The data, they are a-changin'

Paolo Missier Jacek Cala Eldarina Wijaya

School of Computing Science
Newcastle University
{firstname.lastname}@ncl.ac.uk

Abstract

The cost of deriving actionable knowledge from large datasets has been decreasing thanks to a convergence of positive factors: low cost data generation, inexpensively scalable storage and processing infrastructure (cloud), software frameworks and tools for massively distributed data processing, and parallelisable data analytics algorithms. One observation that is often overlooked, however, is that each of these elements is not immutable, rather they all evolve over time. As those datasets change over time, the value of their derivative knowledge may decay, unless it is preserved by reacting to those changes. Our broad research goal is to develop models, methods, and tools for selectively reacting to changes by balancing costs and benefits, i.e. through complete or partial re-computation of some of the underlying processes. In this paper we present an initial model for reasoning about change and re-computations, and show how analysis of detailed provenance of derived knowledge informs re-computation decisions. We illustrate the main ideas through a real-world case study in genomics, namely on the interpretation of human variants in support of genetic diagnosis.

Keywords data change, data refresh, big data analytics, provenance

1. Introduction

Many of the large datasets used to derive knowledge evolve over time. This causes problems as changes in the datasets invalidate some of the insight derived from them. The problem is relevant in data-intensive science, where experimental results often come from computational pipelines or simulations that rely on observational data. In these settings, not only the underlying data, but also the algorithms and external reference data sources used in the analysis evolve. These changes may represent both a threat, i.e. when a stale model is used to make decisions, and an opportunity, namely to upgrade derived knowledge by performing the analysis again. When the processes are computationally expensive and the available budget for re-doing old work is limited, it is important to be able to determine when re-computation, partial or complete, of the underlying analytic tasks in reaction to changes is beneficial.

The potential for exploiting provenance records for partial re-computation has been studied before, in the specific context of database operations. In the Panda system (Ikeda et al. 2011; Ikeda and Widom 2010), for instance, one can determine precisely the fragment of a data-intensive program that needs to be re-executed in order to *refresh* stale results. However, this requires the assumption that very granular data provenance can be collected for database operations, and that the semantics of these operations is well understood.

In contrast, in this paper we take a broader view and consider a more general scenario where (i) the computation involves any program P that has dependencies on external data resources, (ii) the program structure and details of its execution may be only partially observable (coarse vs fine-grained provenance), and (iii) the program may have been executed many times over many inputs, producing a (large) history H of past computations and results.

Changes in the content of the external resources may invalidate some, but not all, of the results in H. Also, as noted in the Panda system, when attempting to refresh the results that are affected, it may be possible to re-compute P only partially. In this paper we show how provenance records from past computations, of varying granularity, can be used to select the precise subset of H that becomes invalid when the content of external resources changes (*re-comp scope*). We also show how the starting point for a partial re-computation of P can be pinpointed.

Our specific contributions are as follows: (i) a formalisation of a re-computation framework under our assumptions, (ii) a discussion of the role of provenance and of how granular provenance translates into efficient re-computation through precise selection of the re-comp scope, and (iii) an illustration of the framework in action on a real-world process of analysis of human genetic variants.

This research is part of the *ReComp* project, which aims to offer models for estimating the impact of changes in input and external data on the outcome of a program, in order to prioritise re-computation over the affected population vis-à-vis a limited budget.

This short paper should be read as an extended abstract. A more complete tech report is available online: `https://arxiv.org/abs/1604.06412`.

Related Work. As mentioned, Panda (Ikeda et al. 2011) collects and exploits provenance to enable data refresh, by selecting the fragments of a data-intensive workflow that must be re-executed. The focus here is on white box computations which involve database operations, which are documented using perfect and granular provenance records. A formal definition of correctness and minimality of a provenance trace with respect to a data-oriented workflow is proposed by members of the same group (Ikeda et al. 2013), leading to a notion of *logical provenance*. Although this may become a potentially useful building block for a future version of this work, it completely ignores the PROV data model (Moreau et al. 2012) which, instead, we regard as a practical foundation to enable interoperability of any provenance-based re-computation framework.

TaPP 2016, June 8–9, 2016, Washington, DC.

A similar perspective to Panda is taken in the Archived Metadata and Provenance Manager (AM&PM) (Gao and Zaniolo 2012), with a focus on database provenance and where the main evolving element is not the data but the database schema. Accordingly, the provenance of schema evolution is captured and can be queried, along with the provenance of the data in the current and past versions of a database.

Using the Prism schema evolution language (Curino et al. 2008)) leads to a formal definition of what here we call a *diff* function, aimed at quantifying the difference between two schemas. That research is vaguely related to our work, which does not specifically address database operations, placing schema evolution out of scope.

Also loosely related to the problem of determining the scope of re-computation is the idea of reusing some of the results and thus effort from past computations, using memoization (Pugh and Teitelbaum 1989).

Finally, as an infrastructure mechanism to enable selective re-computation, the *strong links* approach of (Koop et al. 2010) is relevant in this context.

Example: analysis of human genetic variants. The Simple Variant Interpretation (SVI), process, which we implemented in the *Cloud-eGenome* project (Missier et al. 2015), provides a simple interpretation of human variants to facilitate clinical diagnosis of genetic diseases. A *variant* is a single nucleotide mutation that occurs on a gene. Variants are identified by processing a patient's exome using a sequence of algorithmic steps that, essentially, compare it to a reference genome. SVI takes all variants found in the patient's exome (about 25,000) and a set of terms that describe the patient's *phenotype*, which indicates the patient's *disease hypothesis* (presumed disorder). It selects a small subset of the variants which are relevant for the phenotype, and associates a degree of estimated deleteriousness to each of them. To do this it uses knowledge from reference databases, namely the ClinVar (`www.ncbi.nlm.nih.gov/clinvar`) and OMIM Gene Map (`www.ncbi.nlm.nih.gov/omim`) databases, described in more detail later.

While the presence of deleterious variants may represent conclusive evidence in support of the disease hypothesis, the diagnosis is often not conclusive due to missing information about the variants, or to lack of knowledge in the reference databases about their association with the hypothesis. Thus, the diagnosis is dependent on the content of the reference databases. As this knowledge evolves and these resources are updated, there are opportunities to revisit past inconclusive diagnoses, and thus to consider re-computation of the associated analysis. To appreciate the effect of changes in the reference knowledge, in the Appendix (Fig. A) we show how new additions to OMIM and ClinVar would have affected the ability to carry out a conclusive diagnosis on a cohort of patients. The charts show the number of genes and variants within a gene, respectively, known to researchers and which would have been relevant for those patients. The charts in Fig. 3 provide a similar view of the evolution over time of the genes known to be implicated in Parkinson's and Alzheimer's diseases.

The *ReComp* problem in this use case involves (i) selecting the cases that are likely to benefit from re-computation, (ii) deciding whether complete or partial re-computation is required, and (iii) actually reproducing the original process, possibly requiring a new deployment.

2. Re-computation framework

We now present the main *ReComp* framework elements.

Computation. Consider program P executing on a set $\mathbf{x} = \{x_1 \dots x_n\}$ of inputs and producing outputs $\mathbf{y} = \{y_1 \dots y_m\}$, which also makes use of external data resources, or *data dependencies* $\mathbf{D} = \{D_1 \dots D_m\}$ where each D_i is a dataset, $D_i = \{d_{i1}, d_{i2} \dots\}$. We also associate a version v to each execution. This indicates a timestamp and uniquely identifies one execution of P, denoted by:

$$\mathbf{y}^v = P^v(\mathbf{x}^v|\mathbf{D}^v) \tag{1}$$

Transparency. The *transparency* of P is the level of detail available in observing a computation of P. This includes (a) details on the internal structure of P, and (b) details on which subset of each D_i are used. At one end of the "transparency spectrum", no details are available for either (a) or (b): P is a black box providing no details about its internal structure, and all we know about D_i are coarse-grain statements like "ClinVar was used". On the opposite end of the spectrum, P is a *white-box*, described for instance by function composition $P \equiv P_r \circ \dots \circ P_1$, and we also understand the semantics of each subprocess P_j and know the subset of D_i that was used by any P_j.

Provenance. The provenance of an output $\mathbf{y}$, denoted $prov(\mathbf{y})$, is a PROV document that describes the derivation of $\mathbf{y}$ from $\mathbf{x}$ through P using elements of $\mathbf{D}$. The granularity of PROV assertions depends on the transparency of P. In the most granular case, when P is a white box we can for instance express the usage of any single element $d_{ij} \in D_i \in \mathbf{D}$ by an activity P_k, i.e. using statements of the form:[1]

$$\texttt{used}(P_k, d_{ij}, [\texttt{prov:role} = \texttt{'dep'}]) \tag{2}$$

where the role indicates that d_{ij} is a dependency. Similarly, for inputs x_i (or intermediate values) we can write:

$$\texttt{used}(P_k, x_i, [\texttt{prov:role} = \texttt{'input'}]) \tag{3}$$

In a completely black box scenario, on the other hand, the assertions will be of the form:

$$\texttt{used}(P, \mathbf{D}, [\texttt{prov:role} = \texttt{'dep'}]) \text{ (use of dependency)} \tag{4}$$

$$\texttt{used}(P, \mathbf{x}, [\texttt{prov:role} = \texttt{'input'}]) \text{ (use of input)} \tag{5}$$

In addition to producing $prov(\mathbf{y})$, each computation of the form (1) also generates *history record* h:

$$h(\mathbf{y}, v) = \langle P^v, \mathbf{D}^v, \mathbf{x}^v, prov(\mathbf{y}^v), cost(\mathbf{y}^v)\rangle \tag{6}$$

where it is expected that $prov(\mathbf{y}^v)$ contains statements that make references to P^v, $\mathbf{x}^v$, and $\mathbf{D}^v$. Over time, statements of the form (6) form a *History database* H. Note that we also record the $cost(\mathbf{y})$ of computing $\mathbf{y}$ by executing P on $\mathbf{x}$. In practice this will be expressed as a monetary cost (e.g. when P is executed on a public cloud), execution time, resource usage or as a combination of those.

Change detection. *ReComp* relies on the ability to detect and quantify changes between any two versions of $\mathbf{x}$ and $\mathbf{D}$, i.e. $\mathbf{x}^v \rightarrow \mathbf{x}^{v'}$, $\mathbf{D}^v \rightarrow \mathbf{D}^{v'}$. Thus, we assume there exist three families of *diff* functions that are needed to compare two versions of the elements of $\mathbf{x}$, $\mathbf{D}$, and $\mathbf{y}$.

$$\text{input diff: } \{diff_{in}(x_i^v, x_i^{v'}) | x_i^v \in \mathbf{x}^v, x_i^{v'} \in \mathbf{x}^{v'}\}$$

$$\text{dependency diff: } \{diff_{d}(D_i^v, D_i^{v'}) | D_i^v \in \mathbf{D}^v, D_i^{v'} \in \mathbf{D}^{v'}\}$$

$$\text{output diff: } \{diff_{out}(y_i^v, y_i^{v'}) | y_i^v \in \mathbf{y}^v, y_i^{v'} \in \mathbf{y}^{v'}\}$$

These operate independently on each input, dependency, and output component. Each of these functions will have a different signature, and produce a summary of changes found in its inputs, in a format that may vary depending on the types of $\mathbf{x}$ and $\mathbf{D}$. For instance, $diff_d(D_i^v, D_i^{v'})$ typically computes the symmetric difference $(D_i^v \setminus D_i^{v'}) \cup (D_i^{v'} \setminus D_i^v)$. Other types of *diff* functions can

[1] PROV also allows to express that the d_{ij} are members of a *collection* $\mathbf{D}_i$.

be defined for specific use cases. Note that, although changes in the structure of program P are also relevant and are within the general *ReComp* framework, for simplicity we are going to assume that P does not change.

Role of the H database and of provenance. Upon detecting changes, i.e. using the *diff* functions, the first steps in making re-computation decisions include (i) *scoping rules*, that is selecting the subset $H' \subset H$ of the computations described in H that are affected by these changes, and (ii) defining the starting point of a *partial* re-computation of P, which we call the *starting component* P_s of P. This is the component of P mentioned in the earliest usage of a changed dataset (input or dependency), and it is not necessarily the same as the start of the whole of P. Note that partial re-computation is only possible if the input to P_s is available, i.e. not only should the input be explicitly mentioned in $prov(\mathbf{y})$, but it must also have been cached in a data store.

In a white box scenario, both steps can be addressed by querying the provenance documents in H. We distinguish the case of a change in inputs $\mathbf{x}$ from the case of a change in a dependency $D_i \in \mathbf{D}$. These correspond to the two patterns (3) and (2) above. Specifically, if the change $x_i^v \rightarrow x_i^{v'}$ involves any of the inputs $x_i \in \mathbf{x}$, the scope H' is simply the set of records h in which x_i^v is used as input, i.e. all $h(\mathbf{y}, v)$ such that $prov(\mathbf{y}^v)$ includes the pattern of form (3).

Regarding dependency change $D_i^v \rightarrow D_i^{v'}$, the affected records are those where the computation involved elements in $\mathit{diff}_d(D_i^v, D_i^{v'})$. These are the $h(\mathbf{y}, v)$ such that: (i) $prov(\mathbf{y}^v)$ includes the pattern of form (2) involving data element d_{ij}, and (ii) $d_{ij} \in \mathit{diff}_d(D_i^v, D_i^{v'})$.

Next, within the scope determined as above, we need to determine the starting component P_s of each P. The provenance patterns (3) and (2) suggest that P_s is the activity P_j that appears in the *earliest* occurrence of a usage statement involving a changed input or dependency.

Finally, note that in a black box scenario, with either limited visibility of process structure and/or of data input granularity, the scoping rules cannot be used, i.e. the default scope is the whole of H, and total (as opposed to partial) re-computation of P is required.

3. Detailed use case: SVI re-computation

We now illustrate the framework in use on our SVI case study. One execution of SVI, illustrated in Fig. 1, is carried out for each patient within a large cohort. SVI is an example of process P with inputs:

$$\mathbf{x} = [varset, ph]$$

where $varset$ is the set of variants associated with the patient, and $ph = \{dt_1, dt_2, \dots\}$ is the phenotype expressed using *disease terms* dt_i from the OMIM vocabulary, for example *Alzheimer's*. SVI is a classifier that associates a class label to each input variant depending on their estimated deleteriousness, using a simple "traffic light" notation, i.e.:

$$\mathbf{y} = \{(v, class) | v \in varset, class \in \{\mathsf{red}, \mathsf{amber}, \mathsf{green}\}\}$$

SVI's data dependencies $\mathbf{D}$ consist of the two reference databases, OMIM and Clinvar, along with their version v: $\mathbf{D}^v = [OM^v, CV^v]$ and subject to periodic revisions. OMIM maps human disorder terms dt to genes that are known to be broadly involved in the disease, denoted $genes\ (dt, OM^v)$. ClinVar maintains a catalogue V of variants, and it associates a status to each variant $var \in V$, denoted $varstatus(var, CV^v) \in \{\mathsf{unknown}, \mathsf{benign}, \mathsf{pathogenic}\}$.

SVI uses versions OM^v and CV^v of OMIM and ClinVar to investigate a patient's disease, as shown in Fig. 1. Firstly, the terms in ph are used to determine the set of *target genes* that are relevant for the disease hypothesis. These are defined as the union of all the genes in $genes(dt, OM^v)$ for each disease term $dt \in ph$.

Secondly, a variant $var \in varset$ is selected if it is located on the *target genes*. Finally, the selected variants are classified as red, amber, or green depending on $varstatus(var, CV^v)$.

To illustrate the process consider two patients, Patient 1 diagnosed with Alzheimer's, while Patient 2 is presumably affected by Parkinson's. Since the 90s, two genes have been known to be loosely implicated in these diseases, `PSEN2` and `PARK2`, respectively:

$$\texttt{PSEN2} \in genes\ (\texttt{Alzheimer's}, OM^{1995}),$$
$$\texttt{PARK} \in genes\ (\texttt{Parkinson's}, OM^{1995})$$

However, it was not until 2015 that two specific variants situated on those genes, at position 227083249 and 161807855, respectively have been studied and added to ClinVar. Thus, until 2014 we had

$$varstatus(227083249, CV^{2014}) = \mathsf{amber},$$
$$varstatus(161807855, CV^{2014}) = \mathsf{amber}$$

because neither variants were known to ClinVar.

Diff functions. For OMIM, $\mathit{diff}_{OM}(OM^v, OM^{v'})$ returns the set of terms $t \in T$ for which the mapping to genes has changed:

$$\mathit{diff}_{OM}(OM^v, OM^{v'}) = \{t \in DT | genes(t, OM^v) \neq genes(t, OM^{v'})\}$$

while $\mathit{diff}_{CV}(CV^v, CV^{v'})$ returns set of variants $var \in V$ with changed status, as well as new variants, or removed variants:

$$\mathit{diff}_{CV}(CV^v, CV^{v'}) = \{var \in V | varstatus(var, CV^v) \neq varstatus(var, CV^{v'})\} \cup CV^{v'} \setminus CV^v \cup CV^v \setminus CV^{v'}$$

Use of provenance. The provenance from each SVI tool execution is recorded in the H database. In our *white box* scenario, the relevant PROV assertions generated from an execution of SVI, with block names as in Fig. 1, are as follows:

$$\texttt{entity}(\texttt{om}, [\texttt{prov:type} = \text{'OMIM'}, version = \texttt{'v'}]) \quad (7)$$
$$\texttt{entity}(\texttt{ph}, [\texttt{prov:type} = \text{'prov:collection'}]) \quad (8)$$
$$\texttt{entity}(\texttt{cv}, [\texttt{prov:type} = \text{'CV'}, version = \texttt{'v'}]) \quad (9)$$
$$\texttt{entity}(\texttt{vars}, [\texttt{prov:type} = \text{'prov:collection'}]) \quad (10)$$
$$\texttt{used}(\texttt{PtG}, \texttt{om}, [\texttt{prov:role} = \texttt{'dep'}]) \quad (11)$$
$$\texttt{used}(\texttt{PtG}, \texttt{ph}, [\texttt{prov:role} = \texttt{'input'}]) \quad (12)$$
$$\texttt{used}(\texttt{vClass}, \texttt{cv}, [\texttt{prov:role} = \texttt{'dep'}]) \quad (13)$$
$$\texttt{used}(\texttt{vClass}, \texttt{vars}, [\texttt{prov:role} = \texttt{'input'}]) \quad (14)$$

Note that the `used` assertions are of the form (2) and (3), respectively. These provenance statements can be used to define scoping rules and starting components, as follows.

Re-comp scope due to OMIM changes. The executions h in the re-comp scope following change $OM^v \rightarrow OM^{v'}$ include those where phenotype ph includes terms in $\mathit{diff}_{OM}(OM^v, OM^{v'})$, i.e., those with changes to their gene mappings. The phenotype is found in (12), while the version of OMIM for computing diff is found using (11). As (11) contains the earliest mention of `om`, `PtG` is also the starting component for re-computation.

Re-comp scope due to ClinVar changes. Similarly, following change $CV^v \rightarrow CV^{v'}$, the executions in scope are those that include selected variants on target genes and which appear in $\mathit{diff}_{CV}(CV^v, CV^{v'})$. Using the provenance fragment above, the

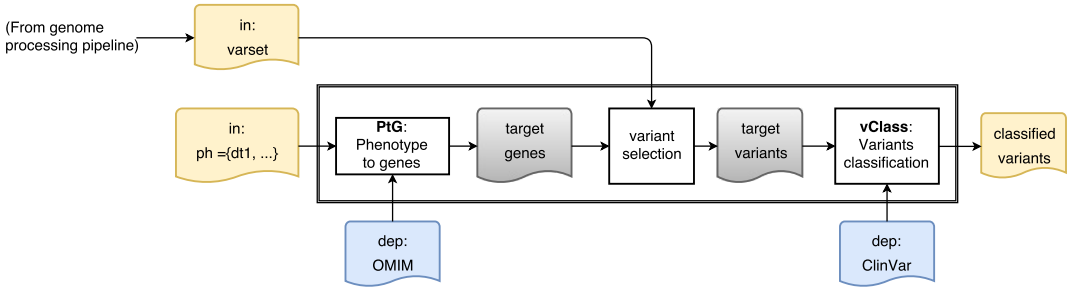

Figure 1. *White box* SVI, with inputs $\mathbf{x} = [varset, ph]$ and data dependencies $\mathbf{D} = [\text{OMIM}, \text{ClinVar}]$

selected variants are found in (14), and the version of CV for computing diff is found using (13). In this case, `vClass` is the starting component for re-computation following a change in ClinVar.

Example, continued. Consider again variants 227083249 and 161807855. Because they are both located on genes that have been known to OMIM, these variants are selected as candidates for testing against ClinVar. As mentioned, until 2014 they were both classified as `'amber'`. Having been added to ClinVar in 2015, however, they both appear in the latest diff between the 2014 and 2015 versions of ClinVar:

$$\{\texttt{227083249}, \texttt{161807855}\} \subset diff_{CV}(CV^{2014}, CV^{2015})$$

According to the scoping rule above, the executions of H where the provenance mentions 227083249 and 161807855 are now in scope, and these include patients 1 and 2 (possibly along with many others for whom these variants are relevant). As 227083249 is catalogued as "probably pathogenic, uncertain significance", the diagnosis for patient 1 is still inconclusive. For Patient 2, on the other hand, we can rule out variant 161807855 as a cause of their disease, as this variant is now known to be benign.

4. Conclusions

Knowledge assets derived from data analytics computations may decay and become obsolete as the datasets or the content of reference data resources used to produce it change over time. While this suggests that re-computation of such knowledge assets may be needed, deciding precisely which of them should be re-computed is not a trivial problem; it requires meta-knowledge about their dependencies on the inputs and on the reference datasets.

In this paper we have discussed the role of provenance in supporting re-computation decisions when results from data-intensive processes are progressively invalidated by the evolution of the underlying data. We have presented a simple reference framework in which data is versioned and functions are available to compute the differences between any two versions. We have clarified how fine-grained and coarse-provenance can be used to assess the impact of such differences on a history of past computations, with different precision, suggesting which past computations should be performed anew. We have illustrated these ideas through a detailed example, concerning the automated classification of human variants for clinical diagnosis. A more complete account of the approach is available online: `https://arxiv.org/abs/1604.06412`.

Acknowledgments

This work is funded in part by UK EPSRC grant no. EP/N01426X/1.

References

C. A. Curino, H. J. Moon, and C. Zaniolo. Graceful Database Schema Evolution: The PRISM Workbench. *Proc. VLDB Endow.*, 1(1):761–772, aug 2008. ISSN 2150-8097. doi: 10.14778/1453856.1453939.

S. Gao and C. Zaniolo. Provenance Management in Databases Under Schema Evolution. *Proceedings of the 4th USENIX Conference on Theory and Practice of Provenance*, (iii):11, 2012.

R. Ikeda and J. Widom. Panda: A system for provenance and data. *Proceedings of the 2nd USENIX Workshop on the Theory and Practice of Provenance TaPP10*, 33:1–8, 2010.

R. Ikeda, S. Salihoglu, and J. Widom. Provenance-based refresh in data-oriented workflows. *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1659–1668, 2011. doi: 10.1145/2063576.2063816.

R. Ikeda, A. Das Sarma, and J. Widom. Logical provenance in data-oriented workflows? In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 877–888. IEEE, apr 2013. ISBN 978-1-4673-4910-9. doi: 10.1109/ICDE.2013.6544882.

D. Koop, E. Santos, B. Bauer, M. Troyer, J. Freire, and C. T. Silva. Bridging workflow and data provenance using strong links. In *Scientific and statistical database management*, pages 397–415. Springer, 2010. ISBN 3642138179.

P. Missier, E. Wijaya, R. Kirby, and M. Keogh. SVI: a simple single-nucleotide Human Variant Interpretation tool for Clinical Use. In *Procs. 11th International conference on Data Integration in the Life Sciences*, Los Angeles, CA, 2015. Springer.

L. Moreau, P. Missier, K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes. PROV-DM: The PROV Data Model. Technical report, World Wide Web Consortium, 2012.

W. Pugh and T. Teitelbaum. Incremental Computation via Function Caching. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 315–328, New York, NY, USA, 1989. ACM. ISBN 0-89791-294-2. doi: 10.1145/75277.75305.

A. Supporting material – Knowledge evolution in genomics

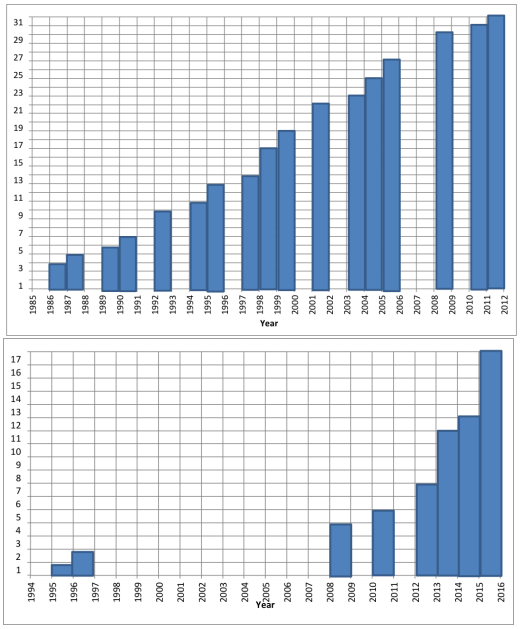

Figure 2. Increase in the count of genes (top) and variants (bottom) over time, related to diseases affecting a cohort of patients at the Institute of Genetic Medicine,Newcastle.

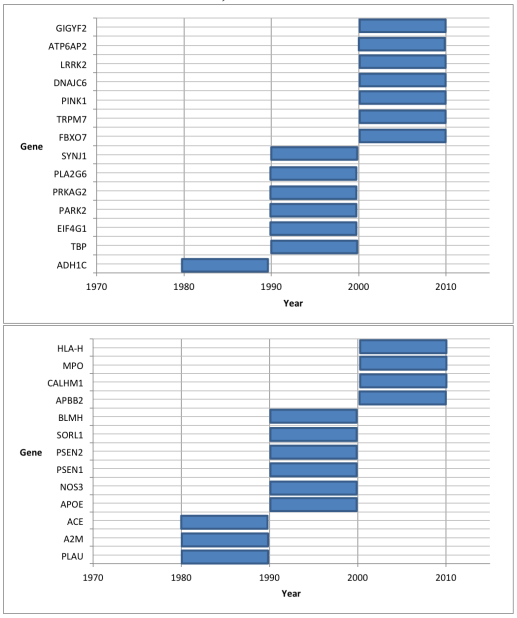

Figure 3. Progressive increase in the count of genes known to be involved in Parkinson's and Alzheimer's over time, in the OMIM Gene Map database.