Discussion Paper No. 17-018

Inventor Mobility Index: A Method to Disambiguate Inventor Careers

Thorsten Doherr

ZEW
Zentrum für Europäische Wirtschaftsforschung GmbH
Centre for European Economic Research

Discussion Paper No. 17-018

Inventor Mobility Index: A Method to Disambiguate Inventor Careers

Thorsten Doherr

Inventor Mobility Index: A Method to Disambiguate Inventor Careers

Thorsten Doherr[a,b]

August 2016

a) Centre for European Economic Research (ZEW), Mannheim, Germany
b) University of Luxembourg

Abstract

Usually patent data does not contain any unique identifiers for the patenting assignees or the inventors, as the main tasks of patent authorities is the examination of applications and the administration of the patent documents as public contracts and not the support of the empirical analysis of their data. An inventor in a patent document is identified by his or her name. Depending on the patent authority the full address or parts of it may be included to further identify this inventor. The goal is to define an inventor mobility index that traces the career of an inventor as an individual with all the job switches and relocations approximated by the patents as potential milestones. The inventor name is the main criteria for this identifier. The inventor address information on the other hand is only of limited use for the definition of a mobility index. The name alone can work for exotic name variants, but for more common names the problem of namesakes gets in the way of identifying individuals. The solution discussed here consists in the construction of a relationship network between inventors with the same name. This network will be created by using all the other information available in the patent data. These could be simple connections like the same applicant or just the same home address, up to more complex connections that are created by the overlapping of colleagues and co-inventors, similar technology fields or shared citations. Traversal of these heuristically weighted networks by using methods of the graph theory leads to clusters representing a person. The applied methodology will give uncommon names a higher degree of freedom regarding the heuristic limitations than the more common names will get.

Keywords: Inventor disambiguation, patents, fuzzy search, heuristic method

JEL-Classification: C81, C63

1. Introduction

This paper uses specific terminology from the field of computer science which sometimes collides with similar terms from other scientific fields with slightly different meanings. The main offender here is the term “cluster”. Although it has its appearance in many other research areas and its overarching theme is always the same, exactly this fact may lead to misunderstandings. Even in this paper the term cluster will be used on two different occasions: the clustering of groups of similar content within the patent documents (i.e., similar addresses or applicants) and the clustering of patent documents that are connected by mutual traits, like similar inventor names, similar addresses or applicants, common technology classes or cited documents, which are partially the results of the former definition. Mutual traits describe the edges and the patents are the nodes of a virtual network. A cluster should contain all patents that share a specific inventor name and have a high probability of being from the same individual. The procedure to identify these clusters is called “traversal” of the network. Traversal is a recursive process of moving through the network along the edges and collecting all touched nodes/patents into a cluster. Traversal stops when all nodes that should belong into a cluster were touched. Because patents are highly interconnected documents, the edges need to be evaluated before the traversal collects too many documents into a cluster. To simplify the heuristic for the evaluation, the algorithm uses a hierarchy based on the local latency and trustworthiness of the different traits. It starts with the addresses of the inventors as valid edges. Every cluster the traversal returns contains patents that share a similar inventor name at a similar home address. Of course this will not suffice to observe inventor mobility as every cluster is just a small time bubble of an immobile phase in the life of an inventor. As these bubbles lump together the information of several patents, they are referred as “hypernodes”, a kind of super patent. Applicant addresses are the next step in this hierarchy. The edges are similar applicants between these hypernodes. The traversal procedure watches the size of the resulting cluster. If it exceeds a pre-defined threshold, it rewinds the traversal to the starting node and increases the quality requirements of the edges for another run. This process can be repeated several times and is called “cascaded traversal”, preventing clusters with an unbelievable high number of relocations for an inventor, which usually are a sign for common

names in combination with a large applicant. The methodology is also applied for the following hierarchies: co-inventors, citations and technology classifications. The variation for the quality of the edges stems from meta-analysis of the original data, counting frequencies of inventor names per applicant, documents per technology classification and so on. The higher the meta ranking, the lower the frequencies and therefore the lower the risk of connecting hypernodes of namesakes. The disambiguation procedure will be discussed in the last chapter as the complexity of the data needs to be reduced to confine the more fuzzy traits of a patent. The following two chapters explain how this can be done.

The term “similar” was used quite often in the previous paragraph. The concept of “similarity” is trivial if it actually means “identity”. This is the case for classifications or citations as these operate with codes or keys assigned by the patent offices. For these, similarity is a binary choice. Identifying similar addresses, applicants or names is a much more involved endeavor. To integrate these into a system of binary choices, it is inevitable to define keys that cluster groups of similar entities together, i.e. all different ways a university appears in the data or all variations of a name including misspellings. As it is a fundamental requirement for this disambiguation method to reduce the complexity in the relations between the patents, the second chapter takes its time to explain the method of identifying these similarity clusters. The algorithm returns a weighted Jaccard index for the comparison of two terms, which is the weighted sum of the intersection of the words these terms consist of, divided by the weighted sum of the union. The weights for the words also stem from a meta-analysis, counting the frequency of every word in the data. The weight is the inverse of this frequency, giving less common words a higher value. The algorithm separates this meta information by the fields used for matching to prevent the blending of weights with a different context: a common street name may also appear in the applicant name field as an uncommon company name. Superordinate weights can be put on the contexts, i.e. applicant name field gets a weight of 70% and the street address a weight of 30%, to reflect the importance of the different contexts for the search. The algorithm produces tuples of matched terms together with a percentage for the similarity that is above a high threshold. This result can be interpreted as the definition of a network consisting of terms as nodes linked with edges rated by the similarity, a situation that is quite analog to the inventor disambiguation problem. The method to identify clusters of similar

terms is the same as for identifying similar patents for a specific inventor name - cascaded traversal.

The third chapter describes this central method with a main emphasis on the clustering of terms, like applicants, addresses and inventor names. Analog to the disambiguation traversal, the sizes for the cascades have to be defined a priori. “How many variants are common for a street address?” or “How many misspellings are thinkable for an inventor name?” are the questions whose answers, based on experience with the data and educated guesses, lead to these sizes. Cascaded traversal always injects exogenous assessment into the clustering. Sometimes it is impossible to give a decisive answer to the aforementioned questions because of highly heterogeneous structured terms. An applicant can be a university, a company, a person, a governmental organization and so on. Asian street names have much more variation than most western streets as they often include building names, floors and departments. It is quite cumbersome to have “one size fits all” for these different cases. The solution, called “nested cascades”, will also be introduced in this chapter. Cascaded traversal not only reduces the complexity in the data by eliminating the vagueness of similarity but is also a way to confine plausible inventor careers in the complex network of patents.

2. The SearchEngine

The patent offices do not administer special databases for assignees or inventors nor are they obliged to verify the names or addresses. Because of that there may exist multiple variants for a specific inventor or assignee, which can be explained by misspellings, different usage of abbreviations, name or address changes over time. If there is more than the data of one authority involved, this problem increases significantly because of different standards. The solution to this problem discussed here is to create an identifier for every group of variants that belong together with a high probability. This is a virtual cleaning process as the data itself will not be changed nor will there be a "preferred" variant that overwrites the other variants. The tool used for this task is simply called "SearchEngine" for further reference and is under continuous development by Thorsten Doherr, ZEW. It combines many ideas that have their origin in the field of computer science like word based heuristics, phonetic algorithms, fuzzy logic and network analysis.

2.1 The Preparer Gateway

One typical problem the algorithm is designed for is to match two tables from different sources by a combination of fields that share the same - usually fuzzy - characteristics, like name, address, city, zip and so on. A direct SQL join by these fields is of limited use because of abbreviations, misspellings and typing errors, different positioning of words or additional/missing words. The extensive harmonization of both tables by transforming the data to uppercase, replacing special letters to their common (phonetic) representation (i.e.: the German "Ü" to "UE"), suppressing of special characters and the unification of abbreviations will improve the situation for a direct join. These cleaning steps are also a part of the SearchEngine implemented as the so called preparer gateway. The preparer gateway is responsible for the harmonization of all the data entering the deeper layers of the algorithm. Besides the more or less cosmetic modifications of the data, it can also implement more extreme phonetic methods, which destroy the readability of the data but improve the robustness against misspellings and typing errors. Every field can be connected to a different list of preparers, some of them specialized to reflect the context of the associated characteristic. It is even possible to associate more than one preparer list to a single field, creating new entities. These combinations of preparers and fields are further called search types. The outcome of the preparer gateway is a set of words without any specific order separated into subsets by the search types they origin from. From this point on the term "word" describes all the token the preparer gateway returns after applying the harmonizing and/or the more aggressive phonetic preparer like Soundex (RobertRussell, Margaret Odell, 1918), Metaphone (Lawrence Philips, 1990), Kölner Phonetic (Cologne Phonetic) (Hans Joachim Postel, 1969) and n-gramm.

2.2 The Heuristic

2.2.1 Identification Potential

The heuristic is based on the assumption that the occurrence of a word is inverse proportional to the identification potential (IP) of this word. Using the internet as an analogy, a quite common word entered into a web search will result in a large list of results making it difficult to find the intended entry. The resulting list of potential hits for a seldom word is smaller and the

identification potential is higher. Because a search usually involves more than one word, the algorithm uses a relative identification potential (rIP). The following section describes the development of this measurement starting with a basic first version:

$$rIP(i) = \frac{occ(i)^{-1}}{\sum_{j \in S} occ(j)^{-1}} \tag{1}$$

with S being a set of words defined by the search term, $i \in S$ and $occ(i)$ returning the occurrence of the word *i*.

To get the occurrences of the words the SearchEngine needs to be fed with the characteristics of one table, the so called base table. After passing the preparer gateway, the words of the search fields will be registered in a special table, the registry. An entry in the registry consists of a word and a counter for the occurrence of this word. The registry is further organized into chapters, one for every search type, to preserve the context of the words. Every single entry is also linked back to the containing records in the base table by supporting tables. The heuristic is extended by the possibility to put different weights on these search type chapters. These chapter weights are called priorities because they also influence the optimization of the implementation by giving the algorithm an order to work with. Another extension to the heuristic is the introduction of offsets that are added to the word occurrences. These offsets smooth out the relative differences between the words and can also be applied per chapter. The occurrence function now requires two parameters: the word and the search type the word belongs to. With $st(i)$ returning the search type of word *i*, $pri(j)$ and $off(j)$ returning the priority and the offset of search type *j* and *n* being the number of search types, the extended rIP_s can be defined as:

$$IP(i) = \max(occ\big(i, st(i)\big) + off\big(st(i)\big), 1)^{-1} \tag{2}$$

$$share(i) = \frac{pri(st(i))}{\sum_{k=1}^{n} pri(k)} \tag{3}$$

$$rIP_s(i) = share(i)\left(\frac{IP(i)}{\sum_{j\in S}\begin{cases} IP(j) & | \; st(i) = st(j) \\ 0 & | \; st(i) \neq st(j) \end{cases}}\right) \quad (4)$$

The function $occ(i, st(i))$ returns the average occurrence within the search type $st(i)$, if word *i* is not found in the registry for search type $st(i)$. The function max returns the numerical highest of the parameters. The function *share(i)* transforms the absolute priorities into relative shares that sum up to 1.

Good values for the priorities and the offsets highly depend on the used preparer and the characteristics of the search types. In most cases a match has a clearly dominating characteristic like a company name that should get a higher priority as the supporting characteristics like address, city or zip. In conjunction with a customized cutoff limit, it is possible to focus the match and reduce the number of false positives. The usage of offsets is much more experimental. They can be used to like a slider between an occurrence based heuristic and a simple word based metric where every word has the same value. This is the case if the offset is negative and higher than the highest occurrence of a search type.

For any search term the words of the records found in the base table are compared with the words of the search term. For every shared word the associated rIP_s is summarized to get a measurement for the identity ranging between 0 and 1. An identity of 1 means all words of the search term exists also in the found record. Missing words from the search term result in a lower identity according to their rIP_s. Only found records with an identity above a given limit are considered candidates.

Until now all candidates with the same matching words are equal. In some cases it is desirable to rank these results according to the words of the candidates that are not part of the search term, thus preferring candidates with less additional clutter. The surplus words of the candidates generate a discount on the identity, called *feedback*. The extent of the discount can be adjusted with the feedback parameter f which has a valid range from 0 to 1. With F being the set of all the words of the found candidate record, the final definition of the rIP_f is available:

$$Jaccard(i) = \frac{\sum_{j \in S} \begin{cases} IP(j) & | \; st(i) = st(j) \\ 0 & | \; st(i) \neq st(j) \end{cases}}{\sum_{j \in S \cup F} \begin{cases} IP(j) & | \; st(i) = st(j) \\ 0 & | \; st(i) \neq st(j) \end{cases}} \tag{5}$$

$$rIP_f(i) = rIP_s(i)\big((1-f) + Jaccard(i)f\big) \tag{6}$$

The function (5) is called Jaccard because it is the implementation of the Jaccard similarity coefficent (Paul Jaccard, 1901). It measures the similarity of two sets of properties by dividing the number of shared properties by the size of the union of both sets. With a feedback of 1, equal priorities and large enough negative offsets (equalizing all occurrences) for all search types, the identity transforms into a Jaccard index measuring the similarity between to sets of words.

2.2.2 Score

The identity is the main result of the SearchEngine algorithm, but it is a relative measurement of similarity. Sometimes it is desirable to have additional information regarding the absolute quality of a match, especially if its identity is close to the cut off limit. Some search terms just have no word with a high identification potential, while other consist mainly of quite seldom words. The SearchEngine returns for every search term an absolute value called *score*, which is an arbitrary indicator for the identification potential with a straight forward design:

$$score = \sum_{i \in S} \frac{share(i)}{occ\big(i, st(i)\big)} \tag{7}$$

The score increases with the share of the words and decreases with their occurrence. A search term with a high score has a higher probability to yield proper results even when the identity is quite low. It is used to rank the results as the score only has a meaning if it is compared with other scores. This property is also quite useful to define further heuristic rankings of search terms, i.e. inventor names.

2.3 Implementation

The rIP_s is used as the heuristic for the search process that collects the candidate records for a search term. Given that the resources for the algorithm are restricted by computing power it is more profitable to first look for words with a higher rIP_s, until the resources for one search step are exhausted. The maximum size of the candidate list is the main regulator for a healthy balance between performance and completeness. Is the so called search depth too high, the performance will significantly be decreased for little benefit consisting mostly of false positives. A too restrictive search depth can lead to a loss of valuable hits, because the word with the highest rIP_s may have a higher absolute occurence. The identity limit will also be considered to further reduce the candidate list for the following steps. A candidate within the list already has a preliminary identity consisting of the words used for filling the list, which are usually just a fraction of the whole search term. If the identity of the unused search words won't push a candidate above the limit, it will be dropped from the list. In the next step the used words of every candidate will be synchronized with the remaining words of the search term to complete the calculation of the identity on the base of the rIP_f. It is this step that requires the majority of the computing power which explains the restrictive selection of the candidates beforehand.

2.4 Handling Misspellings

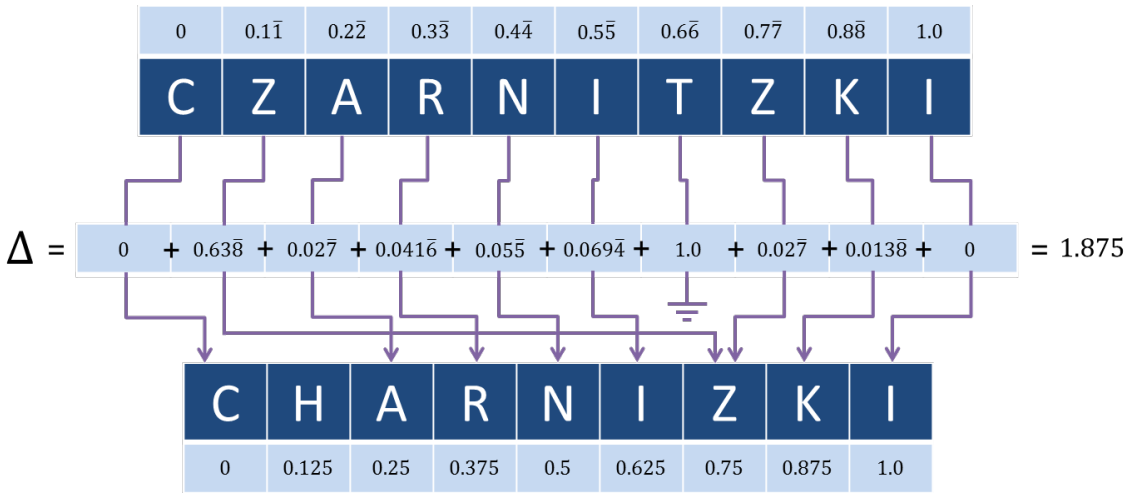
The SearchEngine is a word based algorithm. If a word can’t be found in the registry it will get a rIP based on the average occurrence for its search type. But the main problem is that there are no connected base table records making this word a dead weight. Another problem are misspelled common words that occur in the registry with a very low occurrence compared to the proper words. These words can misguide the search process in favor of the other misspelled entries. Phonetic preparer like Soundex and Metaphone reduce this problem by creating codes for similar sounding words. The n-gram method uses shifted tokenization (i.e.: 3_gram("DOHERR") = ["DOH", "OHE", "HER", "ERR"]), creating multiple tokens for one word and thus reducing the impact of misspellings as they concern only a part of the tokens.

Method	Soundex	Metaphone	Cologne
Code	T652	BRTN	3467
Example 1	TARNOWSKI	BARATON	WAGNER
Example 2	THORENZ	BERTINI	WUCHENAUER
Example 3	TRUNK	BORDIN	WEGENER

Table 1: Words represented by the same phonetic code

There is a price to pay for the gained robustness. The algorithm will return much more false positives because the phonetic representations do not only include the misspellings but also similar "legal" words. In the case of n-grams all entries containing the same tokens have the same identity as the heuristic ignores positioning. The problem is, that phonetic methods are specially designed to retrieve false positives in the hope that the intended result will be within them. Usually these methods are used in an environment where an operator enters single requests into a terminal and examines the retrieved results. For the SearchEngine an additional layer has to be applied that fulfills this task. This layer simulates the operator by applying a string comparison function for every search type that implements phonetic preparer. This function returns a value between 0 and 1 for the similarity of strings. Because of this numeric property, it can easily be integrated as an equivalent to the identity of a search type. It compares every word of the search term with every word of the found term to identify the pairings with the highest similarities. The final result is the sum of these values divided by the highest possible score based on the term with the most words. An additional score will be calculated that compares the terms as whole strings. The maximum of both scores defines the identity. Both types of comparisons are necessary to guarantee a high flexibility of the measurement against different positioning of words and unclean separated words (i.e. by missing blanks). Because this flexibility requires a large number of comparisons the underlying algorithm has to be very efficient. The method used is called Least Relative Character Position Deltas (LRCPD). Every character in a string has a relative position between 0 for the first and 1 for the last character. The algorithm searches for every character in the first string the matching character in the second string with the smallest difference between the relative positions. If a

character can't be found a maximum delta of 1 is used. The sum of the deltas divided by the length of the first string returns a disparity measure between 0 and 1.



$$lrcpd(word1, word2) = 1 - \frac{\Delta(word1, word2)}{len(word1)} = 1 - \frac{1.875}{10} = 0.8125$$

Fig. 1: Two different spellings of a complicate name compared by the LRCPD method

The LRCPD heavily depends on the direction of the comparison. For a symmetric behavior the comparison has to be done in both directions using the lower result. Another problem is the reduction of the deltas with increased string lengths. The limes of the average delta approaches zero for the comparison of infinite strings. For this reason the LRCPD implements a search scope around the relative position of the searched character. Starting from this position the search will be carried out in both directions until the character is found or the absolute distance to the start position exceeds the scope. The delta of a found character will be adjusted as if the string length equals this limit, always resulting in deltas between 0 and 1. The SearchEngine uses an arbitrary default scope of 12 characters in both directions (not including the start position). A higher limit is only recommended for results that will be manually checked.

Search types that implement phonetic preparer somehow distort the idea behind the original heuristic. The codes or fragments returned by the phonetic methods have a different distribution of occurrences than the original words. Through fragmentation or aggregation the number of words stored in the registry is reduced, the average occurrence is increased which

leads to more candidate records. This effect is subdued by the LRCPD layer but the main advantage of the original heuristic, finding candidates by the most identifying words, is watered down. Because of that, the SearchEngine supports incremental search steps. Multiple runs with different settings can be merged into one result set. Pairings of previous runs will not be overwritten by following search steps. It is advised to use phonetic preparer for later runs to fetch the candidates that actual have misspellings and to keep the main bulk of the results according to the heuristic.

2.5 Reducing Complexity is a Complex Business

Now that there are all tools and methods in place, the actual task of creating identifiers for variant groups of applicants and inventors can be put into the focus. The only difference to a common match of two different data sources is that one data source is matched with itself. There exist many different approaches to disambiguate or match this kind of data. Trajtenberg et al. (2006) used the Soundex method and introduced a frequency based heuristic. Raffo and Lhuillery (2009) analyzed different cleaning methods for a simple string based algorithm and compared them to n-gram methods in respect to recall rate and false positives. Schoen, Heinisch und Buensdorf (2013) combined simple string matching, n-gram and a Jaccard similarity coefficient for their “name game”. All these approaches have in common that the matching results are transitive, be it by the method used or enforced by subsequent cleaning procedures. The latter is seen problematic but imposed nevertheless as “the only plausible course of action” (Trajtenberg et.al., 2006). The results of the SearchEngine can also be forced into transitivity by applying a feedback of 1 transforming the identity into a weighted Jaccard index. The advantage of transitive matching in respect to disambiguation is the consistent mapping of entities into groups. Intransitive matching means that the identity of the reversed match can differ from the original identity. The reverse match can even return a value below the identity threshold. If transitive pairs define a network consisting of easy to identity clusters of fully connected subgraphs, intransitive links between nodes create complex directed subgraphs. The connection strength between two nodes can be defined as a tuple consisting of the maximum and the minimum of both identities eliminating the direction of the edges, but the graphs are still not complete.

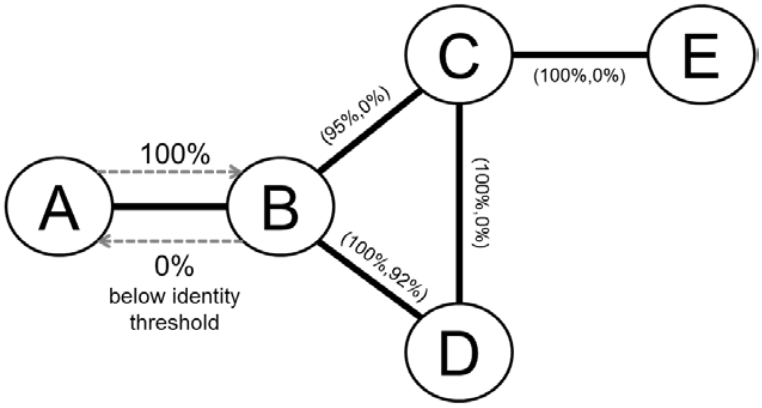


Fig. 2: Undirected graph

It is obvious that a network analysis is required to identity the clusters in such a network. This requires a lot more effort than simply collecting the clusters in the network of complete subgraphs defined by the transitive match. But this effort is justified by the additional freedom of the intransitive match. Intransitive matching allows pairs consisting of overspecified and relatively underspecified search terms to exist as connected nodes. An overspecified search term has additional clutter that distracts from the actual target, i.e. mentioning subdivisions that obfuscate the firm name. As long as the actual target exists in the data in a proper specified form it will collect all the overspecified entries even if these are not able to find the actual target on their turn.

A high identity threshold provides that the single connections in the network are believable. But the size and the structure of a graph can lead to initially unexpected composition of a cluster. Two meta structures can be identified as the main perpetrators in this regard: black holes and thickets. A black hole is a node that has a suspicious number of connections. These are caused by underspecified data artefacts, i.e. a company name consisting only of a legal state or a city name. Luckily these can easily be detected and mitigated before the traversal of the network by cutting all weak connections of a node whose number of connections exceeds an artifact threshold.

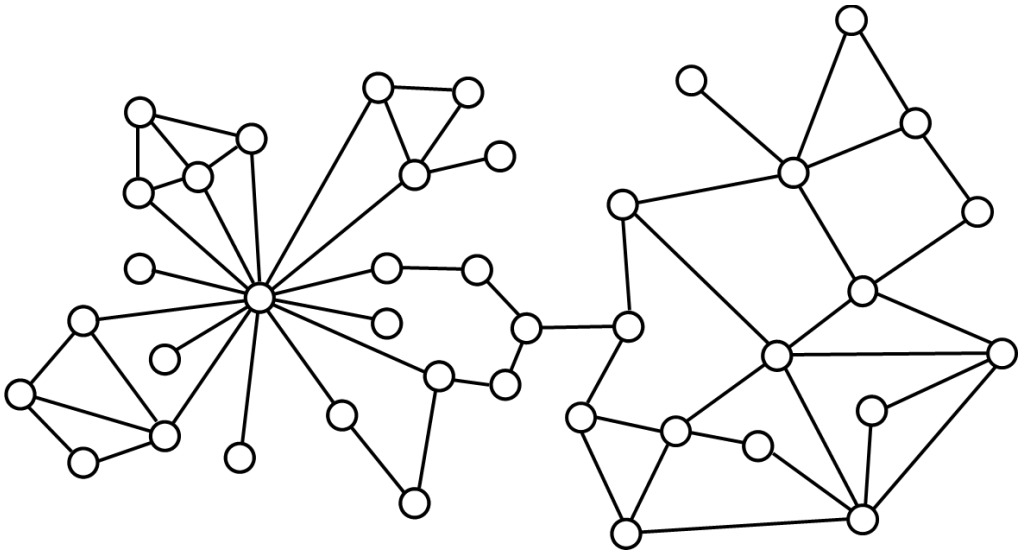


Fig. 3: A black hole and a thicket

A thicket can't be identified pre traversal. Its structure is chaotic and only discernable from a healthy cluster during traversal. It is the direct result of under- and overspecified terms that build upon each other. An overspecified term can link to several underspecified terms which open the way to subgraphs containing overspecifed terms and so on. In this context "overspecifed" does not automatically mean "clutter", but also proper specified terms of common words. To solve this problem suspicious large clusters can be traversed again, but now with a limit on the connection strength. A more efficient method to this approach is the cascaded traversal.

3. Cascaded Traversal

3.1 Definition of a Cascade

Originally introduced to cut down thickets, cascaded traversal has some additional benefits. The basic idea was to identify thickets during traversal and not after the complete network analysis to save computing time. Every time a cluster reaches a defined node limit the traversal has to start again with a more limiting threshold for the connection strengths. The cluster size limit is a discrete value that can be determined by answering questions like: "How many misspellings are imaginable for a name?" or "How many different variants of a company name seem to be

plausible?". The answers may be arbitrary but as there are usually multiple cascades with increasingly restrictive conditions in place, the whole process can be adjusted for adaptability. To define a cascade following conditions and steps have to be considered:

- Define a set of rules with **increasingly** restrictive conditions for the validity of a connection. Any rule has to include the restrictions of the previous rule.
- Attach an activation size to every rule, i.e.: unrestricted, min > 90 @ 4, min > 92 @ 6, min > 95 @ 6, min > 97 @ 11
- The rules will be exclusively activated in order of definition. The active rule will be replaced if the cluster size of the following rule is reached during traversal.
- Every time a new rule is activated, the traversal of the network starts again for a given start node with the new rule in place.
- A valid start node is any node that does not already belong to a cluster created by the cascaded traversal of a previous start node.

Any rule creates a new virtual network that is a thinned out version of the network defined by the previous rule. As the propagation of this thinning out process is independent from the start node, there is no overlapping of the resulting clusters. As the cluster size limit can grow from rule to rule to reflect the increase of the connection quality of the remaining network, the cascade adjusts itself by being easy on smaller groups and even letting larger groups survive, as long as the connections are strong (see figure 4).

As manual checking is out of the question for large numbers of observations the identity threshold for the disambiguation of the applicants and inventor names is quite high to guarantee that the connections in the resulting graph already have a good quality. The maximum value for a connection is always equal or higher than the identity threshold. The minimum value for a connection can be zero if the search in the reversed direction returned an identity below the threshold. For this reason only the minimum is used for the rules. The rule set for the inventor names should be more restricting than the rules defined for the applicants, because the inventor name index is the base for the next step to identify individual inventor careers. The applicant variant index is only used as an instrument to identity these careers in conjunction with the inventor name index. The probability that two inventors with the same name index have patents for two different applicants that are by mistake in the same variant

group is quite miniscule. Given the quality of the patent data an additional variant index can be created for the home addresses of the inventors.

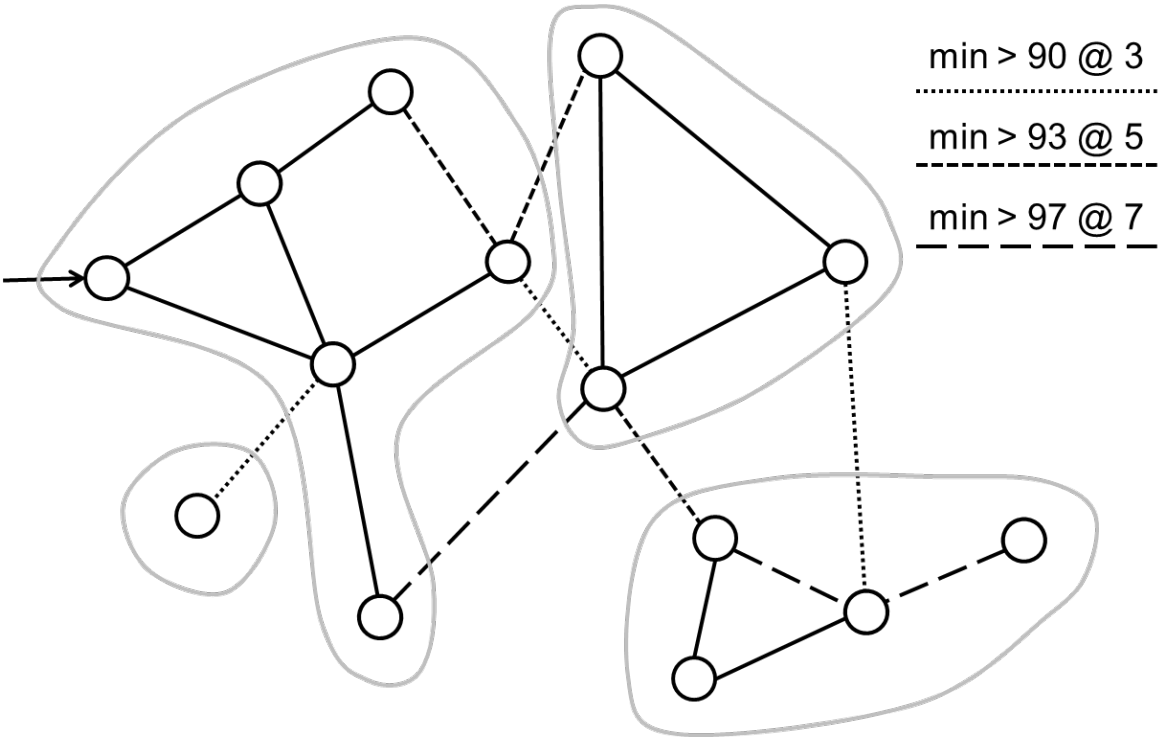


Fig. 4: A cascade with incremental cluster limits is thinning out a thicket

3.2 Nested Cascades

Compared to inventor names, the structure of applicants or addresses naturally shows much more variation. The variance of the term lengths is also much higher. Long applicant names or complicated addresses are more prone to create a high amount of quite similar variants than their shorter - less complicated – counterparts, because there are more opportunities for abbreviations, different positioning of words, misspellings and so on. It is difficult to define cluster sizes for cascades, if there is a high degree of heterogeneity in the data in regard of valid variants. This problem can be solved by applying a two-step approach consisting of a first cascade harmonizing the data by defining preliminary clusters of very similar variants and the main cascade to encompass major variants. The preliminary cascade always has at least one rule

that is active from the start. This rule enforces the identification threshold of the search or a higher limit on the connections, i.e. min >= 90 @ 0. Further rules can be included to control for extreme cluster sizes. The resulting network consists off connected hypernodes containing variants of comparable similarity. The cluster sizes of the second cascade are based on these hypernodes. The cascade syntax is extended by a semicolon separating the steps, i.e.: min >= 90 @ 0, min >= 95 @ 100; min >= 90 @ 5, min >= 95 @ 9, min >= 97.5 @ 21. Theoretically more than two cascades can be nested using this method but the practical use is questionable.

3.3 Rough and Fine Inventor Cascades

Inventor names typically don't have many different variants, but still, some names are more prone to different spellings or underlie specific cultural habits. In the US, the middle initial is more important than in Europe, where it often is left out. The same inventor can appear with her middle initial, the full middle name or completely without it. Some inventor have names containing special characters that somehow got lost in the administrative process and are replaced with more common characters without specific replacement rules. These circumstances produce different spelling schemes that extend the expected variations of simple misspellings. A fine cascade that yields perfect variant groups for the majority of inventor names may put the variants for the more problematic cases into different groups. A rough cascade with more generous activation limits and softer restrictions on the connection strength is able to look over these shortcomings and properly put all variants of the difficult cases into one group but also many names that should not belong together. To get the best of both cascades the rough cascades also takes the address of the inventor into account. The rough classification groups all subordinate fine variant keys at the same address together. The rough variant index defines the namespace but the fine variant index defines the links for the following step, the Inventor Mobility Index.

4. The Inventor Mobility Index

The rough inventor name variant index defines a namespace into which all patents of inventors with this name belong. There are two extreme positions imaginable: all patents are from one person only or all patents are from different namesakes. The truth most probably lies in between. By investigating the data available for the patents of this namespace it becomes clear that these patents have different types of connections between them. This could be two patents being invented at the same employer (applicant) or inventor home address, sharing some co-inventors, citing each other or are about a similar technology. These pairwise connections span a network with heterogeneous definitions of connection strength. One way to solve this problem is a network analysis of the whole graph using patents as nodes. Because of the high interconnectivity and the disparity of the connection quality the high probability of thickets could be countered with cascaded traversal. But as the criteria for the cluster size limit is the number of patents it becomes obvious that the definition of a rule set will be uncomfortably arbitrary. The number of patents per inventor is very heterogeneous and the connection restrictions require some kind of ranking or weighting of the different connection types. The better solution is to define a hierarchical order of the connection types. First, only the types are used for traversal which are defined for all patents and have a good disambiguation potential. Common sense suggests the usage of the inventor address and the applicant as opposed to the patent classification. Resulting clusters are the basic milestones of the inventor career. These clusters are also called hypernodes as they become new nodes of a nested-graph model (Alexandra Poulovassilis and Mark Levene, 1990, 1994) as the following layers of connection types are applied in order of reliability. This approach has two major advantages: it is possible to aggregate the data of the patents within a hypernode to create additional information a single patent could not provide and it is more comfortable to define a cascade using career milestones. Also, the cascaded traversal partially solves the common name problem. One symptom of a common name can be a huge name space occupied by dense thickets. Cascaded traversal keeps the number of milestones within a tolerable limit for an inventor career without the explicit knowledge about the commonness of a name.

4.1 Meta Data

Not all common names may produce large thickets and the cascades that effectively cut down common name thickets may be too restrictive for some very active inventors. For example, under the name of the iconic “JOHN SMITH” are only 52 patents filed at the US Patent Office (2014), resulting in an unobtrusive network. The cascaded traversal can’t solve these problems based only on the information of the patents within a namespace, but it is also required to include meta information that ranks this information in the context of the whole patent data. The most important meta data in this regard is the score the SearchEngine returns for every inventor name (see 1.2.2). It can be used to rank the names by their identification potential, which is another term for commonness. Names with the same score get the same rank. Ranks are distributed without gaps and normalized to values between 0 (high commonness) and 1 (unique). These ranks can be used like percentiles to further refine the cascades by injecting global context, i.e. accepting any connection as long the percentile rank for the identification potential of a name is greater equal 75%. Other meta information is constructed by counting the number of disambiguated names appearing for an applicant or inventor address, by counting the forward and backward citations of a patent or how often a patent classification is used. Additional flexibility is gained by not only reporting the count but also constructing a percentile rank that can be interacted (multiplied) with the percentile rank of the inventor name.

4.2 Traversal

4.2.1 Home Address

Having the home address on the inventors is the ideal case. After defining a variant key for the different existing addresses using the SearchEngine it is possible to define the first layer of hypernodes within a name space. A hypernode contains all the patents a person invented at a specific home address. As the nested graph is complete no traversal is required to collect the patents of a hypernode. Some patent offices do not collect the street address. In these cases it is required to constrain the assignment by interacting the meta information of the address and the inventor. The rough inventor name variant index has higher restrictions in regard of these

meta information than the fine classification (see 1.7). Although the namespace is defined by the rough inventor name index, the connections between the nodes always require the same fine index. The address nodes containing more the one fine variant are the gateways between different spelling schemes from now on. The home address should always be the first connection type as it lays the ground work for the following network cascades. It is much easier to interpret activation limits of cascades as relocations between home addresses than, for example, the number of transitions between citation clusters.

4.2.2 Applicant

Usually the applicant is the employer of the inventor. In cases where the inventor address is missing or did not lead to an assignment due to heuristic restrictions, the applicant is the substitute for the home address to define a career milestone. The fact, that not only a patent can have multiple applicants, but also a hypernode defined by the address may contain patents from different applicants, results in a potentially incomplete graph. The traversal follows the links defined by the applicant variant keys that are shared between the hypernodes. Cascades rely on the meta information of the applicants in conjunction with meta data about the inventor name. The activation limit is based on an acceptable number of relocations of an inventor working for one applicant.

4.2.3 Fellows

This connection type implements the advantages of the hierarchical approach. The distinct co-inventors of the patents within a hypernode are the research fellows of the inventor. Hypernodes are connected by similar fellow names. The strength of a connection is defined by the absolute number, the maximum and average percentile rank of the joint fellows including the inventor owning the namespace. Another element of the cascade is a modified Jaccard index based on the squared number of joint fellows compared to the union of all fellows of both hypernodes. This arbitrary index takes into account that the probability of drawing the same combination of names by chance is closer to a quadratic than a linear function of the unmodified Jaccard index, although the real shape is unknown. This connection layer often

identifies reorganized applicant entities that could not be joined by a variant id because of name and location changes.

4.2.4 Direct Citations

A direct citation creates a strong link between two patents. If the same inventor name appears in both documents the probability of them being the same person is very high. In spite of this excellent connecting property, this connection type is only at fourth position, as a patent not necessarily has citations that connects back to the same data source. The cascades use meta information about the number of citations made of a patent because some patent authorities, i.e. the USPTO, include all citations suggested by the applicant, which can lead to counts in the thousands. The counting data helps to enforce a higher threshold on the inventor name percentile rank for these extreme cases.

4.2.5 Indirect Citations

A link defined by an indirect citation is a third document cited by patents of different hypernodes within a namespace. The cited document itself does of course not belong to the namespace, otherwise it would have been a direct citation. This connection type is much weaker than a direct citation as the document may only state prior art and can be referenced by many other patents. As usual for weak connection types, a multitude of parameters need to be controlled for to minimize the risk of wrong assignments. As with the direct citations, a reference counter is required to filter extreme cases. A modified Jaccard index ensures that the joint citations are in a healthy relation to the overall citations of both hypernodes. A high inventor name percentile rank ignores all other restrictions.

4.2.6 Expertise

The international patent classification (IPC) codes describe the technologies that define the inventiveness of a patent and thus indirectly the expertise of its inventors. The codes follow a hierarchical system which allows for truncation at specific positions or separators to get a broader view on the involved technology respectively expertise. There are two cascades for the expertise based on the main group, defined by the characters before the oblique stroke

separator, followed by a run on the main classification defined by the first 4 characters. The expertise within a hypernode consists of a distinct list of these codes. Every code gets two different weights. The meta weight is defined by a meta table delivering a percentile rank for the inverse occurrence of a code. An uncommon code has a higher weight than a common code. The internal weight is based on the occurrence of the code within the hypernode. A code that appears more often has a higher weight. The cascade consults a modified Jaccard index based on the meta weights and another value based on the mean shares of the internal weights of the matching codes. The expertise is a very weak connection type and should always be the last in the traversal order because larger hypernodes allow for a better assessment of the expertise of an inventor.

4.3 Final Discussion

A more extensive discussion of the exact definitions of the cascades is omitted, because they heavily rely on the quality of the original patent data and the way the patent system is organized. The EPO collects full address information, including street names, about the inventors and applicants. With such precise information about the location the usage of meta information for the inventor address is unnecessary. Its examiners define the required citations on their own without intervention of the applicants, leading to a much smaller average citation count per patent compared to the USPTO. This has a direct impact on the cascade specifications for the direct and indirect citations. If there is uncertainty in earlier cascades like the ones for addresses or applicants, the whole principle of the cascades is weakened. The activation limit of a cascade should be based on the real life context of the data, like relocations in regard of inventor addresses. If there is structural ambiguity in the address information, like missing street addresses, the cascades not only are about relocations but also have to take this uncertainty into account. A high level of uncertainty, especially in the early cascades, shifts the focus of the algorithm from the specification of the cascades to a more general approach using meta data as the central element for identifying the individual degree of ambiguousness. The cascades degenerate to a simple vehicle to activate the cascade conditions relying on this meta information.

References

Manuel Trajtenberg, Gil Shiff, Ran Melamed (2006): The “Names Game”: Harnessing inventors’ patent data for economic research. Working Paper, National Bureau of Economic Research.

Anja Schön, Dominik Heinisch, Guido Buenstorf (2013): Playing the ‘Name Game’ to identify university patents in germany. Working Paper, Social Science Research Network

Alexandra Poulovassilis, Mark Levene (1994): A nested-graph model for the representation and manipulation of complex objects. ACM Transactions on Information Systems 12(1), 35-68

Julio Raffo, Stephan Lhuillery (2009): How to play the “Names Game”: Patent retrieval comparing different heuristics. Research poliy 38(10), 1617-1627