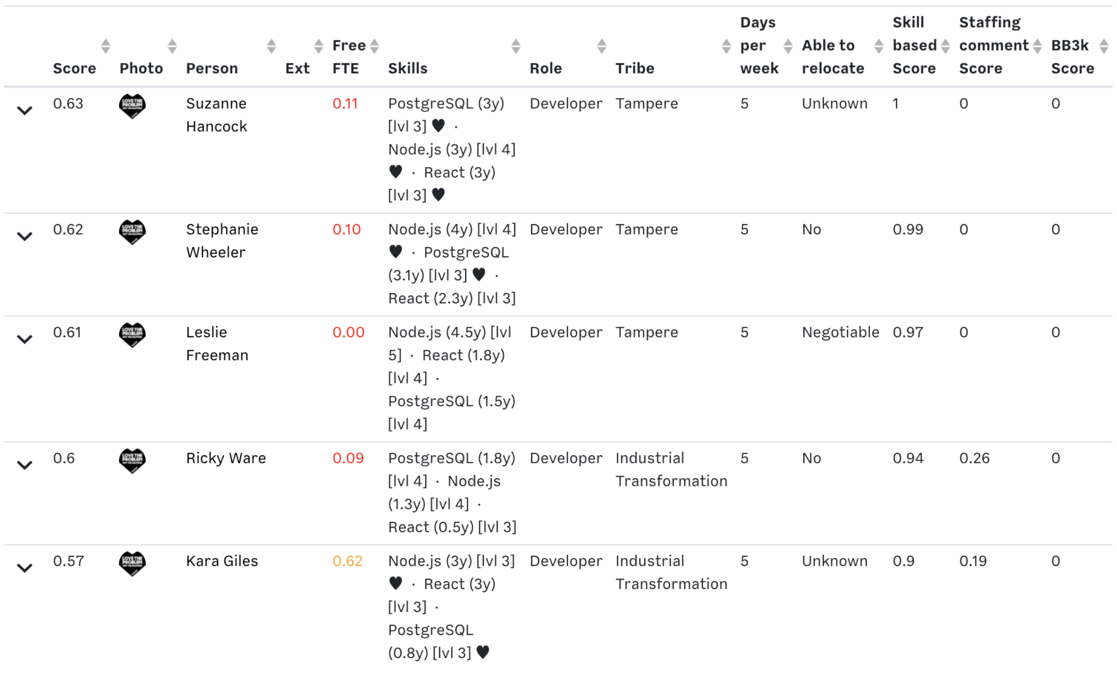
6 Results

In this section, the feasibility and accuracy of the implementation presented in Section 5 will be evaluated to answer the second research question. For the recommender system to be feasible, it would need to produce results that rank people with relevant skills higher than people without them or people further away from the desired proficiency range. Two different approaches are used to evaluate the accuracy and feasibility of the system: first two different scenarios were queried and results analyzed, secondly an interview was held with Olli Hietamies to ask an expert review on both the feasibility and accuracy of the system which is transcribed in Appendix B. Olli works as an operations manager at Futurice and is one of the most likely candidates to use the system as part of his daily operations as part of his staffing pipeline.

6.1 Skill-based Search

To get an understanding of how accurate the queries with the skill-based search are an imaginary staffing scenario was created. The scenario went as follows: we are looking for a senior developer who has a good grasp of full-stack work with knowledge in Node, React, and PostgreSQL. The interviewee described this kind of search being almost the exclusive type of search thus proving to be a good testing ground. Based on the aforementioned scenario the following query was executed with the "Favor" setting on experience: Node (Intermediate), React (Intermediate), and PostgreSQL (Intermediate). FTE restrictions were omitted in this query as the number of results with current Futurice production data was too scarce as most employees were having their vacations during the summer and the rest were generally well utilized. The vacation check did get some praise in the interview though, as it is something that gets easily overlooked during the staffing process that the interviewee has normally. The interviewee mentioned that languages the person can speak usually plays an important part in the staffing process as some clients only accept Finnish speaking consultants. Unfortunately, the language options were left out of the implementation as this aspect was completely missed during the initial interviews thus ended up being ignored as a feature.

At a glance, the results in Figure 17 seem to be reasonable in the sense of the skills that the top candidates possess and the overall accuracy of the skill-based search was backed off by the expert interview transcribed in Appendix B as well. The operations manager stated that during his testing of the system the results obtained using the skill-based search were all usable and reasonable and some even produced candidates that he could have missed if he would have used his normal staffing process. On the other hand, the interviewee raised a valid concern about the visibility of all candidates as he stated that it would be really hard to spot if somebody was missing. Fortunately, the implementation does not filter anyone out if no filters are defined except based on the role determined by the SVM so the only possible way of missing people would be complete mislabel on the role queried. The aforementioned cases have not yet been presented in the skill-based search.



	Score	Photo	Person	Ext	Free FTE	Skills	Role	Tribe	Days per week	Able to relocate	Skill based Score	Staffing comment Score	BB3k Score
⌄	0.63		Suzanne Hancock		0.11	PostgreSQL (3y) [lvl 3] ♥ · Node.js (3y) [lvl 4] ♥ · React (3y) [lvl 3] ♥	Developer	Tampere	5	Unknown	1	0	0
⌄	0.62		Stephanie Wheeler		0.10	Node.js (4y) [lvl 4] ♥ · PostgreSQL (3.1y) [lvl 3] ♥ · React (2.3y) [lvl 3]	Developer	Tampere	5	No	0.99	0	0
⌄	0.61		Leslie Freeman		0.00	Node.js (4.5y) [lvl 5] · React (1.8y) [lvl 4] · PostgreSQL (1.5y) [lvl 4]	Developer	Tampere	5	Negotiable	0.97	0	0
⌄	0.6		Ricky Ware		0.09	PostgreSQL (1.8y) [lvl 4] · Node.js (1.3y) [lvl 4] · React (0.5y) [lvl 3]	Developer	Industrial Transformation	5	No	0.94	0.26	0
⌄	0.57		Kara Giles		0.62	Node.js (3y) [lvl 3] ♥ · React (3y) [lvl 3] · PostgreSQL (0.8y) [lvl 3] ♥	Developer	Industrial Transformation	5	Unknown	0.9	0.19	0

Figure 17: The skill based query results for query React, Node and PostgreSQL.

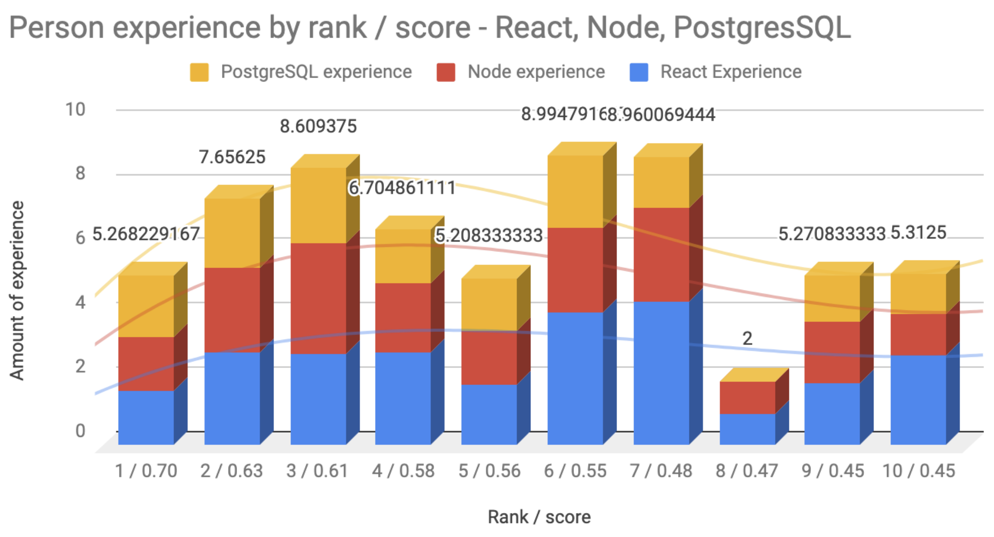


Figure 18: Averaged and aggregated experience values of the candidates with parent skill values obtained as a response to React, Node and PostgreSQL query. Experience displayed as experience units which are obtained by normalizing each skill to 0-5 range and then averaged with proficiency values mapped to same range.

To further determine the accuracy of the results an in-depth analysis of the top 10 candidates was carried out on the results produced by the Node, React, PostgreSQL query. As can be seen from Figure 18 the candidate ranked as number one has approximately under 2 units of experience for each skill which means that their average of normalized time-based and proficiency-based experience is little over 0.5 under the average. The experience selector "intermediate" in the query should represent the average time-based experience in that skill inside the company and intermediate proficiency. From Figure 17 we can see that the first ranked person named "Kelly Davis" does have intermediate (level 3) proficiency but her time-based experiences are all almost half from the company averages are Node: 1.6 years, React: 1.56 years and PostgreSQL: 1.87 years. Fortunately, the rank is not solely determined by experience but also the other two data sources contribute to it and from Figures 19 and 18 we can see that Kelly has obtained a relatively high score from BB3K which explains the ranking as first with sub-optimal experience values. When checking the following two candidates on ranks 2 and 3 we can see them being more in line with the experience values being over to 2.5 units per skill but no additional score from the other data sources.

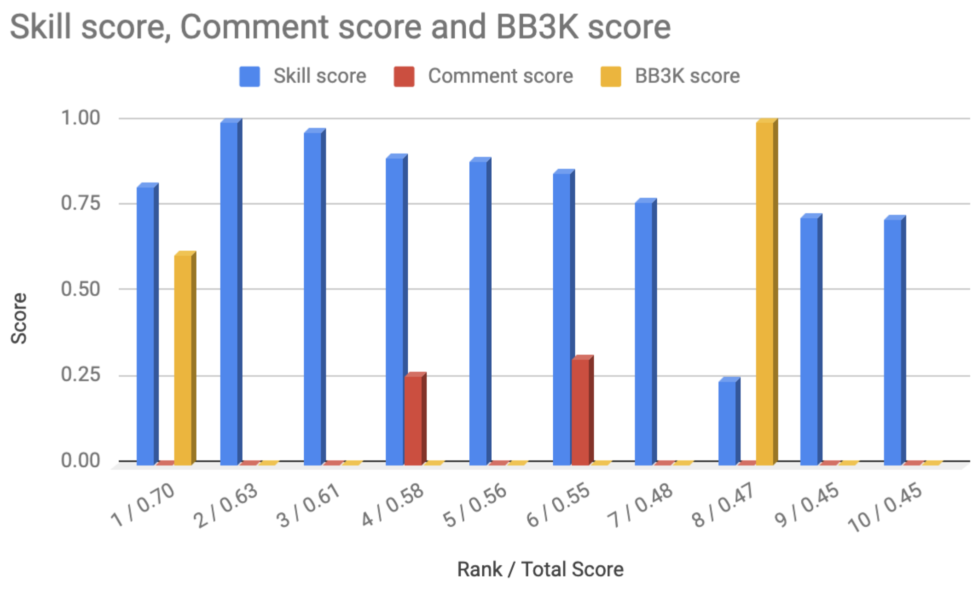


Figure 19: The score distribution of the ranks 1 to 10 obtained as a result of a React, Node and PostgreSQL query.

The trend lines seen in Figure 18 also indicate that the people recommended after rank 3 fluctuate approximately over and under the average of 2.5 experience units per skill with rank number 8 being an outlier experiencewise but can be explained with their high score from BubbleBurster 3000. Figure 19 shows also that ranks 4 and 6 have received a score based on their staffing comments and for example, the person at rank 5 had a staffing comment that started like this:

> "My most recent experience is in full-stack development using technologies such as React, Node, Python, PostgreSQL, Typescript ..."

While the person who received the highest score had a staffing comment like this:

> ... I would wish to grow as a full-stack developer. Preferred stack: React, Typescript/Javascript/Python, Node.js, PostgreSQL, AWS.

Clearly, the staffing comment scoring is working in the sense that the people who have listed the skills in their comments get a score but the scoring itself seems to be very disproportionate as in pure text processing sense there is very little difference in the two comments. The most likely reason for the aforementioned gap in the scores between the two comments is the min/max normalization that produces big differences to scores if all the scores are originally very close to each other. That normalization could be dialed down or completely dropped for the BM25 results anyway as they are already on the 0-1 scale. Still based we can determine that the staffing comment scoring is viable on paper but needs some adjustments on normalization and the interviewee stated that it is useful as the comments are the most underutilized piece of information that we have on people. This mostly happens because the comments are thought to contain only preferences that staffers themselves rarely check as the salesmen who talk with the candidates about the projects will determine if the candidate wants to work in the project they are proposing. Still, the interviewee states that it would be beneficial to start accommodating people's preferences much earlier in the staffing pipeline which the recommendation system can increase.

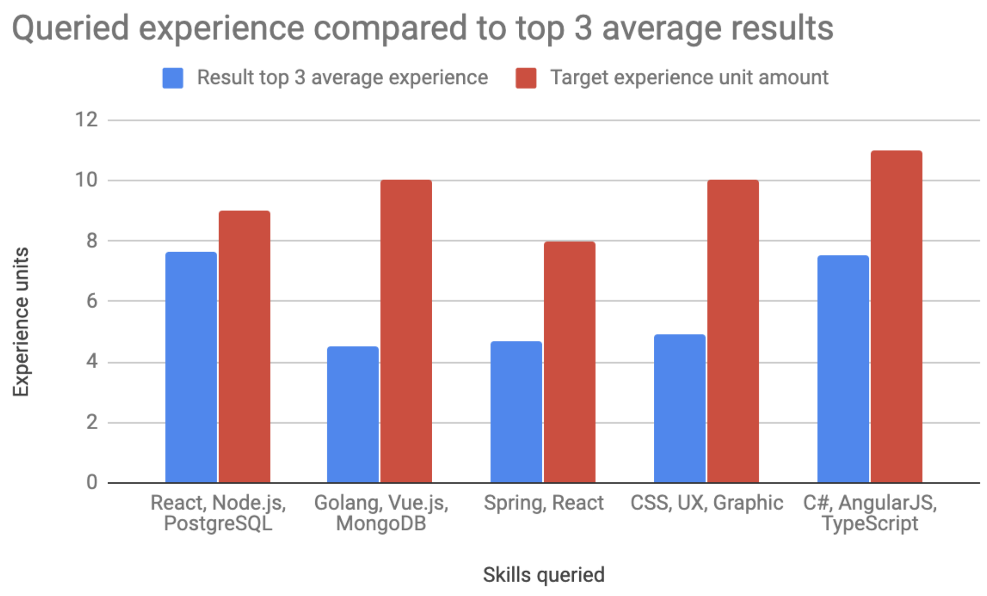


Figure 20: The comparison between queried experience and the average experience of top 3 results received.

To acquire a broader view into the accuracy of the results 4 additional, experience weighted, general queries were executed:

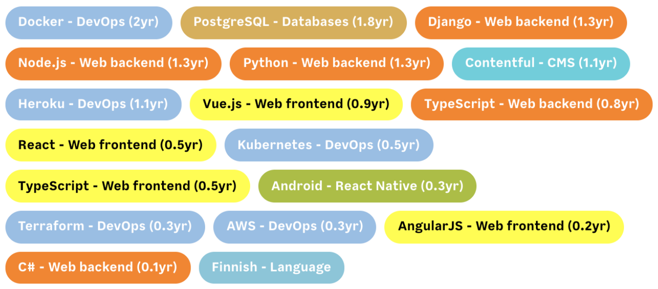
- React: Intermediate, Node.js: Intermediate and PostgreSQL: Intermediate
- Golang: Advanced, Vue.js: Advanced, MongoDb: Beginner
- Spring: Advanced, React: Advanced
- CSS: Intermediate, UX Design: Advanced, Graphic Design: Intermediate
- C#: Intermediate, AngularJS: Advanced, TypeScript: Advanced

The top 3 of each result was taken and the average of each person's experience and proficiency was added together which then were averaged over the 3 people obtained. This produces the average experience units that the top 3 results of each query got in the queried skills. The Figure 20 shows each of the obtained averages for each query compared to the requested experience levels. The figure shows that using more rare combinations like Go and Vue where the selected skills are relatively fresh and thus not likely to be found on many people. Same with Spring and React where Spring is quite old technology and React being the main front-end framework in use today. Still the results as a whole display that for each query the system can provide a person that has at least half of the experience wanted. With a relatively low candidate pool of 600 such results can be also expected. All in all the skill-based flow can be seen as a success as even the expert interviewee thought that it should be merged with the current search functionality as soon as it is deemed ready.

6.2 Person-based Search

Person search is the more experimental part of the implementation as nothing similar has been available in Power before. An imaginary sales case was also made to evaluate the accuracy and feasibility of the person search tool. As the feature was originally intended to ease project rotation, the case chosen went as follows: "We have Ricky working a customer project as a back-end developer with the project using Typescript, Node, and Docker and we want to find a suitable replacement for Ricky as he is rotating out". The query was carried out with Ricky selected as the query person and emphasis on the preferences of the candidates. Ricky's skills and experiences are displayed in Figure 21.

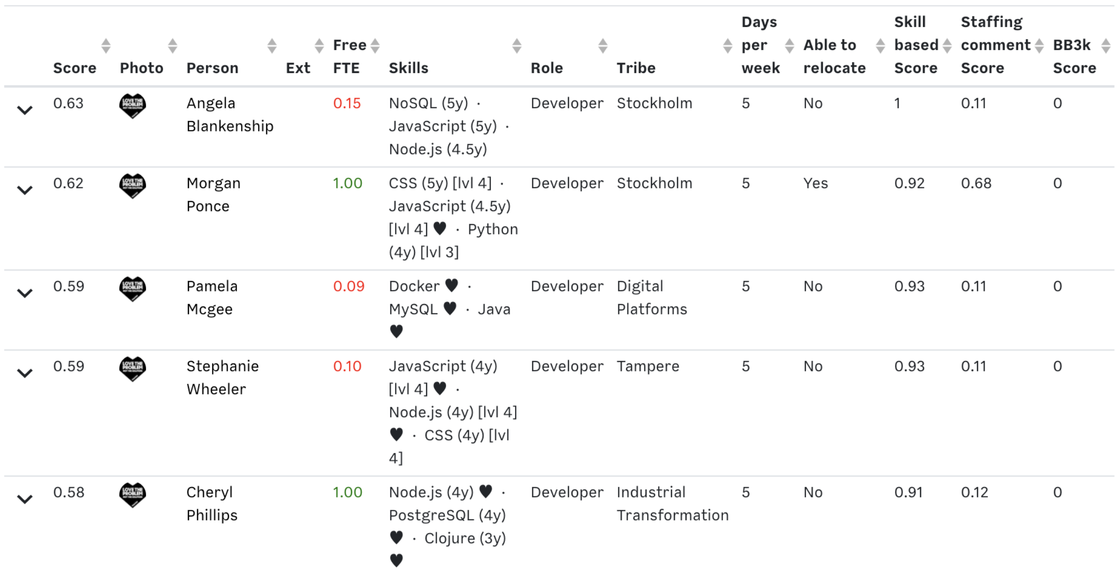
The inherent problem of the search can be seen right from the start as Ricky's skills are quite spread out on both front-end and back end as well as include multiple different programming languages not needed in the project at hand. This came up in the interview as well with the interviewee saying that the two use cases would be that the person querying does not know the group of people they are querying as well as people in operations usually know or the staffer is trying to find an as broadly experienced person as the person in the query. For him, such a case was staffing designers, as he works mainly with developer staffing. The interviewee further added



Docker - DevOps (2yr) · PostgreSQL - Databases (1.8yr) · Django - Web backend (1.3yr)
Node.js - Web backend (1.3yr) · Python - Web backend (1.3yr) · Contentful - CMS (1.1yr)
Heroku - DevOps (1.1yr) · Vue.js - Web frontend (0.9yr) · TypeScript - Web backend (0.8yr)
React - Web frontend (0.5yr) · Kubernetes - DevOps (0.5yr)
TypeScript - Web frontend (0.5yr) · Android - React Native (0.3yr)
Terraform - DevOps (0.3yr) · AWS - DevOps (0.3yr) · AngularJS - Web frontend (0.2yr)
C# - Web backend (0.1yr) · Finnish - Language

Figure 21: The skill set and experience of the person "Ricky" used in the search.

that in designer staffing, the old projects they have done, are more a deciding factor in finding similar people than skills listed in Power. Unfortunately, Power is missing a lot of information about the projects themselves and utilizing such data is impossible at the moment as it does not exist.



Score	Photo	Person	Ext	Free FTE	Skills	Role	Tribe	Days per week	Able to relocate	Skill based Score	Staffing comment Score	BB3k Score
0.63		Angela Blankenship		0.15	NoSQL (5y) · JavaScript (5y) · Node.js (4.5y)	Developer	Stockholm	5	No	1	0.11	0
0.62		Morgan Ponce		1.00	CSS (5y) [lvl 4] · JavaScript (4.5y) [lvl 4] ♥ · Python (4y) [lvl 3]	Developer	Stockholm	5	Yes	0.92	0.68	0
0.59		Pamela Mcgee		0.09	Docker ♥ · MySQL ♥ · Java ♥	Developer	Digital Platforms	5	No	0.93	0.11	0
0.59		Stephanie Wheeler		0.10	JavaScript (4y) [lvl 4] ♥ · Node.js (4y) [lvl 4] ♥ · CSS (4y) [lvl 4]	Developer	Tampere	5	No	0.93	0.11	0
0.58		Cheryl Phillips		1.00	Node.js (4y) ♥ · PostgreSQL (4y) ♥ · Clojure (3y) ♥	Developer	Industrial Transformation	5	No	0.91	0.12	0

Figure 22: The results obtained by using a person with web development background as the search parameter.

as a downside in the interview. Still, the person search is able to recommend people with similar kinds of skill sets to some extent as can be seen from Figure 23. But as can be seen from the downward trend of the graph that the system just tries to recommend people whose skills overlap the query person as much as possible with the people with the most similar skills coming as top-ranked. So, in theory, the results are as expected.

The interview also brought to light two cases where the results were plainly incorrect, both of the query people being designers. In those erroneous cases, the

SVM had labeled the roles of the queried people wrong and was only showing people from internal roles as results, which in these two designers cases was wrong. Most probably the design directors, who have been active designers earlier and thus have design skills in the system but are now in internal roles and the low amount of training data caused the SVM to struggle on some combination of design skills that the two people had. Another problem that came up during the interview was of course the language limitations, as when trying to find suitable replacements or similar people they should be able to speak the same languages as well. Unfortunately, the languages, being handled as skills in the underlying system Power, could not be kept as part of the query vector that was produced from the person used in the query as it produced too many results that only had a perfect match in spoken languages. Therefore the results did not actually reflect the technical skill of the query person enough.

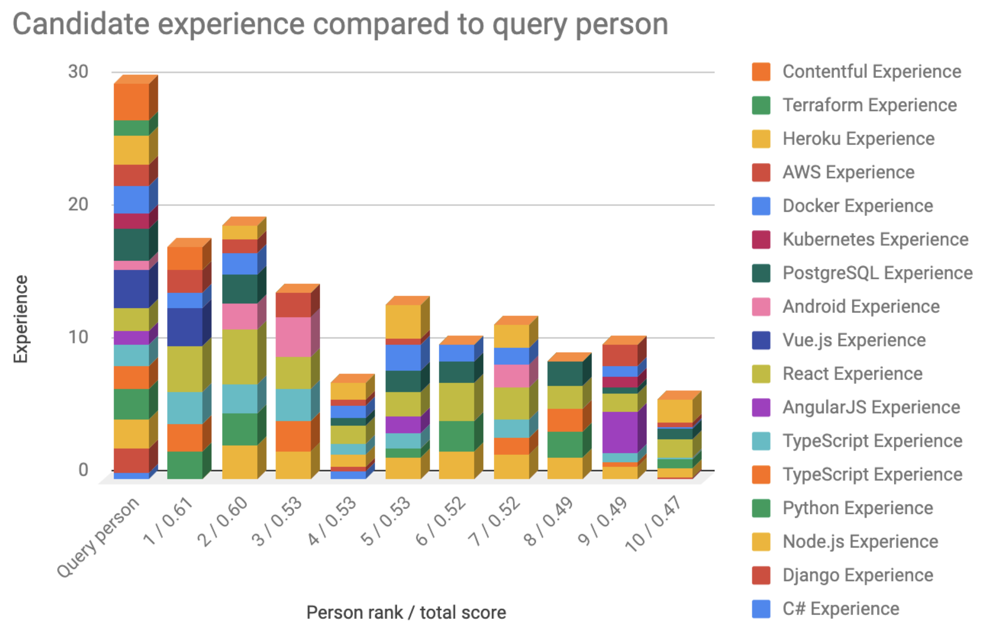


Figure 23: The experience distribution of the query person and the recommendation results as experience units, which are obtained the same way as in Figure 18.

The interviewee did not completely dismiss the idea of a people-based search. For example, he saw that this kind of search could work nicely for the salesmen who do not have such vast people knowledge inside the company as the staffers and operative people do. For them, this kind of tool could be an easy way to find people they have never met or talked with before without asking somebody first.

7 Conclusion

In this thesis, I made a case study about how to aid a digital service consultancy company called Futurice's project staffing with a recommendation system by utilizing their existing data sources and how feasible and accurate such a system would be. The main goal was to create a working recommendation system that could be integrated as part of the personnel management tool Power.

The thesis covered different recommendation system types, document retrieval, and ranking, multiple attribute utility theory, classification tasks using support vector machines, and method for finding neighbors in vector space. Related research to assisted staffing was explored to cast light on different implementations that have been implemented in the past on the topic of recommendation system assisted staffing. The company Futurice and it's internal conventions were introduced and it's staffing process analyzed through company culture and interviews. The conventions, habits, and problems unearthed in the interviews were used to weight out different technical solutions in contrast to research on the topic.

As part of the thesis, a working prototype was implemented and later merged as part of Futurice's personnel management tool. Based on the existing research and the findings revealed in the interviews the presented system consisted of two different query features, one with skills as the search basis of the query and one with people, representing all their skills, acting as the basis.

To further assess the feasibility and accuracy of the presented system an analysis of the results was conducted as well as an interview with an expert who would be using this system as part of their staffing routine. Based on the interview and analysis the system was found to be feasible in the use case it was presented and the results were generally accurate with a few exceptions occurring in person-based search.

Developing a recommender system for staffing has certainly affected my view how to approach situations with multiple data sources but also has made it clear that when trying to optimize staffing one needs to know both the existing systems that were used for the same task as well as the whole culture of the company which the staffing is done for. Also, the conventions are not always shared between all staffers as some might have their individual differing views on how it should be done.

While researching and implementing the recommender system I faced some interesting problems mostly related to transforming the existing data to a suitable format for the recommender system to consume. For example, at first it felt that the language skill, which most people have in the Power data, was skewing the results too much, but after presenting the completed system to an expert it ended up being crucial to the workflow of the staffer and one of the most requested features to implement in next patch. This brings us to the next finding which was real-world and real data testing which I found crucial to successfully implement this kind of recommendation system. I noticed that getting feedback from the end-users steers the development in the right direction. The evaluation methods you have chosen might tell you that your results are accurate but the end-users might be signaling that the whole premise of the feature is not that useful. This almost happened with the person-based search related to presumptions made of its use cases.

Another problem faced was the difficulty of validating the results. The only way to make sure that the results are what is wanted is to ask the experts who do the staffing. Ranking people on relevance to certain tech is also hard for these aforementioned expert staffers inside the company so getting definitive answers is near impossible. I found that the only reasonable way forward is continuous validation as the system is used where the users will report discrepancies faced and the weights of the recommendations are then tuned once enough feedback is obtained.

The next steps for the system would be to address the problems that arose as part of the final interview, namely including language filter support and tuning person search to utilize more information about the project that the person is currently working on. Also, a more sophisticated recommendation system implementation for the skill data could be researched to more address the weighting on skills and preferences. In the farther future, some intention analyses could be done on the staffing comments to be able to differentiate whether the person would prefer the technologies next or that they just have experience in them.

References

[1] *1.4. Support Vector Machines — scikit-learn 0.21.2 documentation.* URL: https://scikit-learn.org/stable/modules/svm.html.

[2] Chris Anderson. *The Long Tail.* 2004. URL: https://www.wired.com/2004/10/tail/.

[3] Ahilton Barreto, Márcio de O. Barros, and Cláudia M.L. Werner. "Staffing a software project: A constraint satisfaction and optimization-based approach". In: *Computers & Operations Research* 35.10 (Oct. 2008), pp. 3073–3089. ISSN: 0305-0548. DOI: 10.1016/J.COR.2007.01.010. URL: https://www.sciencedirect.com/science/article/pii/S0305054807000226.

[4] Joeran Beel et al. "Research-paper recommender systems: a literature survey". In: *International Journal on Digital Libraries* 17.4 (Nov. 2016), pp. 305–338. ISSN: 1432-5012. DOI: 10.1007/s00799-015-0156-0. URL: http://link.springer.com/10.1007/s00799-015-0156-0.

[5] James Bennett and Stan Lanning. *The Netflix Prize.* 2007. ISBN: 9781595938343. URL: https://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf.

[6] Robin Burke. *Hybrid Recommender Systems: Survey and Experiments 1.* Tech. rep. URL: https://www.researchgate.net/publication/263377228_Hybrid_Recommender_Systems_Survey_and_Experiments.

[7] Robin Burke. "Hybrid web recommender systems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 4321 LNCS. Springer, Berlin, Heidelberg, 2007, pp. 377–408. ISBN: 3540720782. DOI: 10.1007/978-3-540-72079-9{_}12.

[8] Robin Burke. *Knowledge-based recommender systems.* Tech. rep. University of California, Department of Information and Computer Science, 2001. DOI: 10.1002/2327-6924.12499. URL: https://www.cs.odu.edu/~mukka/cs795sum09dm/Lecturenotes/Day6/burke-elis00.pdf.

[9] Corinna Cortes, Vladimir Vapnik, and Lorenza Saitta. *Support-Vector Networks Editor.* Tech. rep. 1995, pp. 273–297. URL: https://link.springer.com/content/pdf/10.1007%2FBF00994018.pdf.

[10] T. Cover and P. Hart. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (Jan. 1967), pp. 21–27. ISSN: 0018-9448. DOI: 10.1109/TIT.1967.1053964. URL: http://ieeexplore.ieee.org/document/1053964/.

[11] Christian Desrosiers and George Karypis. "A Comprehensive Survey of Neighborhood-based Recommendation Methods". In: *Recommender Systems Handbook.* Springer, 2010, pp. 107–140. URL: https://link.springer.com/chapter/10.1007/978-0-387-85820-3_6.

[12] *dorianbrown/rank_bm25: A collection of BM25 algorithm variants.* URL: https://github.com/dorianbrown/rank_bm25.

[13] Ward Edwards and Detloff von Winterfeldt. *Decision analysis and behavioral research.* Cambridge University Press, 1986.

[14] *Enterprise Container Platform | Docker.* URL: https://www.docker.com/.

[15] Frank Faerber et al. *An Automated Recommendation Approach to Selection in Personnel Recruitment.* Tech. rep. 2003. URL: http://aisel.aisnet.org/amcis2003/302.

[16] Kim Falk. *Practical Recommender Systems.* 2019, p. 432. ISBN: 9781617292705. URL: https://learning.oreilly.com/library/view/practical-recommender-systems/9781617292705/.

[17] Alexander Felfernig and Robert Burke. "Constraint-based recommender systems". In: *Proceedings of the 10th international conference on Electronic commerce - ICEC '08.* New York, New York, USA: ACM Press, 2008, p. 1. ISBN: 9781605580753. DOI: 10.1145/1409540.1409544. URL: http://portal.acm.org/citation.cfm?doid=1409540.1409544.

[18] Alexander Felfernig et al. "Automated debugging of recommender user interface descriptions". In: (). DOI: 10.1007/s10489-007-0105-8. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.330.4922&rep=rep1&type=pdf.

[19] Alexander Felfernig et al. "Developing Constraint-based Recommenders". In: *Recommender Systems Handbook.* 2010, pp. 187–212. ISBN: 9780387858197. DOI: 10.1007/978-0-387-85820-3. URL: www.springer.com.

[20] R. A. Fisher. "The Use of Multiple Measurements in Taxonomic Problems". In: *Annals of Eugenics* 7.2 (Sept. 1936), pp. 179–188. DOI: 10.1111/j.1469-1809.1936.tb02137.x. URL: http://doi.wiley.com/10.1111/j.1469-1809.1936.tb02137.x.

[21] Geoffrey W Gates. *The Reduced Nearest Neighbor Rule.* Tech. rep. URL: https://sci2s.ugr.es/keel/pdf/algorithm/articulo/gates1972.pdf.

[22] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. *Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity.* 2010. ISBN: 9781605589060. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.464.8494&rep=rep1&type=pdf.

[23] Abigail Gertner, Susan Lubar, and Beth Lavender. *Recommendations to Support Staffing Decisions.* Tech. rep. URL: https://pdfs.semanticscholar.org/73fa/5a0198f6790d4101edc01d077b19205eebc0.pdf.

[24] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. "A Practical Guide to Support Vector Classification". In: (2003). DOI: 10.1007/s11119-014-9370-9. URL: http://www.csie.ntu.edu.tw/~cjlin.

[25] Pedro Isaias, Cristiane Casaca, and Sara Pifano. "Recommender Systems for Human Resources Task Assignment". In: *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. IEEE, 2010, pp. 214–221. ISBN: 978-1-4244-6695-5. DOI: 10.1109/AINA.2010.168. URL: http://ieeexplore.ieee.org/document/5474702/.

[26] Sylvia J. T. Jansen. "The Multi-attribute Utility Method". In: *The Measurement and Analysis of Housing Preference and Choice*. Dordrecht: Springer Netherlands, 2011, pp. 101–125. DOI: 10.1007/978-90-481-8894-9{_}5. URL: http://www.springerlink.com/index/10.1007/978-90-481-8894-9_5.

[27] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*. 1979. DOI: 10.1109/TSMC.1979.4310245.

[28] S Sathiya Keerthi and Chih-Jen Lin. *Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel*. Tech. rep. URL: https://www.mitpressjournals.org/doi/pdf/10.1162/089976603321891855.

[29] Vipin Kumar. "Algorithms for Constraint-Satisfaction Problems: A Survey". In: *AI Magazine* 13.1 (Mar. 1992), pp. 32–32. ISSN: 2371-9621. DOI: 10.1609/AIMAG.V13I1.976. URL: https://www.aaai.org/ojs/index.php/aimagazine/article/view/976.

[30] Yao Lu, Sandy El Helou, and Denis Gillet. "A recommender system for job seeking and recruiting website". In: *Proceedings of the 22nd International Conference on World Wide Web - WWW '13 Companion*. New York, New York, USA: ACM Press, 2013, pp. 963–966. ISBN: 9781450320382. DOI: 10.1145/2487788.2488092. URL: http://dl.acm.org/citation.cfm?doid=2487788.2488092.

[31] H. P. Luhn. "A Statistical Approach to Mechanized Encoding and Searching of Literary Information". In: *IBM Journal of Research and Development* 1.4 (Apr. 1957), pp. 309–317. ISSN: 0018-8646. DOI: 10.1147/rd.14.0309.

[32] Yuanhua Lv and Chengxiang Zhai. *Adaptive Term Frequency Normalization for BM25*. 2011. ISBN: 9781450307178. URL: http://sifaka.cs.uiuc.edu/~ylv2/pub/cikm11-adptTF.pdf.

[33] Alan K Mackworth. *Consistency in Networks of Relations*. Tech. rep. URL: https://www.cs.ubc.ca/~mack/Publications/AI77.pdf.

[34] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. "Introduction to Information Retrieval". In: (2010).

[35] Chakkrit Snae Namahoot and Michael Brückner. *Standard-Based Bidirectional Decision Making for Job Seekers and Employers*. Tech. rep. 2017, pp. 10–20. DOI: 10.1007/978-3-319-66805-5. URL: http://www.springer.com/series/7409.

[36] *Natural Language Toolkit — NLTK 3.4.4 documentation*. URL: https://www.nltk.org/.

[37] Michael J. Pazzani and Daniel Billsus. "Content-Based Recommendation Systems". In: *The Adaptive Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 325–341. DOI: 10.1007/978-3-540-72079-9{_}10. URL: http://link.springer.com/10.1007/978-3-540-72079-9_10.

[38] Pearl Pu. *Decision Tradeoff Using Example-Critiquing and Constraint Programming*. Tech. rep. URL: https://link.springer.com/content/pdf/10.1023%2FB%3ACONS.0000049205.05581.24.pdf.

[39] Sebastian Raschka. *How to Select Support Vector Machine Kernels*. URL: https://www.kdnuggets.com/2016/06/select-support-vector-machine-kernels.html.

[40] Paul Resnick and Hal R. Varian. "Recommender systems". In: *Communications of the ACM* 40.3 (Mar. 1997), pp. 56–58. ISSN: 00010782. DOI: 10.1145/245108.245121. URL: http://portal.acm.org/citation.cfm?doid=245108.245121.

[41] S E Robertson et al. *Okapi at TRECC3*. Tech. rep. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/okapi_trec3.pdf.

[42] Badrul Sarwar et al. *Item-Based Collaborative Filtering Recommendation Algorithms*. Tech. rep. 2001. URL: http://www.ra.ethz.ch/cdstore/www10/papers/pdf/p519.pdf.

[43] Michael Scholz et al. "Measuring consumers' willingness to pay with utility-based recommendation systems". In: *Decision Support Systems* 72 (Apr. 2015), pp. 60–71. ISSN: 01679236. DOI: 10.1016/j.dss.2015.02.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167923615000263.

[44] *scikit-learn: machine learning in Python — scikit-learn 0.21.3 documentation*. URL: https://scikit-learn.org/stable/index.html.

[45] Karen Spärck Jones. *A statistical interpretation of term specificity and its application in retrieval*. 1972. DOI: 10.1108/eb026526.

[46] Andrew Trotman, Antti Puurula, and Blake Burgess. "Improvements to BM25 and Language Models Examined". In: (2014). DOI: 10.1145/2682862.2682863. URL: http://dx.doi.org/10.1145/2682862.2682863.

[47] Geoffrey I. Webb et al. "Lazy Learning". In: *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2011, pp. 571–572. DOI: 10.1007/978-0-387-30164-8{_}443. URL: http://www.springerlink.com/index/10.1007/978-0-387-30164-8_443.

[48] Markus Zanker, Markus Jessenitschnig, and Wolfgang Schmid. "Preference reasoning with soft constraints in constraint-based recommender systems". In: *Constraints* 15 (2010), pp. 574–595. DOI: 10.1007/s10601-010-9098-8. URL: https://link.springer.com/content/pdf/10.1007%2Fs10601-010-9098-8.pdf#page=21.

A Initial interviews

This appendix is the initial semi-structured interviews that were used to shed more light on the staffing process and problems inside Futurice. The answers have been translated from Finnish to English and some parts of them have been left out to comply with client NDA's.

How does your usual staffing workflow go?
Jetro: First step ask for more information from the customers or (Futurice sales) people coming with customer cases. Usually the case is that first we (Futurice) only know if the need is for front-end developer or back-end developer. It is important to know what kind of limitations are connected to the case. Is a certain spoken language needed, what are the tech frameworks that are required by the client, when does the project start, are we going to use customer premises? Then I'll ask has the person checked Power for available people and the also check my self if they have missed something. But Power is the main tool I use for staffing but also it is possible that I might know straight away the perfect fit for the project without consulting any systems. (After filtering people in power) I usually just browse through the results and keep an eye on the timelines and who is available.
Pette: Usually we have two cases: One where we get an RFP or we are already part of defining the problem. The more preferable situation is that we're part of defining the problem from the start as it allows us to steer and affect the solution also into more suitable format for us. Because in those cases we can think from the perspective that we have these people available, how can we utilize their expertise so that it also benefits the customer. Which is much better than the case of RFP's which dictate the problem that needs to be solved or in worst cases the solution that has to be implemented. When we can affect the problem specification, the most defining property is that how well we know the existing team and customer. Then we will look for the best fit for the team so that it brings some new supportive expertise or "fills a hole" in the existing team. For example we have an customer which has 5 developers in a certain team with 2 of them coming from outside of Futurice and we knew before summer that both of them are going to leave the project with one of them being the most senior developer with that project stack. So we wanted to offer them developers that are not the most senior in the sense of technologies but a person that has a lot of universal experience in the field, a person that can remain calm even in hairy situations. With the idea being that the next senior techwise person not leaving can take the techlead of the team while the experienced newcomer acts as a calming presence. In these cases the personality and dynamics matches take a lead with the competence fit coming second. In these cases we can also offer the most opportunities for person expertise development. Also even gender balances can be taken into account in these scenarios, just to make the teams more balanced. These kinds of situations are called rotations. In a case of RFP, where somebody has already defined what is going to be done and might even dictate which kinds of techs or frameworks are going to be used. These might involve a bidding competition which means that the client is going to interview all candidates for that single position only.

In all simplicity in those cases the staffing flow is that we just try to find best fit competence and experience -wise for that position using the tools that we have. Of course we try to challenge the customer and pry more information. In these cases the point is usually to get in a customer relationship with the customer and then start to challenge the way they do things and offer alternative more to direction of the first approach. The things that RFPs (when searching in our internal tools) usually are after are keywords, techs and experiences, age (as in experience in industry) and personal references with emphasis on the industry of the references.

Is some aspect, service or feature missing from current system in regards of staffing
I. from the staffers point of view?
Jetro: Data could be much richer and more up to date. Also possibility for people to tell which technologies they don't want to work with. The lack of this has already caused some people to delete some of their skills from the system just to avoid being proposed to do it. Power has come a long way from what it initial was, not yet perfect though. Unfortunately common is that people have forgotten to either mark that somebody has been allocated already into a project or somebody has forgotten to mark their holidays into our system. These kinds of false positives make me second guess the data.
Pette: What we would need to have (preferably as public information) is have references of all people (which projects and for whom) and the technologies used in the projects (not as universal). Person references which means that somebody (from client side) knows that our guys know what they are doing and that person can be contacted or can write a actual reference for us to use (for example in LinkedIn). In general I think we should get our individiuals more to front instead being this blob of manpower called Futurice. The other side that we should have more information about what people want to do next. Two times a year (interval on preference updates) is not enough to actually know what the person wants right now. So collecting this information more often and in a more structured manner would be beneficial. Being it tech experience, client industry, project position or roles. We only have the tech preferences now but would need the other aspects as well. The more senior the person is, the understanding of the industry plays a more bigger part, so knowing about these preferences and experiences should be more in light.

II. from an employees point of view?
Jetro: All the problems that existed, have been fixed in last few months. One has been the visibility of possible projects to people who are on bench, like a possibility to browse possibly interesting cases on a board. There already something along that way implemented but it is quite underutilized and the people who submit the cases into the system rarely remember to fill out the description about the techs included etc.

Do you see any problems in Futurice's way of doing staffing?
Jetro: One problem is that not all of us staffers know everyone who is being staffed well enough. Usually staffers only know their own tribe well enough for staffing,

which creates silos quite easily and the same thing happens also between competences (developers, designers, business advisory). Of course in such a case when you don't know the person being staffed you should talk to them and try to get to know them better. And that (not knowing everyone and siloing) I regard a small problem at least it's not as efficient as it could be when everyone knew everyone well enough. Especially problematic is that certain people have to be physically around in the office to be able to staff certain people. Even though systems have a lot of (staffing and skill) related information, most still reside inside main staffers' heads. Quite few people have submitted their My Journey canvases into the intra- systems so those usually don't offer any tips about what the person wants to do next careerwise.
Pette: One of the biggest challenges is managing expectations inside the organization, meaning that we might promise too grand things when the person starts in Futurice. For example the opportunity to learn new is very subjective thing. For someone it means that they get new project every 2 months with new techs and for some it means that they get to change project once in 2 years with maybe some new techs. So unifiying the promise we give to our employees about the projects we can offer and how often is one of the aspects that needs work. And at least when I pitch a new project to a new candidate I try to be as realistic as possible about the upsides and downsides of the projects to manage expectations and to avoid any letdowns in the future. And managing these expectations takes time mostly because the amount of communication it involves between multiple different people. One aspect is that people want to select offerings even though they have not been won. Which causes people to wait for a case for months long that might not come through. Also the notion that one cannot say that I WANT to be in this project but one can say I don't want to be in this project when the project is still in offering phase. Only after we have closed multiple cases then you can choose between them.

Have you accumulated any tacit knowledge during the staffing process and if so what kind?
Jetro: Depends what kind of relationship the person and the manager (staffer) have, how much information is shared between them. The better you know the person the more accurate you can staff them. Also there might information like health issues that can't be shared in internal systems and communication, such as recovering from burnout etc.
Pette: Understanding about the old cases the person has done and how a certain person works in a team. This is something that is missing from all systems at the moment. We (staffers) are also genuinely interested about what people want to do inside Futurice hence we know the people we staff quite well and can usually tell what kind of project would interest this person straight off the bat.

How large of an impact does the employees preferences have?
Jetro: Most people can affect their decision given that their reasons are not completely arbitrary. But if one has declined a project on preference grounds one should be more willing to accept the next projects, after all this is still a workplace. Usually people have pretty good understanding themselves when they've said no too many

times and this is not a problem basically at all. **Interviewer**: What about when you have two people of which one really wants a certain project but does not necessarily meet all the requirements versus a person that has all the skills needed but has no preference but would be still willing to participate in the project? **Jetro**:In that case it usually the salesman's and customers call.
Pette: There are cases when we need to push projects even though we might know that the person might not like said project. Usually this occurs when a person has been too long without a project. Sometimes I might feel that I might know better what the person should do. Of course this might not be the case and these situations are always hard. But I could say that we have usually 1 or 2 such 'pushing'-cases ongoing at all times (meaning that a person does not seem to accept any projects). Sometimes it might be that I feel that the person has done a taxing or rough stint and needs a more 'easy' project even though the person just wants more challenges. For example if a person wants to change their core tech stack, like from mobile dev to back-end dev, it might be quite hard to sell the guy who could be senior as a junior dev.

Can you give an estimate how many have strong preference to do certain tech and how often it is a sine qua non?
Jetro: A lot of people have preferences. But it being a sine qua non depends completely on their personality. Some people are content with any project and some want a very specific one. Usually with seniority people also have a bit more specific view what they want to do and also they want some change to their core techs.

Do employees have preferences regarding clients?
Jetro: Not necessarily a certain client but an area of industry can be a preference. Modern industry like game industry is usually more interesting to people than stiff industries like financing. In some cases people want to improve a service that they themselves use a lot.

Do clients have preferences?
Jetro: Still quite many customers want to hold interviews and even might a have a scoring system or some kind of pre-tasks. Exception is usually with customers we've worked for long and we've established a common trust, usually in those cases they've come to know that we've provide competent people and let us staff the projects quite freely. In current market a lot of companies have senior developers free, so the clients can pick the best of the bunch quite easily which affects the junior developers ability to pick the most interesting cases.
Pette: Yes a lot. In most cases the client feels that they are part of the team. So even the culture fit and person dynamics can be part of the requirements that the customer has. So the interviews can be also more cultural interviews as well as technical interviews, just to see that the person is a match to the team. And in RFP cases the interviews tend to be more technical and usually aim to verify that the person can do the tasks that were required in the RFP.

If you would get complete freedom and budget needed over changing staffing, what would you do?
Jetro: I would improve the staffer groups communication and work processes. As the staffing team has been mostly same over the years, it might require some new people to be introduced to actually change the ways that people have accumulated. I would also just allocate more money to Power development to get more stuff done.
Pette: I would not change our systems or processes. First I would alter the Career Path, My Journey model and recruitment that we have. So that we would better take into account the consultant's need to create their own personal brand. Culturally this is a bit hard as building an personal brand is no way the most natural need that people usually have. Big part of developers don't feel like it is an important aspect of their career. But I would like to underline it more that it is part the job to have an personal CV and brand, that can be advertised to our clients. It help in multiple areas of work because then even if our people had well-made, up to date LinkedIn profiles we wouldn't need any internal CVs. We could just link the profiles as part of the offerings.

Do customers usually have leeway in the start/end times of projects?
Jetro: It depends quite a lot on the customer and how much history Futurice has with the said customer. Old customers are usually ok with waiting and it might be even possible to start early with them as there is not so much time needed to vet the consultants.
Pette: In a case where the client already has a fully working team, then the client tends to protect that teams functionality as much as it can. Thus the flexibility of new hires start and end dates is high. The other case is that the RFP usually have more strict frames in them and it usually means that the schedules have been approved somewhere higher up and are not that likely to flex.

B Final interview with Olli Hietamies

This is the interview conducted with Olli about the staffer and it's results. The content has been translated to English and shortened, cut or paraphrased to leave out information about named clients or employees of Futurice or if discussions concerned things not related to this thesis.

Regarding the skill based search: is it useful, how would you use it?
Olli: Well the basic use case for when staffing a project is that I already know the techs that I am searching, and this has been the case for as long as I can remember, at least until last summer. And for example at the moment I've been extensively searching for skill combinations of React front-end and Java back-end and as bonus Finnish language is needed. And what I specifically like about this tool at hand is that the current search that I'm using is quite rare combination in modern development so the weighting is really helping. In some cases I might want to weigh back-end know how more or in some cases front-end, setting the need for the required amount of experience is beneficial to me as we might be looking for person that has years of experience on the skill needed instead of some little amount of knowledge. What was a bit hard for me was the scaling of proficiency levels as I'm more used to old way of thinking where experience is only metered in person days or person years and not as levels that each can individually set based on their feeling.
To me the actual inner workings of the tool are not that important and only the actual results are interesting, meaning that I don't need to see the individual scores of each underlying data source but would be content with just seeing the correct order and ranking of the results. Even the total score could be left out. But in general what is usually most important for me is the language, as it is something that just needs to be there for some projects and clients and almost always when it's required, it's Finnish. In actual skills it would be ok if some of the skill were further or "less" than I queried but usually the language is a must when its defined.
Interviewer: So basically there could be a separate selector for the language?
Olli: Yeah, but the languages in their current from are as skill in power so it would need some more thinking how to implement it. And for example Tampere people rarely have Finnish as a skill at all (in Power). So I don't find any Tampere people when searching with Finnish selected even though I would want to. And because of that sometimes I need remember to do separate searches without the any language selected just to check who there is available. And I do realize that we have quite many people who tend to go unseen in power due to situations like the aforementioned but not limited to just languages but to skills as well. And I don't think even projects have the skills set so that we could infer the experience information from those. But as a whole I think that the skill-based search mode of the "staffer" should be incorporated as part of the current search functionality as soon as these problems have been tackled.

How did the results appear to you in general? Did they produce results that you were expecting and did the results seem "correct" to you?
Olli: The results seemed correct, I did not do a straight comparison between Power's old people search but it seemed to produce correct results. I liked the fact that it

showed me the experience straight in the listing unlike the default search in Power does and thus it was easy to evaluate that the results seemed correct. Of course it was hard to say if someone who would be a no-brainer match was left out of the results. The the purpose of the date selectors was left a little blank for me.
Interviewer: Their purpose is to act as time boundary for the FTE selector, so the time range when the person needs to have x FTE free or no vacation.
Olli: The start date I got, but end date was a question mark for me as I usually only check that the person is free on the start date of the project as people in Futurice tend to be either free until the end of time or then in project, long projects many months ahead are quite rare and also the projects I'm staffing tend to be long projects and not short stints of week or two. But the vacation check is a good point which easily tends to be forgotten when checking manually for candidates to projects. But the visual representation of the power's original search is the way how I usually check for availability. Sure if you're looking a candidate for a very small stint the end date can be handy, it's just not what I usually need to do due to the nature of the projects I'm staffing.

What would you change about the current skill query in staffer or what would you do next to make it better?
Olli: Maybe it could be the AND/OR possibility for the skills to accommodate the languages for example, but on the other hand it would change the fuzzy nature of the tool and it's ability to provide "surprising" candidates. But as mentioned before, languages is one that would need some extra attention. Also a way to export the results would be nice as usually I'm after a list of names to pass on. Usually the search I make is the tribes of Helsinki, maybe sometimes with Tampere included, then techs, Finnish language and and one month ahead or what the default is.

Do you usually read the staffing comments?
Olli: Lets say not often enough. Usually my case is to just provide the shortlist to the salesmen and it would be good to check the preferences already here but as the salesmen talk with the candidates still it's easy to forget. What I sometimes check is the days per week the person works as some have written it there.

Regarding the Person based search:
Was the functionality useful? How would you use it? Why?
Olli: Here I tried to approach this from the designer staffing perspective and for example in that case I see that the diversity in data sources would be beneficial. Especially when the determining factor would be something else than the skills data from Power. For example designers' skill options are quite narrow thus they tend to similiarize the candidates too much. And usually the projects that they have been part of and thus the different designing roles inside drive a large factor in designer experience and selling. Getting matches based on "I know this person has done this kind of work, I wonder who has also" would be beneficial in designer staffing but as our structured data on project information in Power and at Futurice as whole is so thin I don't we can still do that. One thing that came up when I was testing this,

was that it would be nice to use when customer asks for another developer similar to what they already have. But in that case would it really be just skills they are after or is more the personality and overall experience that the client is after. Rotations could also be one good case to use this feature on but in those cases its more about what techs the project needs not the necessarily all the skill that the person has. For example we have a person in project wanting to rotate out who has done a lot of PHP but lately done some React in projects, and if we find a similar person to them they probably are gonna be PHP heavy which is not the trait we're looking for here. In some cases this could be useful and in some cases not that much. The designer side is the one where I could use this as I don't personally know these people as well as the developers in our site. Therefore it would be easier for me to just use this search functionality as I wouldn't have a hunch who would be a good fit already.

Were the results accurate? What kind of inaccuracies did you come across?
Olli: In this case it's really hard to say, there were few cases where the results were plainly wrong both designers. On developer side it's hard to say if the functionality actually aids me, once again the language problem is there. With the few test searches I did using developers as query it produced the "usual suspects", so on that side I only got pretty accurate and "correct" results and the queries seemed to contain the people I expected to be similar. For me it's easier because I know the people I'm proposing as candidates quite well. But for example the guys doing the actual selling to clients might only know the one guy with certain tech skills and I think in those cases this feature would work quite nicely, it just would need some testing. Also this enables "lucky surprises" where some name, who I thought I knew, pops up with skills that I hadn't noticed providing me with new information about that person without interacting with them. And in that sense both of these features skill and person search are quite refreshing and useful as they bring up the candidates that would've been filtered by the old boolean based search system because they forgot to mark a new skill they acquired or didn't have Finnish set as a skill even though they speak it natively.
Olli: And more generally related to this whole staffer, I think the information about different industry experiences and preferences would increase the accuracy of the recommendations and even the overall data in Power quite a bit. Also project information in general could be more accurate. But I think it's good that we're already utilizing the staffing comments in the search as they can easily be forgotten and I think it would be nice to even highlight the mismatch when the person does not like (by telling in the comment) what is being searched and still gets top rankings.

Generally how do you see experience preference favor selection, is it useful in both or would you remove it from either of the search modes?
Olli: Thinking aloud here but if I think it from the perspective of developer/designer I would probably only allow preference based searches to make sure that nobody would never get offered a project that they would not like to do. On the other hand when I'm filling out the forms for public tenders only thing that makes a difference

is experience in some tech for example Java even though the candidate might've done and preferred React for the past few years. And if you get in because of your experience in Java you still might be able to code React apps while in the project. So in cases like the aforementioned, the experience you have in tech that you don't want to do any more might get you projects that actually have techs that you prefer to do. So in that sense the experience search has a use too. >I think especially in the people based search it would need to clearly state whose experience and preferences are being weighted here. In people based search I see it's quite rare that you want to find people that want to do the same things as somebody has as skills, usually would want to find people who are as experienced.

C Person query for staffer

```sql
SELECT person.*, tribe.site_id,
CASE WHEN allocs.total is NULL THEN 0 ELSE allocs.total END AS max_alloc,
CASE WHEN abs.sum is NULL THEN 0 ELSE abs.sum END AS absence_days
FROM spreadsheet_person as person
LEFT JOIN (SELECT person_id, COUNT(*) as comp_count
    FROM spreadsheet_personcompetence
    GROUP BY person_id) as comps_count
ON person.id = comps_count.person_id
LEFT JOIN (
 SELECT DISTINCT person_id, COALESCE((MAX(rp.total_allocation +(
  SELECT COALESCE(SUM(total_allocation),0)
  FROM spreadsheet_rollingperson as rp2
  WHERE rp2.person_id = rp.person_id
            AND rp.id != rp2.id
   AND rp2.start_date <= LEAST(rp.end_date, '{end_date:%Y-%m-%d}'::date)
   AND rp2.end_date >= GREATEST(rp.start_date, '{start_date:%Y-%m-%d}'::date)
   AND rp2.proposed = false
  GROUP BY person_id
 ))), rp.total_allocation) as total
 FROM spreadsheet_rollingperson as rp
 WHERE end_date >= '{start_date:%Y-%m-%d}'::date
  AND rp.person_id IS NOT NULL
  AND rp.proposed = false
 GROUP BY rp.person_id, rp.total_allocation
) as allocs
ON allocs.person_id = person.id
LEFT JOIN (
    SELECT person_id,
    SUM(LEAST(end_date, '{end_date:%Y-%m-%d}'::date) -
    GREATEST(start_date, '{start_date:%Y-%m-%d}'::date))
    FROM spreadsheet_absence
    WHERE end_date >= '{start_date:%Y-%m-%d}'::date
    AND start_date <= '{end_date:%Y-%m-%d}'::date
GROUP BY person_id) as abs
ON person.id = abs.person_id
LEFT JOIN spreadsheet_tribe as tribe
ON person.tribe_id = tribe.id
WHERE person.employment_end_date >= '{start_date:%Y-%m-%d}'::date
    AND person.employment_start_date <= '{end_date:%Y-%m-%d}'::date
    AND (abs.sum <= {absence_days_max} OR abs.sum IS NULL)
    AND ((person.days_per_week/5 * person.utz_target / 100)
        - allocs.total >= {fte_needed} OR (allocs.total IS NULL
        AND person.utz_target/100 <= {fte_needed}))
    AND person.primary_role_id IN ({role})
    AND comps_count.comp_count IS NOT NULL
    {sites}
    {tribes}
    {person}
ORDER BY person.id;
```

Listing 3: Database query for fetching candidates that fit the query constraints.