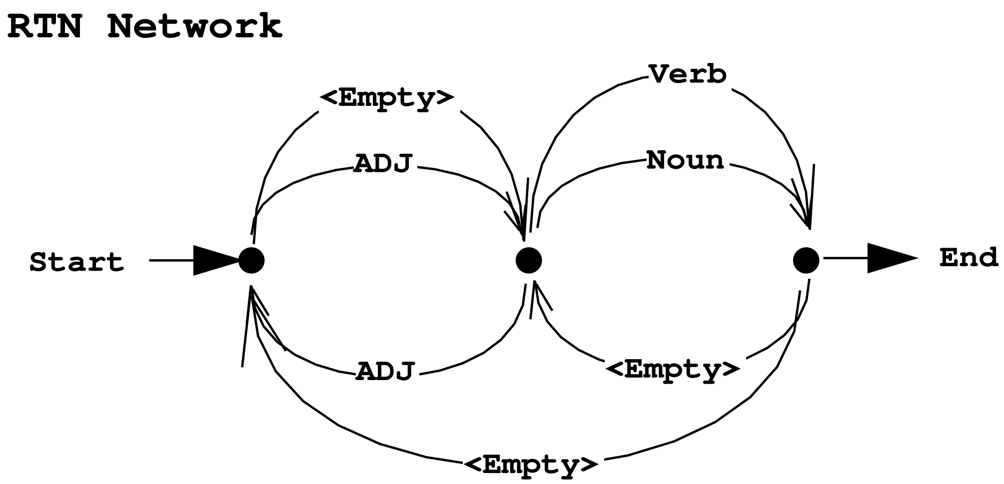
Using ATN (CF-Grammar) to express Design.

date: 6-10-1999
author: Joost Geurts

Automatic design is a tricky issue because of the large amount of rather difficult choices you have to make while generating one. These choices are not difficult in the sense that is hard to compute them but more in a way that there are many, in a way similar (one is not really better than the other) options you have to choose from. The choice you make can have effect on several other places in the generating process, and should be made considering earlier choices. For example when somewhere in the process you have to place a label next to an image. You have 4 options above-, below-, left- and right of the image. The question which-one to choose is a hard one because none of the four options is obvious better than the others. But when you choose for the option 'right' it turns out, the 4 images you wanted to be placed next to each other do not fit on the screen any more. To compensate this you can either place the label below the images or place one of the images below the others, or place some linking mechanism which takes you to the next screen. Choosing between these options is kind of similar to the choice you had before, whether to put the label above, below, right or left to the image. In short, making a certain choice can have a great effect on the design. When doing this automatically it means lots of backtracking on wrong choices and trying alternatives. There are however a lot of constraints involved in the automatic design process. Backtracking means going back to your last choice point and try the alternative (depth-first approach, breadth-first approach consumes to much memory in this case and is therefore no valid option). But trying an alternative on the lowest level is maybe not the thing you want (although it might turn out to be a solution). For example, placing the labels below the image will get you enough space to place the images next to each other. But what if you rather have one of the images below the others and keep the labels right to the images. The problem here is that you'll have to find a way to control the backtracking.
One of the solutions is to avoid backtracking. This way you provide several pre-made templates which reduces the complexity of the problem enormously because in this case you only have to choose between the different templates. However this solution has some disadvantages as well. One of them is that you can predict on forehand how the design is going to look like because there is no other one than the pre-made ones. This will fortify the notion of a pre-canned-presentation. Another one is that in this way you will have to provide templates which are aimed for a certain media-collection. In other words, you know what media items you get and the template tells you where to put them. You would rather just specify 'the look' you want independently of the format of your media items.
Another solution is to find a way to control backtracking. If we imagine the design specification as a graph where the nodes represent states (the design so far) and the edges represent choices. Finding a solution would be modelled as a search problem. A best-first approach would be a good option because in this way you have a mechanism based on a heuristic function to control your choices and thereby control your backtracking. The hard part however is to find a good heuristic. One way of doing this is by adding costs to the edges. Low cost for good alternatives and high costs for less good ones. The 'best' solution would be the lowest cost path trough the graph. Computing these costs wouldn't be a straight forward thing. When you have two subbranches A and B, where branch A is twice as deep as branch B. The total cost of branch A would most likely be greater than the costs of branch B because of the fact that A has more edges than B. Choosing option B because of that wouldn't be a fair choice. Second problem in choos-

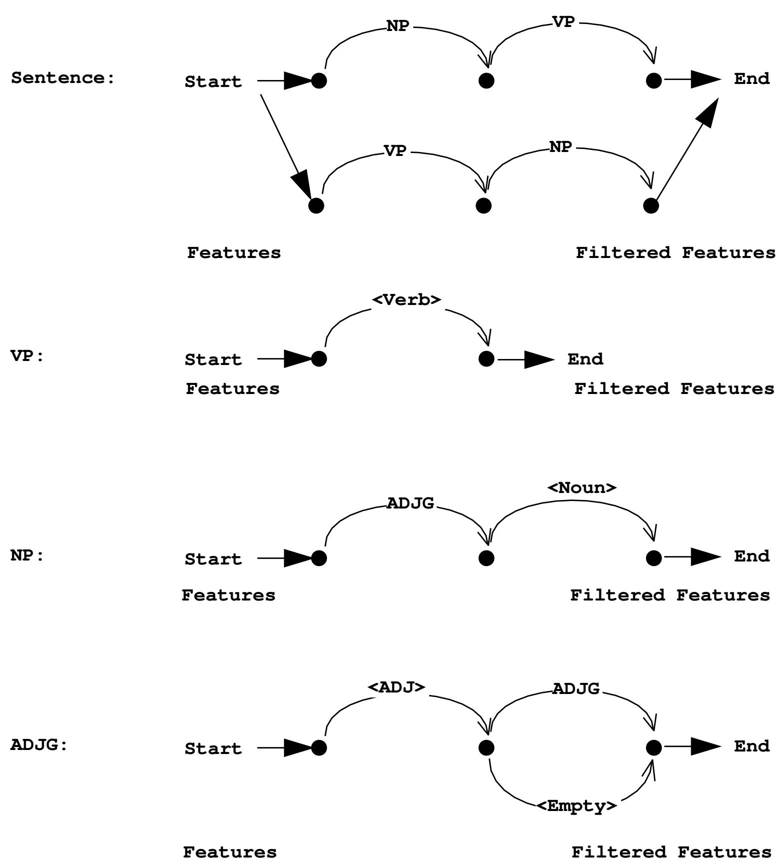
ing a best-first approach is that your design is implicit in the graph. When you want to modify your design you will have to adjust the costs for the edges, which is not a very nice interface.
Summarised there are two problems:
- Controlling backtracking
- Nice interface

Within natural language processing they have had similar problems. Because of the many categories a sentence can start with (NP ‘Pete walks’, VP ‘walks pete’, ADJ ‘lazy Pete walks’, etc.) they needed a structure which enabled them to process these in a structured way. One of the first attempts was what was called a Recursive Transition Network (RTN)



This network could accept sentences like the following
Pete walks (starting with a noun)
Lazy Pete walks (starting with an adjective)
Big lazy Pete walks (starting with an adjective)
Walks Pete (starting with a verb)
Walks lazy Pete (starting with a verb)
It provided a way to control backtracking although is was far from ideal, because it was very hard to let the network only accept correct sentences. For example just ‘Pete’ would be accepted in the above example. Besides this, it was very difficult to ‘control’ the network because of the unforeseen effects is has by adding an edge to a complex network. Therefore some alternatives where suggested. One of them was a ATN (Augmented Transition Network) network. This Network consisted of several sub networks. High-level networks contain references to the sub-level networks. So, when in a high-level network a reference to a low-level networks occurs, the sub-network is loaded and executed. When the sub-networks ends correct, execution is resumed at the high-level network. This way backtracking becomes again more controlled. The ATN networks also provided a solution to the problem which occurred due to Context Free Grammars. Which is that you can’t express consistency in CF grammars. If the noun is plural then the verb should be plural as well. The way this was done was be adding a feature lists, containing tuples like ‘time:past’, ‘number:plural’ etc. to the network. The high-level network passes a feature-list to the sub-network. The sub-network can decide on basis of the feature-values whether the input is correct and can, if desired add a new value to the feature-list which is passed back to the high-level network.

ATN Network



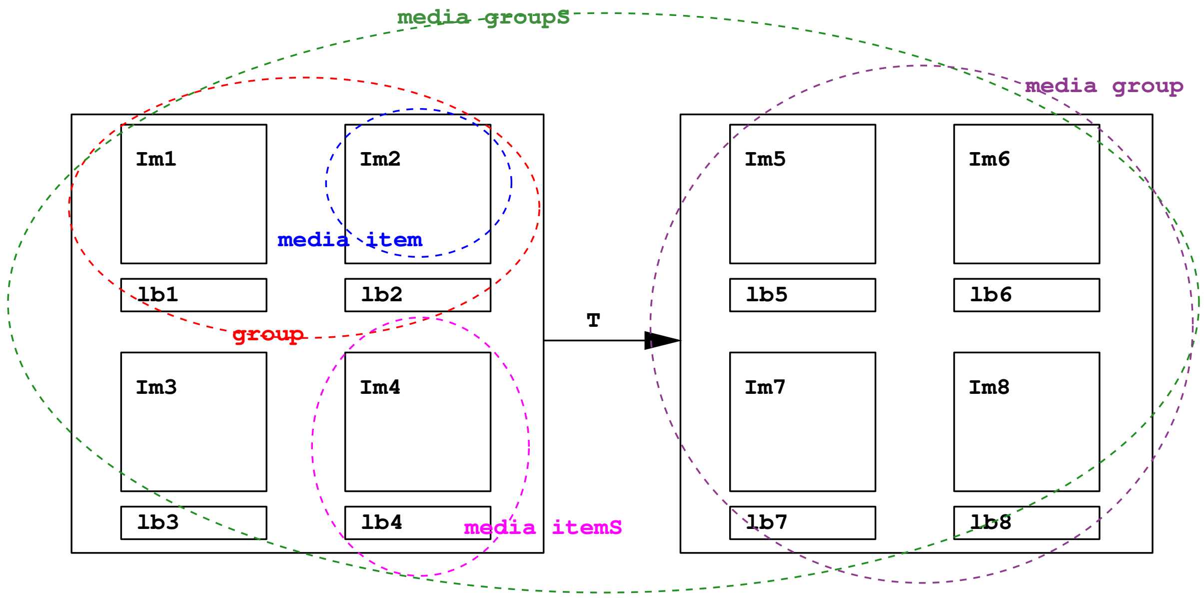
The problems in natural language processing which are: Controlled backtracking, and providing consistency seem to be applicable in a way to automatic design as well. If we are able to express a design in terms of a context free grammar and control choices by the use of feature-values the problems of controlling backtracking, being consistent and having a nice interface could be solved.
The design-grammar could look something like:
(note: The grammar is not complete and probably not correct but it just meant to get an idea.)

Table 1:

Rhetorics	SequenceSlices SlideSlices CircularSlices	there is an order and start no order there is an order but no start
SequenceSlices	Slice ‘after’ SequenceSlices	
SlideSlices	Slice SequenceSlices	
Slice	MediaGroups	A slice represents a subject

Table 1:

MediaGroups (MGroup)	MGroup ‘leftof’ MGroups MGroup ‘rightof’ MGroups MGroup ‘above’ MGroups MGroup ‘below’ MGroups MGroup ‘linkto’ MGroups MGroup ‘after’ MGroups	
MediaGroup	Groups	A MediaRroup represents a group of media-items which have a relation.
Groups	Group Group Groups	
Group		A group is a set of media items where there is one main item the others are dependant of the existence of the main. e.g. a caption with an image
MediaItems	MediaItem MediaItem ‘leftof’ MediaItems MediaItem ‘...’ MediaItems	MediaItem is the main item leftof, rightof,below, above, linksto, after
MediaItem	Image Video Audio Text	



This grammar specifies all the possibilities a design can have. The feature-values(constraints) determine/control how the final design is generated. Feature-values/constraints come in two flavours, one is at the beginning of a generation. Here you specify user-preferences, and design-dependant preferences such as ‘no audio’ if the user has no sound hardware and ‘prefer image above text’ if your design requires a fancy look. The second way of using feature-values/constraints would be run-time. Once you have selected a video there shouldn’t play another video at the same time. To achieve this you could add a feature-value ‘no-video’. To achieve consistency the feature value could be used as well. Once you decided to put a label below an image, you could add an feature value which states to prefer to put labels below images in the future.

There are still a lot of questions to be answered, for example the question whether is should be a parsing process on a collection of grouped media-items, or a generating process according to a certain rethoric.
Anyway I hope the general idea is clear enough. Comments are most welcome.

Joost