A Ranking Tool for the Amboseli Baboon Research Project

4/22/07

Tyler Brock
Computer Science 192
Advisor: Jun Yang

The Amboseli Baboon Research Project is a long-term coordinated series of studies of yellow baboons, Papio cynocephalus, in the Amboseli region of East Africa, immediately north and west of Mt. Kilimanjaro.[1] The data collected in the process of these studies is retrieved, stored, and maintained within the Babase system, designed to facilitate working with and learning from the information. Most importantly, the project aims to better understand the sociality of baboons and in doing so, the nature and importance of rank, relatives, and social bonds. However, as time passes and technology progresses, the forms of organizing, viewing, interacting with, and manipulating data changes tremendously. This project is a way of leveraging state of the art technology to gain more from the Amboseli research.

The main goal of this project is to create an updated ranking tool for use with the available information in the Babase system. While in the past it may have been overwhelming to write a graphical interface, if one was at all available or fathomable, it is now standard practice and expected to have a graphical component to most software that can work in powerful ways with the data. The previous implementation of the Babase system involved the use of FoxPro, a relational database management system, which is currently outdated and no longer widely in use. Additionally, the Amboseli project was limited by a text-based data manipulation tool and unable to harness the power and flexibility of a graphical interface. That is why Java was chosen as the programming language for the ranking tool's successor. In an effort to ensure longevity and cross platform extensibility as well as further empower the members of the research group, it was the best programming language suited to the task at hand and has the best available tools for the programmer.

Technical

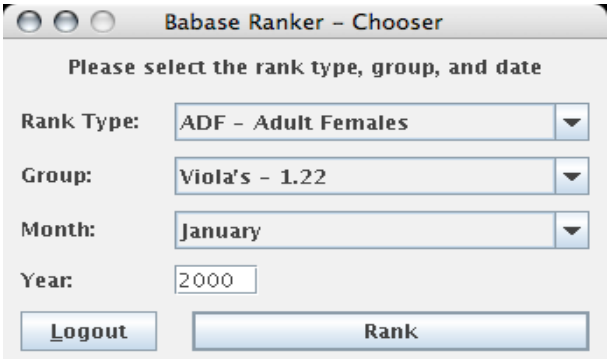
Technically, there is a lot that went into making the new Babase ranker. Papio, the main server with a new database management system, PostgreSQL, has a high level of security, as is to be expected, and there is no way to access the computer from outside of the lab without the use of a virtual private network, or VPN. As a result, all client

[1] http://www.princeton.edu/%7Ebaboon/

computers that use the new Babase ranking tool will need to have a compatible VPN client installed and configured on their computer in order to be able to run the program as the program accesses the database directly, using the PostgreSQL JDBC driver, even though it can be launched over the web. Once someone has unfiltered access to Papio's services, they will be able to launch the JNLP Java web start application from the main Babase website. All necessary jar files will be downloaded, and the necessary database connections will be tunneled through the virtual private network. All jar files related to the project need to be signed with a certificate from the biology department so that they can be considered secure.

Interface

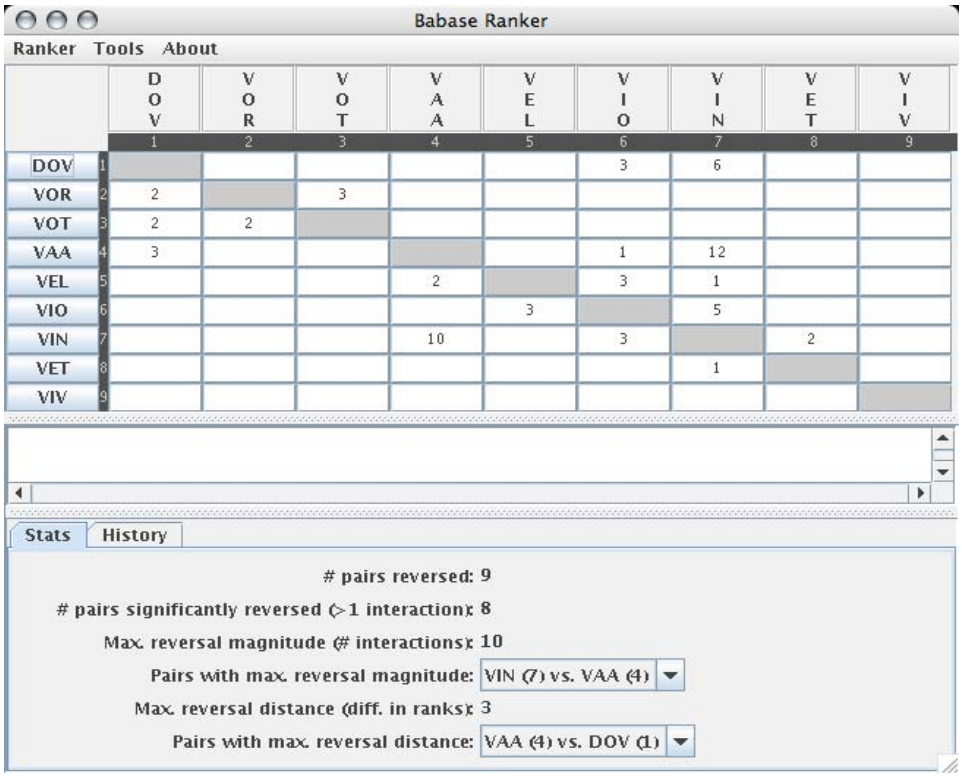
The interface for the ranking tool is simple and elegant. At initial launch, the user is greeted with a login window and the options to select a database, enter in a username and password and login or quit. Without the proper credentials, a user will be denied access. After successfully logging in, the user is again presented with a set of options related to selecting a group of individuals they wish to rank in the database they have previously selected. The drop down lists allowing choices for the rank type and available groups are populated dynamically from database to ensure that the interface limits the users choices to possible choices for these fields. The months field is statically populated but allows the user to choose a month for the year that they enter into the box below which must be of the 4 digit format. This choice was made as rankings are based on a month to month periods of data collected in between the first and last days of the month in a given year. Therefore, a day field is not needed. After entering the required information, the user is give the option to perform the specified ranking on the selected group and period or to logout in order to log back in as another user or to select an alternate data source. If the user chooses to continue with the ranking, he will be presented with the main ranker interface and interaction matrix. An image of the chooser interface is shown below.



The main ranker interface is also intuitive and easy to use. There are three menus in the menu bar with possible options for the user. The ranker menu functions as a file menu and includes actions that happen on the ranker or program-wide level, the tools menu provides various automatic ranking possibilities that modify the ranking in the interaction matrix, and the about menu provides information about the program. Within the ranker menu, the user has options to start a new ranking, or return to the ranking chooser, open or save ranking files in a human readable format, read or save a current ranking to the database, logout the user, or quit. The ability to quickly jump to a chooser for a new ranking without having to log back in is a great time saver in addition to the useful key mnemonics that allow users to run an automatic ranking and start a new one simply using keystrokes. There will be more information regarding the importance of the tools menu and the automatic rankings contained therein later on in the paper.

Beyond the menu bar, there is also an interaction matrix, a status window, and a tabbed panel tabs containing statistics regarding the current ranking and history of moves within the current ranking. The interaction matrix itself, once populated, has individual three letter short names, or snames, for each individual. On the left hand portion of the matrix, snames can be selected for swapping with another sname in order to switch their relative ranking positions in the ranking matrix, these are known as swappable headers. Across the top, snames can drag individuals across the matrix in order to rearrange their ranking in a sliding fashion, these are known as slidable headers. Ranks are shown next to the snames at the top of every column and to the left of every row. Therefore it is very easy to visualize and manipulate the rankings until a user is satisfied with the current state and store it in the database or locally for later use. Reading the ranking matrix is

done as follows: the row individual dominated, or was considered to have come out on top of in an agonistic interaction, the column individual however many times is represented by the count in the cell at that location. An image of the main ranker interface is shown below.



The interaction matrix itself is populated by querying the Babase PostgreSql database. Specifically, the Actor_Actees view is utilized as it joins the Parts, or participants table, on the iid key with the Interact_Data table as actor and actee in order to provide a list of interactions between given snames which we can easily count. The ranker filters this with the group data from the BIOGRAPH table in order to ensure that the individuals were actually in the user selected group during the time frame of the ranking and that they fit the rank type being requested. Using this result set of interactions it is possible to populate an interaction matrix whereby the cells represent the number of times individuals in row i dominated the individual in column j. Only individuals who are part of the group being ranked are shown even though there may be individuals who interact with members of the group yet which are not a member of it.

Rankings can be saved and read on the fly from that database although the ranking tool automatically loads the last ranking if there is one available for the selected criterion.

Automatic Ranking

While building the new ranking tool it was also a good chance to include the possibility of doing an automatic ranking based upon the interaction criteria in the database. It is extremely difficult to mimic the intricacies of a manual ranking done by a biologist and at best any automatic ranking algorithm designed by computer scientists would be a gross simplification of biological ranking rules. Yet, attempting to come up with a possible solution is an interesting exercise in the field of computer science. While a ranking algorithm we design may not be optimal in real situations, it will seek to optimize the rankings based upon available interaction data taking into consideration the idea that if one individual dominates another it should be ranked higher than the other individual. Any automatic ranking algorithm should attempt to preserve this idea in implementation.

There are many ways to determine a ranking. Given an interaction matrix, a ranking implementation could be as naïve as counting the number of wins for a given individual, implemented as the automatic domination ranking within the ranker, or use a matrix score ranking that is based on criteria such as the difference in the number of wins and losses for a given baboon. An example of a matrix score being calculated for the individual with the short name VEL can be seen below. Additionally, one could further increase the effectiveness of these algorithms by giving weight to transitive dominance using matrix multiplication. However, these methods are based only upon matrix computations and make no use of the directed nature of the data.

	DOV	VOR	VOT	VAA	VEL	VIO	VIN	VET	VIV
DOV						3	6		
VOR	2		3						
VOT	2	2							
VAA	3					1	12		
VEL				2		3	1		
VIO					3		5		
VIN				10		3			
VET							1		
VIV									

$$VEL = (2 + 3 + 1) - (3) = 3$$

One example of a ranking system that takes these ideas a step further is the I&SI method of linear ordering. This ranking system makes use of the concept of dominance, which De Vries describes as one individual, A, winning more encounters against B than B wins against A. If they have the same number of wins they are said to be equidominant but if no encounters are observed there can be no statements made on the dominance relationship between the two individuals. This method then goes on to implement the idea of an inconsistency, which is when an individual dominates another individual yet is ranked below that individual in the set of ordered ranks. Therefore, the first step in the I&SI method, according to De Vries, is to "find an ordering of individuals such that the number of inconsistencies called I is minimal."[2] However, if there are cycles within this linear ranking, where A dominates B who dominates C who, in turn, dominates A, a ranking of these individuals cannot be completed successfully using this method. DeVries states, "If there are circular triads (or circular polyads) present in the set of dominance relationships, not all inconsistencies can be resolved… Therefore in any one ordering of the individuals, at least one of these three dyads will be inconsistent."[3] Thus, in order to solve this problem he introduces the idea of a strength inconsistency.

A strength inconsistency is the value of the space in between ranks of two individuals, referred to in his paper as the SI portion of the I&SI method. The strength of

[2] DeVries, H. (1997). Finding a dominance order most consistent with a linear hierarchy: a new procedure and review. Ethology & Socio-ecology group, Utrecht University.

[3] Ibid.

the inconsistencies represents the amount which that particular inconsistency disturbs the overall ranking. De Vries' ranking method combines the original concept of inconsistencies and this logical extension by basing his algorithm on minimizing the number of I's in the matrix and then shrinking the total strength of the inconsistencies without increasing the number of I's.[4] While the I&SI approach to the ranking problem locally minimizes I, Wittemyer and Getz suggest that this concept can be built upon further to create a decent solution by repeating the algorithm with different initial orderings of individuals which they suggest are organized in the following manner:

1. *Individuals are first ordered by their win-loss ratio, where pairs with equal ratios are ordered by number of wins*
2. *A permutation technique is implemented that randomly switches individuals a random number of times from their initial ordering such that a spectrum of initial conditions are searched to ensure that the final solution is not just a local optimum.*
3. *The rank order for each starting position is then solved by inserting dominant individuals above their subordinates.*
4. *If the total change in the number of inconsistencies followed by the total rank sum difference of these inconsistencies increases as a result of the row insertion, the insertion is reversed.*
5. *The rank order is regarded as converged when the value of I followed by the value of SI remains unchanged after repeated permutation runs. Generally 100 runs is regarded as sufficient for locating the minimum of both the I and SI metrics.*[5]

However, this method seems haphazard at best as it is not very consistent and cannot produce a unique ranking order when data is incomplete or when there exist permutations

[4] Ibid.
[5] Getz, G. W. a. W. M. (2006). A likely ranking interpolation for resolving dominance orders in systems with unknown relationships. University of California at Berkeley, Department of Environmental Science, Policy, and Management.

of rankings that produce the same I and SI values. Therefore, our scheme builds upon some of these concepts and includes the use of both matrix computations and graph theory in attempting to find a likely optimal ranking that intuitively makes sense, even when data is incomplete or when there needs to be a method to break ties. The ranking algorithm works as follows:

Brock-Yang Ranking Algorithm

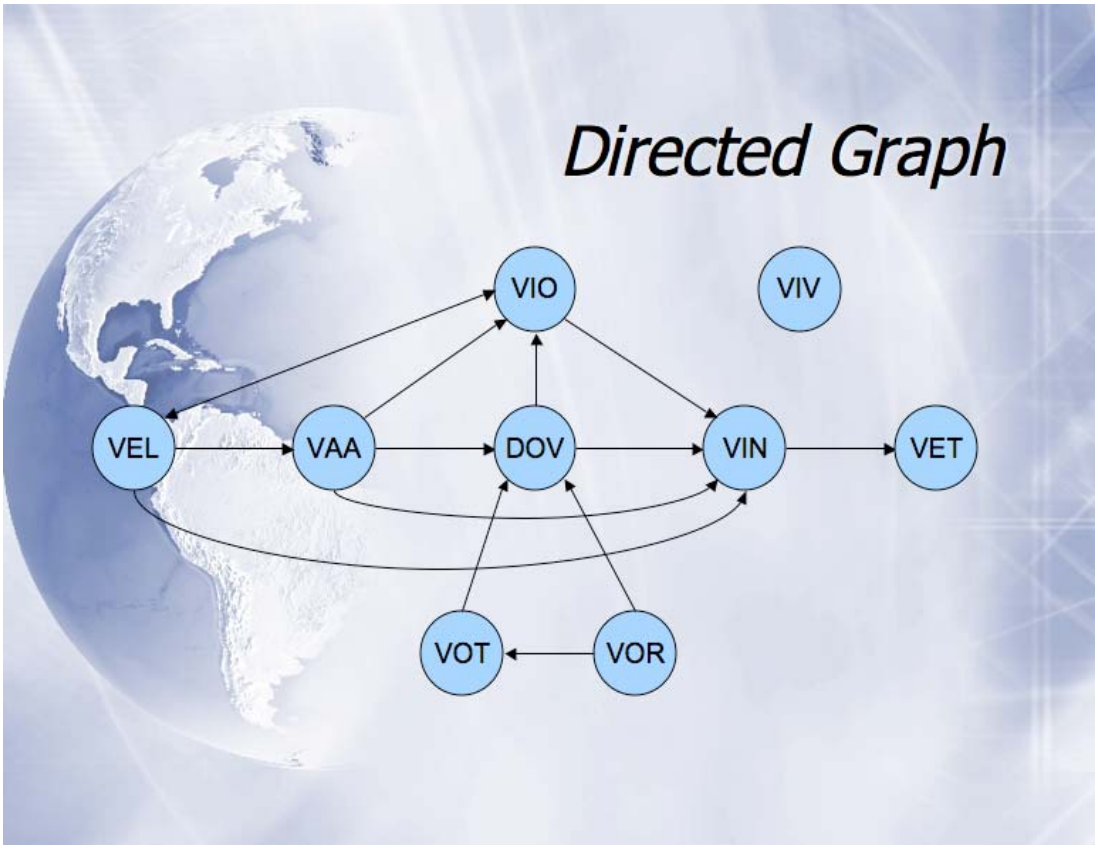
1. Construct a graph of interactions – a directed graph which may contain cycles
2. Find strongly connected components (SCCs) within the directed graph
3. Determine a ranks within SCCs –using matrix computations to break ties
4. Rank the whole graph using a topological sort

Create Directed Graph of Interactions

In order to construct a directed graph representation of the interaction matrix you first make nodes for all the individuals. Then, for every individual you traverse the matrix creating edges whenever individual i dominates individual j. Domination being defined as:

Individual i dominates Individual j when: [row i][col j] > [row j][col i]

This would indicate individual i has come out on top in more agonistic interactions. Intuitively, it makes sense that if one baboon wins more than another, it should be ranked above the other one where a directed edge indicates dominance. Additionally, two edges are introduced, creating a cycle, when there is a tie amongst individuals in the interaction matrix in order to signify that the order of dominance may go in either directed direction. Using the previous example, this following image is of a directed graph that would be constructed using the aforementioned method on the given interaction matrix.



This graph is impossible to rank using a simple topological sort as would be customary on a directed acyclic graph. However, because this graph contains strongly connected components, where every node in a strongly connected component can reach any other node in the strongly connected component, this is impossible. In attempting to resolve ranking orders of this type, if A dominates B who dominates C but C dominates A there is a contradiction that cannot be resolved using graph theory alone.

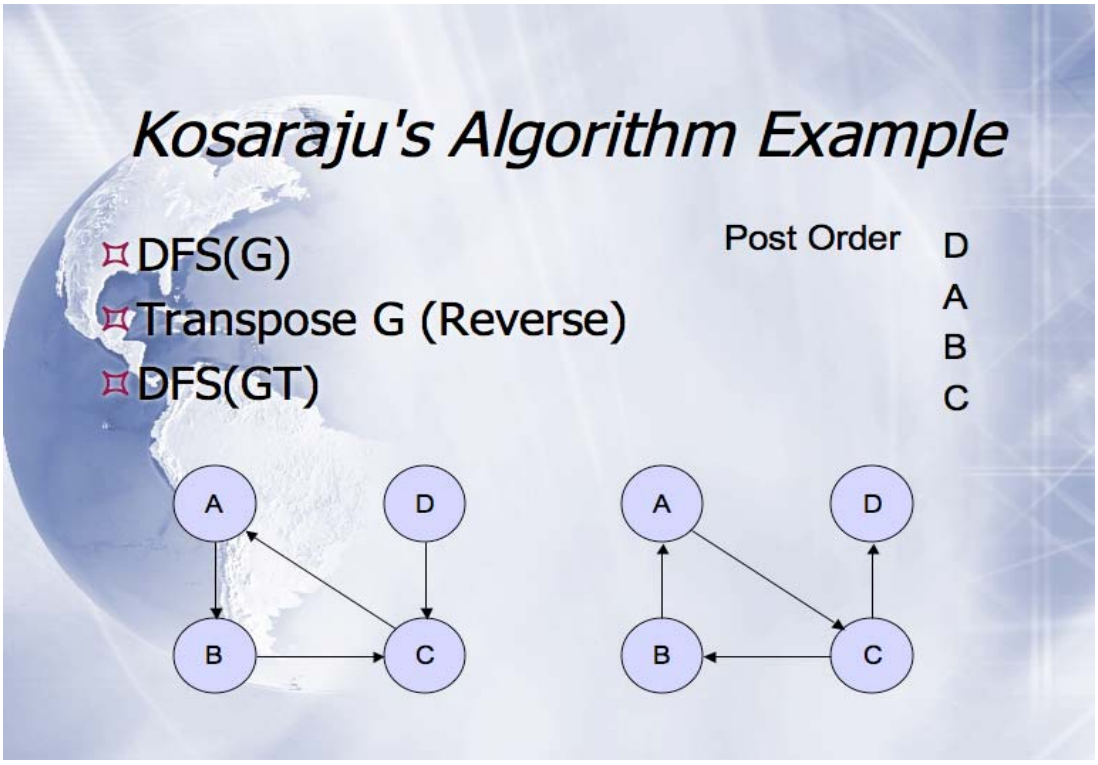
Identify Strongly Connected Components

In order to continue on a non-acyclic graph containing strongly connected components it is necessary to identify those components so they can be ranked separately from the acyclic parts of the graph. In order to do this we implemented Kosaraju's algorithm that works as follows:

Kosaraju's Algorithm

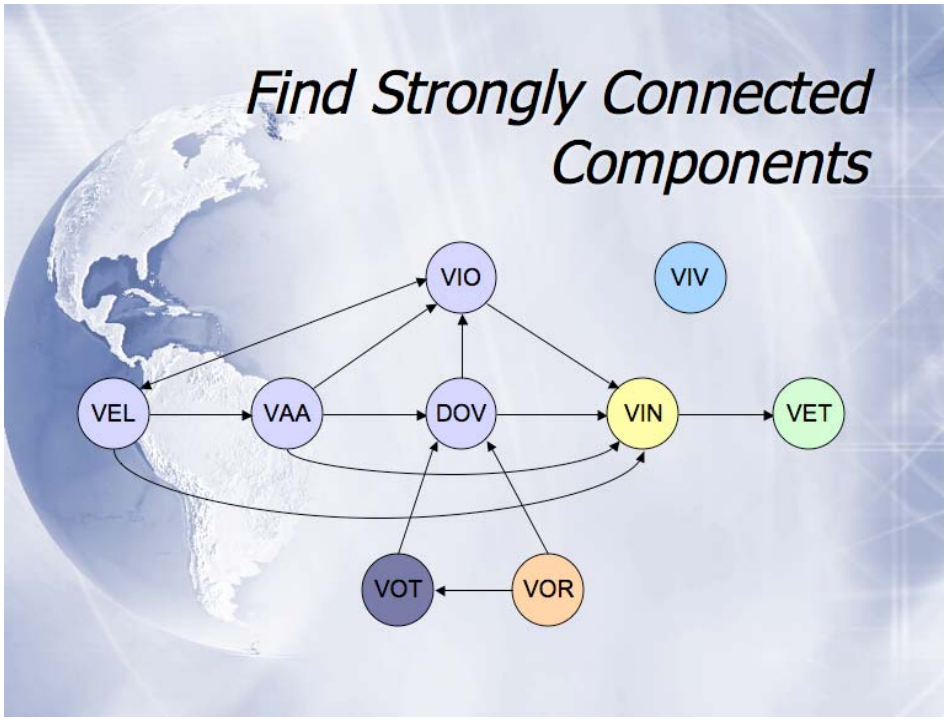
1. Perform a depth first search on the directed graph G – keeping a record of the post order traversal
2. Create a transposed graph of G called G^t
3. Perform a depth first search on G^t starting at the nodes in the post order you recorded from the first depth first search
4. Each tree produced in step 3 is a strongly connected component

To make this work, and for other parts of the program, I created an abstraction called a cage which holds individual baboons. The cage keeps track of, among other things, the ranks of the individuals in the cage, if known, and the number of individuals in the cage. However the connections amongst nodes, or amongst individuals in this case were stored in each individual within a list of parent and children nodes that represent people who have dominated or been dominated by that given baboon. Kosaraju's Algorithm will now be demonstrated with an example related to the graph on the left in following picture:



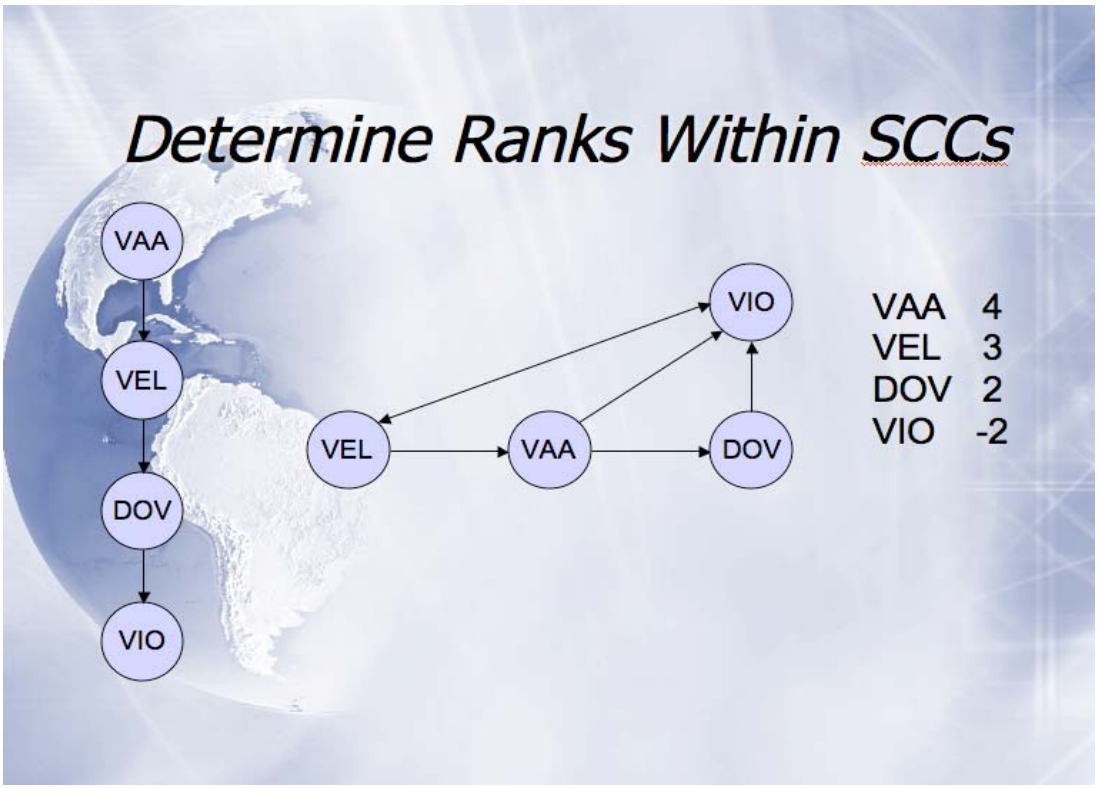
Depth first search, or DFS, will be run starting at the node with the highest matrix score, or in this case alphabetical order, and a post order stack will be kept as the recursion returns from the DFS on the original graph. DFS typically starts at a given node and traverses down the graph recursively until it reaches the deepest node it can and then returns back through the successive calls to the original node before moving on to starting on the next not-visited highest ranked node. After running this algorithm on the original graph, which visits A then B then C and then returns back to A as it adds to the post order before moving to D, the following post order is produced: D, A, B, C.

After obtaining the post ordering of the DFS on G, Kosaraju's algorithm then transposes the graph of G in order to create G^t. This is simply done by running through the individuals in the cage abstraction and asking each of them each to swap their collections of parents and children, effectively reversing the order of the graph shown in the previous figure on the right. Subsequently, the transposed graph has a modified DFS run on it during which starting nodes are selected in post order, each returned tree, which happens whenever a new starting node is picked, because there are no more unvisited nodes to explore, is now a strongly connected component. If Kosaraju's algorithm were run on our original and running example, coloring the members of each SCC differently, the following graph would be the output:



Determine Ranks Within SCCs

Now that we have each of the strongly connected components identified, we need to be able to rank them. Each of the singleton SCCs are easy to rank because one simply places the singleton individual as first rank. However, among strongly connected components with more than one node, we simply use the matrix score in order to determine what rank each individual should have within their strongly connected component. Therefore, continuing the running example, the SCC with four individuals would be ranked as shown in the following diagram:

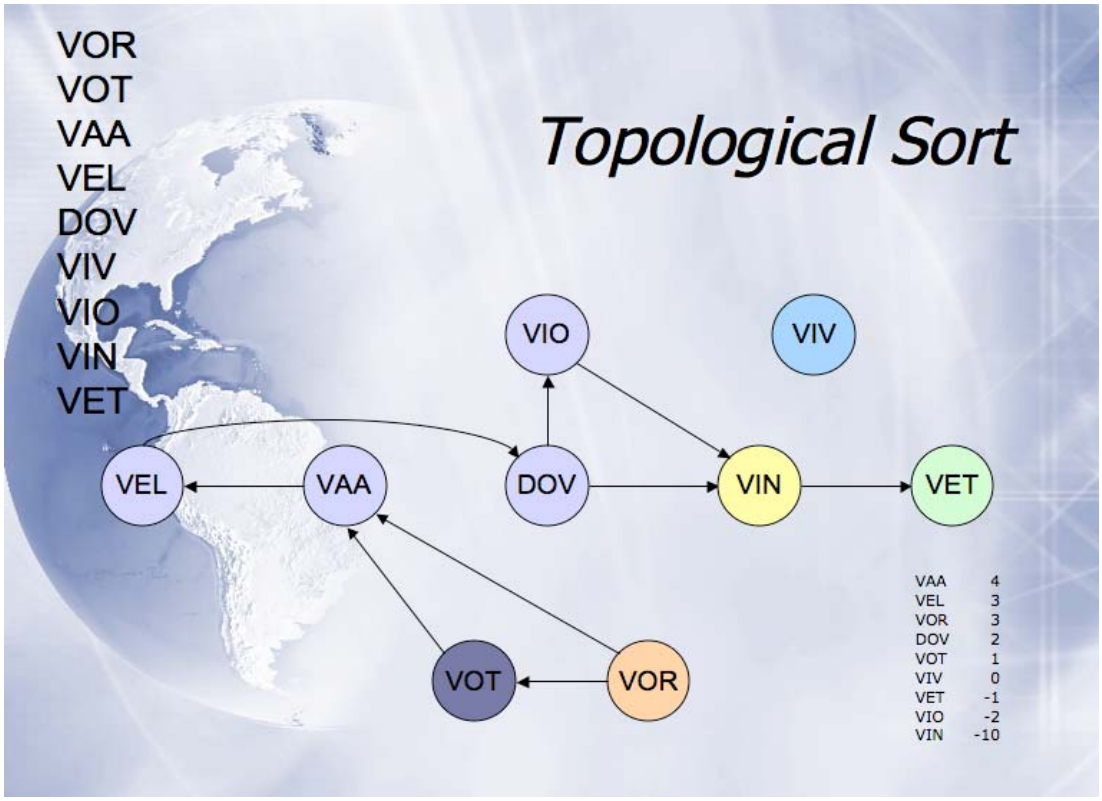


Here it should be clear that VAA has the highest matrix score of 4, VEL has the second highest with a score of 3, and so on, which is used to construct a directed acyclic graph of these components. This directed acyclic graph, or DAG, when added back into the larger graph of individuals that are either singleton SCCs or already ranked in a similar fashion will transform the original graph into a DAG which can be ranked with a topological sort.

However, care must be taken in replacing a SCC with and ranked SCC by removing all internal edges that are not part of the ones generated by the matrix score ranking and then connecting all of the original SCCs incoming edges to the highest ranked node and all the outgoing edges to the last ranked node. In doing this process carefully, we are able to maintain the structure and form of a DAG that it is now possible to rank.

Topological Sort

Topological sort can now be used on the newly formed directed acyclic graph in order to produce a ranking for all of the individuals. As seen in the figure below, with the final ranking visible in the top left, a node with no incoming edges is selected, ties again being broken using the matrix scores viewable on the bottom right, and then removed along with its connected edges, repeating the process until there are no nodes left in the graph. The order in which nodes are removed determines their rank; the first removed being the highest ranked individual in the cage as the following individuals assume subsequent ranks. This is the end of the Brock-Yang algorithm.



Conclusion

In summary, there now exists a completely new and powerful ranking tool written in Java that can be used to manually rank the baboons of the Amboseli project in an intuitive manner. Additionally, individual rankings may be computed automatically using the available interaction criteria in a reliable and sensible method using the Brock-Yang algorithm. While there may be discrepancies with what biologists know is the true ranking order based upon oversimplification of the rules and the absence of additional external information, given what data is available in the database to the ranker program, this is a very good suggestive indicator of what a possible optimal ranking may in fact be.

Bibliography

DeVries, H. (1997). Finding a dominance order most consistent with a linear hierarchy: a new procedure and review. Ethology & Socio-ecology group, Utrecht University.

Getz, G. W. a. W. M. (2006). A likely ranking interpolation for resolving dominance orders in systems with unknown relationships. University of California at Berkeley, Department of Environmental Science, Policy, and Management.