26th Meeting of the Wiesbaden Group on Business Registers
- Neuchâtel, 24 – 27 September 2018

Jakob Hruby, Dipl-Ing.
Statistik Österreich

Session 6: Profiling and Globalisation

An Iterative Enterprise Group Algorithm

Abstract

Keywords: Business Relationships, Business Ownership and Control, Iterative Group Building

For statistical as well as for legal purposes it is vital to have a solid understanding of group membership of firms and of the roles that human controllers play in this context. Major focus in Statistical Business Registers is laid on the issue of the interconnectedness of the various forms of legal entities amongst each other as well as on their links to human owners and controllers.

The paper describes an algorithmic methodology to derive groups of enterprises for a given set of firms and people, provided that only one-to-one-relationships are known.

It, furthermore, explores common types of links such as ownership, capital shares, other controlling relationships and manually evaluated control and how to prepare the data and the algorithm to combine them for statistical use. It also deals with the problems arising from such connections and their combinations like dealing with circular ownerships and contradictory information.

Typical solutions to this problem are recursive by nature. They tend to have a rather long runtime and either carry out the same operations multiple times or have to use extra storage for caching information. This is impractical - in particular - when working with large sets of data and large groups for which business group involvement needs to be kept up to date, which in general is the task that business registers have to carry out.

The proposed methodology uses an iterative approach, which converges towards the final result in very few steps without either of these problems. It was motivated by mathematical induction, a simple, yet, very powerful proof technique, and uses the property that, if properly prepared, each one-to-one-relationship already represents a group of a certain kind itself, and that the iterative addition of further members to such a group does only add to that group but cannot change another legal member's membership status.

This method is currently in use in the Austrian Statistical Business Register as well as in the Austrian Beneficial Ownership Register.

1) Problem description and preparing of data

For deriving enterprise groups we start off with the legal units and those people or legal units, who/which are known to be directly related to a certain legal unit, for example as shareholders, owners, operatives or as any other roles recorded in the Business Register.

For each type of role and each legal unit we have to quantify to what percentage of control this role leads. For example, an owner is given a complete, so in numbers 100 percent control over a legal unit, a shareholder is given that amount of control, which the capital held by this person amounts to, and an operative is given no share of control at all. Defining how much control each role gets is the first important question we have to answer when trying to define enterprise groups. Although in general this seems to be straight forward, it is last but not least a legal question which roles are given what amount of control in which type of legal form.

Answering that legal question, this approach leads to a set of data in which we have quantified one-to-one relationships between a legal entity, called the child, and a person or legal entity that exerts a certain percentage of control over that child, called the parent.

Child	**Parent**	**Share**
Example LLC	John Doe	20%
Example LLC	Shareholders United LLC	80%
Sample LP	Samantha Sample	100%
…	…	…

In practice we use unique numeric identifiers in the place of names.

This is still a rather raw form of data and we should deal with some common issues that may arise from such data:

Doublets: In some cases, the same parent unit may play multiple roles that lead to shares held of the same child. When preparing the data, it is advisable to sum the shares of such relations to form unique one-to-one relations.

Zero percent relations: In some cases, for example with capital shares of an LLC, we may encounter data that leads to zero percent relations. For all purposes of building groups these relations can be treated as if they did not exist. Hence, it is reasonable to discard them beforehand, so that algorithms do not even have to work with such entries.

Self-control: In rare cases a unit may hold shares of itself. Since a legal unit cannot really exist on its own, but needs human control to make decisions, such shares should generally be discarded. This naturally leads to other shares held of that unit amounting to higher percentages, since the total amount of shares still needs to sum up to 100%.

Contradictory information: Sometimes, in particular when handling multiple sources of information at the same time, we may hit upon cases in which we have different pieces of information from different sources for the same enterprise. In such a case we have to rank the sources by priority and exclusively handle information given by the corresponding source. For example, we may have an old company register entry telling us a certain distribution of capital of an LLC and a currently manually evaluated piece of information that a completely different person is holding the only capital owner of that unit. In this case we want to discard the whole piece of information given by the company register. Otherwise, combining those sources would lead to a confusing mixture of capital shares that would even exceed the total capital owned of that LLC.

2) The intuitive, recursive approach

Now that we have defined the size of the shares of one-to-one-relationships we need to connect these pieces of information, so that for every single enterprise we find that person or enterprise, that has direct or indirect control over it, or prove that there is none of such kind in our data.

The criterion for **direct control** is straight forward; it means that an enterprise or person has more than 50% shares of an enterprise. Meanwhile the criterion for **indirect control** of a share is given when an enterprise directly or indirectly controls an enterprise that holds the share in question. If an enterprise has indirect control over a share and the sum of all shares directly and indirectly controlled exceed 50% we speak of indirect control.

There are two important observations that can be drawn from these definitions:

1) Direct control is always immediately visible from one-to-one relations. The concept to determine whether there is control in larger groups and networks of enterprises and people is of little interest, since a controller of a controller is always an indirect controller but never a direct one. Indeed, the concept of direct control is nothing but a part of the definition of indirect control, which can be seen easily, since the sum of all shares that are directly or indirectly controlled exceeds 50% if only direct control is given. As a matter of fact this means that this share alone exceeds 50% and additional shares only add to that sum. It is therefore sufficient to only talk about indirect control and refer to it as *control*.

2) Indirect control is defined recursively, meaning the concept is part of its own definition. This only makes sense if we can assure that this recursion does not lead to infinite looping when we try to determine whether a share is indirectly controlled. For this purpose our data has to meet three criteria:

A) We are not dealing with infinitely long networks. This is particularly given when our whole data set is finite to begin with, a criterion that is naturally fulfilled when dealing with real data.

B) At least one of the indirectly controlled shares is actually directly controlled. Otherwise our definition would come down to "The criterion for indirect control of a share is given when an enterprise indirectly controls an enterprise that holds the share in question.", which obviously is a tautology. Once again, the fact that with given real data we only deal with finite one-to-one relations helps with this, because then there is only one case when this can occur, which is when two enterprises control each other directly or indirectly. In all other cases after a finite number of recursions in our definition we will reach an enterprise or person that is not controlled further, and therefore the shares they hold can only be controlled directly.

C) As already argued in B) a troublesome case of data is possible in which enterprises control each other directly or indirectly. In such a case the given definition of (indirect) control becomes a tautology, which means it is ill-defined. There are two basic solutions to this problem. The first is to simply prevent this case by making it illegal by law to create such enterprise constructions. The second on is to change our definition(s) of control. A third one, namely the elimination of such data beforehand, is not possible by the simple fact that to find those occurences of control over one another we would first have to find the controllers, hence this would come down to changing our definition of control. Although we know from reality that such cases occur we will first examine the case in which our data is clean of such occurrences to begin with. Later on, we shall examine a way to handle other cases as well.

So, given that the definition of indirect control is recursive, it is only natural to design an algorithm, which finds the controllers of each registered enterprise, recursively. The following is a description of such an algorithm:

```
Function findController({Enterprise}, ListOfAllRelations)
        Enterprise.parents = emptySet
        Enterprise.shares = emptySet
        For Relation in ListOfAllRelations do
                If Relation.child = Enterprise then
                        add(Enterprise.parents, , Relation.parent)
                        add(Enterprise.share,  Relation.share)
                End
        End
        If Enterprise.parents=emptySet do
                Return (Enterprise)
        Else
                UpperPar=FindController({Enterprise.parents}, ListOfAllRelations)
                SumParents=Sum(Enterprise.shares, by=UpperPar)
                m = max(SumParents)
                p = position(m, SumParents)
                If m>50 then
                        Return (p)
                Else
                        Return (Enterprise)
                End
        End
End
```

In essence what this algorithm does, is, it asks an enterprise for its shareholders, which then does not return the direct shareholders, but instead the controllers of these shareholders, which are derived by the same algorithm. So the program recursively opens itself until it finds the upmost shareholders it can find and then checks whether these are direct controllers of their child enterprises. If so, it returns those as the parent, otherwise the child is said to control itself. This is the result which gets recursively returned until we hit upon the first instance of the program.

As with all recursive algorithms this one holds a few flaws that are natural to programs of this kind. But without going into further details about these issues, there is also another more specific problem

with this algorithm when one wants to apply it to a bigger data set with multiple enterprises: The algorithm wastes a lot of information by calculating the same owners over and over again.
This can be easily seen from a simple example with a person A and two enterprises B and C, with A holding 100% shares of B and B holding 100% shares of C and our task being to calculate the owners of B and C respectively.
So when deriving the owners of C with the presented program we will implicitly calculate the owners of B, and then get the return for C that A is the owner. Despite this, we will still have to run the program for B separately.
When we are working with bigger datasets and huge networks of enterprises this method becomes highly inefficient. A simple workaround would be to refine the program to cache information that has been calculated during the program execution, which is a rather technical task, requires extra storage, still requires extra checking for whether information has already been processed and cached, and still leaves us with the general problems with recursive algorithms.

Lastly, we still have not dealt with the problem of circular relations. To deal with this issue, we additionally need a dynamic blacklist of parents that is being tracked while executing the algorithm and which contains every unit that has already been handled.
With such a list, we ask whether this parent is already on the blacklist, and in case it is, we treat the parent as nonexistent (because we have dealt with that branch in the control relations already) every time we want to open up the program again for a parent. And if it is not, we add the parent to the blacklist, so that we will not jump to it in a consecutive recursion of the program.

With all these smaller and bigger issues of the recursive program, it may just be more reasonable to look for a different solution.

3) An Iterative Enterprise Group Algorithm

For the new approach we want to take, we change our view on the data. Instead of thinking of the data as interconnected enterprises we will see them as groups of enterprises, their subgroups and their members.
A group describes the maximum network of interconnected members under the leadership of a single enterprise or person. A subgroup, however, can be any interconnected part of a group under the leadership of a (sub-)leader. It does not need to be the maximum group of enterprises that have a common leader. This in particular implies that any one-to-one relationship that signals direct control is already a subgroup. So, in this sense, our starting information already tells us about subgroups and the task now is to extend these groups until they are maximum groups.

Similar to the way that for the recursive algorithm we have defined control criteria we now will define a membership criterion for our new approach.
We determine that an enterprise is a member of a group, if the direct one-to-one relationships controlled by members of these groups sum up to more than 50 percent.

In contrast to the former (indirect) control criterion we can immediately see that this criterion is not recursive. Therefore, to check whether an enterprise extends a given (sub-)group or not we do only have to check the direct relations of that enterprise but never the relations of those group members amongst each other.
For this approach to be taken into action we need to meet two sets of inputs:
1) The set of all enterprises we want to join into groups.
2) A set of (sub-)groups formed by these enterprises that we can iteratively extend with further enterprises from set 1 until no more extensions can be performed.

Set 1 is trivially given from our one-to-one-relations. Any enterprise that is either a parent or child in those relations is part of it.
For the second set we only have to observe a simple property which is that any one-to-one-relation with over 50% shares is already a (sub-)group that can be extended. (Alternatively, we could define any enterprise to be a (sub-)group for itself as a starting point and extend those which then in the first iteration step would extend those groups by those members that are controlled by more than 50% by a single other enterprise)

The iterative algorithm now is defined by taking the set of all one-to-one relations, adding all relations that lead from a child into the same group and in case of the group jointly holding more than 50% of the shares, adding the child to the group. We also immediately add all children of that enterprise child to the group.
And then we take that set of extended (sub-)groups as input and repeat this process. Since we added members to the group in the last step, unless all our groups are already at the maximum number of members, this will lead to some sums of shares that are jointly held by groups increasing. Iterating this process until it has stabilized - so no sums of shares have increased within a step - will give us the maximum groups without ever having to check a recursive control criterion.
This also in part deals with the former problem of the recursive algorithm that is to say we cannot perform the algorithm on sets of relations that lead to circular control.
In this approach, a circular relationship will lead to (sub-)group leaders switching back and forth during the process at worst, but it does not have any implication on the process of determining an enterprise´s membership status to the (sub-)group! The only annoying implication is that the ongoing switching of group leaders makes it hard to determine whether the process has already stabilized or not. The simplest solution necessary for that end is to simply estimate the amount of iterations needed to create maximum groups and then perform a hard-coded adequate amount of iterations instead of dynamically determining whether the necessary amount of iterations has been met. In our experience the necessary amount for Austrian enterprise groups lies in the range of 5 to 6 iterations (leading to a maximum depth of groups of about 9 members) and the performed amount of iterations is set to be 20.

A more formal description of the algorithm is given by first extending our data model for relations to the following:

Child	**Parent**	**Share**	**Group Head**	**Combined Share**	**Previous Combined Share**

This adds information as to which group this one-to-one relationship leads into (identified by the Head of the Group) and how many shares from the child this group jointly holds. The Previous Combined Share is there to track the change in shares between iteration steps. As a starting point we set the values for Group Head to the Parent, the Combined Share to the Share (which the parent holds) and the previous Combined Share to zero.

Our iteration step is now defined as:
1) Replace the Combined Share of all relations with the sum of all Shares of entries for which both the Child and Group Head entry match. This gives us the newly combined sum of Shares that a Group Head holds directly or indirectly of a child, according to the information that is currently implied by the structure of our so-far determined (sub-)groups.
2) Determine all relations for which the Combined Share has grown to above 50% from under 50% in the last iteration step, by comparing the Combined Share with the Previous Combined Share. (The algorithm is set up in a way that this value can only grow over time, but never decline again. This

makes sense as our algorithm only explores possible further connections into a (sub-)group. But once it leads into a (sub-)group, it can never stop doing so!)
These are our new group members in those groups that are given by their Group Head attribute.
3) For all children of the relations determined in 2), hence the new members of groups, we determine those relations in which these children are Group Heads and replace the Group Head in that relation with the Group Head from the in Step 2) determined relation. This means that a new member that is joining a group tells all their (direct and indirect) children which group their relation leads into.
4) The last step is simply to determine whether our break condition has been fulfilled. As argued previously, due to circular relations, the simplest condition is simply to stop after a certain amount of iterations. If the condition is not met, we simply start again at step 1).

Once this iteration has been completed we have a set of data in which for every child-parent relation we hold the information which group head this relation leads to and how many shares are (directly or indirectly) controlled by that group head. For the combined shares amounting to over 50% we say that the child is a group member of the group led by the group head.

4) Experiences and Applications of the Algorithms

In the past year the Austrian business register has switched from a Java-based recursive solution to an SAS/SQL-based iterative solution. We have since cut the runtime from multiple hours to a few minutes for a few hundred thousand relations processed by the programs. (As should be, the results of the calculations on equivalent sets of data have been the same)
However, to calculate the (indirect) controllers of single units the recursive algorithm does have its advantage, as the iterative approach always calculates all groups for all given enterprises at once, which still takes longer than running the recursive program for a single unit. Singling out only those units which are necessary for a single unit's calculation in the iterative approach is also unreasonable, as this is simply one of the major tasks the recursive algorithm implicitly performs anyway.
Since the switch last year, we have improved the concept of the iterative approach to process further information like the depth of enterprise groups with it.