Paper SM03

DOT Language for CONSORT Flow Diagram

Nicolas Guerro, Idorsia Pharmaceuticals Ltd., Allschwil, Switzerland

ABSTRACT

SAS® makes it possible to manage a wide variety of graphics using SG procedures, annotations and templates... Nevertheless, it is often seen that some graphics require many tweaks and pre-processing for the program to remain readable, data independent and reproducible. This is the case with flow diagrams which have been popularized by the CONSORT (CONsolidated Standards Of Reporting Trials) group to represent the interactions between populations in a structured way. Several solutions are available in existing papers, however the one that will be discussed here relies on the graph description language, DOT, to visually describe the relationship of objects before demonstrating how R is used to execute it and SAS to manipulate data, retrieve and refine the output. The goal of this paper is first to review existing methods for creating flowcharts and then to establish a neat and efficient process by opening the audience to the DOT Language that could further add value in many other applications.

INTRODUCTION

The CONSORT flow diagram is an increasingly widespread diagram and yet its realization is not so obvious. There are methods or tools sophisticated enough to "draw it by hand" but the interactivity with the data is broken. Other programming-oriented approaches make it possible to overcome the static side but quickly turn out to be a headache.

After recalling what a CONSORT flow diagram is and defining the user/technical needs and requirements, we will recall several implementation technics that have already been exposed in the past before detailing an implementation strategy based on 3 technologies (Figure1). The purpose of this strategy is to be able to generate a flow diagram regardless of the design of the study and to break free from limitations other mechanisms face.

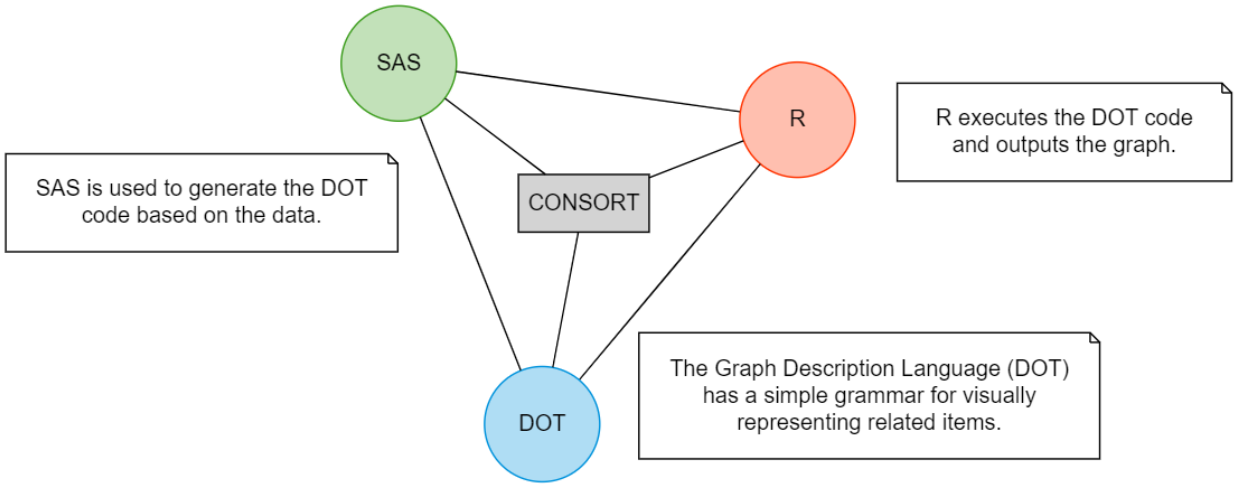


Figure 1: CONSORT flow diagram strategy

CONSORT FLOW DIAGRAM

The CONSORT flow diagram is a recommended type of graph for reporting the progress through a trial analysis [1], especially at the time a reporting is submitted in journals or papers. It takes up most of the results that can be found in the Clinical Study Report tables.
For each stage of a trial, it describes the number of participants by group as well as losses together with reasons for exclusion.

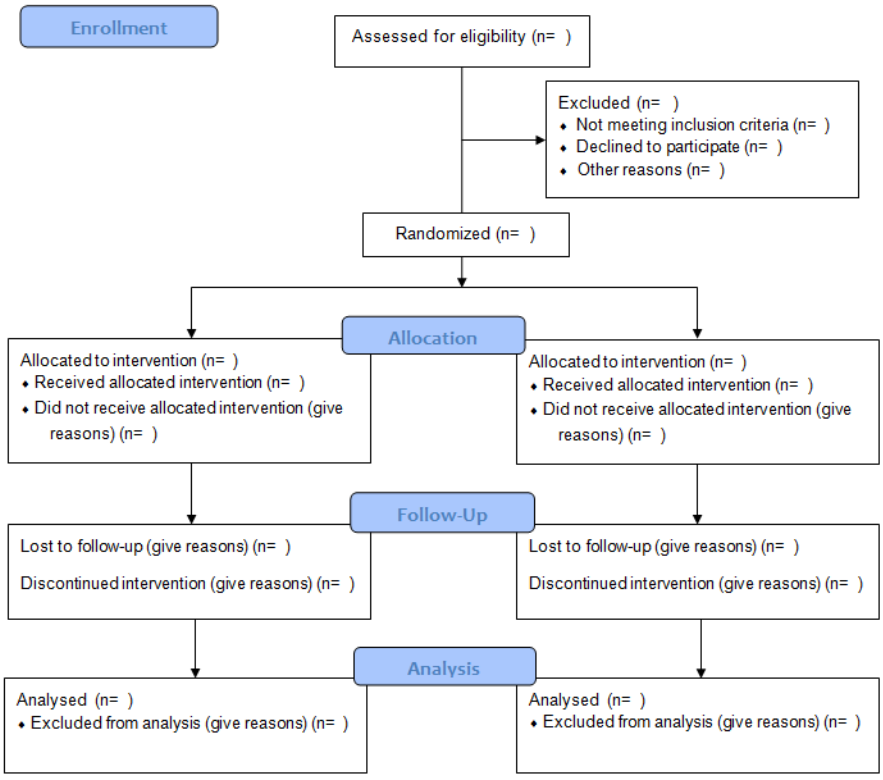


Figure 2: Flow diagram of a parallel randomized trial of two groups [2]

USER REQUIREMENTS

Let us zoom in on the previous graph (Figure 2) to analyze it, name its elements and define the user needs as well as the technical constraints.

Graph	▪ The reading of the diagram is directional, i.e., it is from top to bottom.
Node	▪ Each cell (or box) is named “node”. They are aligned and distributed proportionally along the margins. The top node is the root node.
Edge	▪ The link connecting one node to another is called edge. ▪ The port (head and tail) is located in the middle of a node.
Parent	▪ A node that precedes another will be called "parent". Multiple nodes can have the same parent. They are called sibling nodes.
Level	▪ The level of a node is the number of edges along the unique path between it and the root node.
Outer	▪ As opposed to a “parent”, a node can be a “child”. However, only the external child nodes (for instance, those indicating a discontinuation from the trial) are to be identified because the reading direction is from left to right in the context of the CONSORT plot.

Cluster

- Several nodes can be grouped together to form a subgraph or a cluster.

Rank

- In a top to bottom directed graph, a rank defines several nodes that must be horizontally aligned.

Text

- The content (text) of a node can be composed as follows:
  - A free text
  - A built text associated with a population variable combining the label of the population variable and the calculated value corresponding to the sum of patients fulfilling the population criterion (N=XXX).
  - The values of a variable (e.g., a treatment arm)
  - The items of a codelist including or not the categories where no patient meet the criterion (n=0). These can be presented in columns or rows, accompanied by a title.

Attributes

- Attributes of interest for nodes and edges are (among others): color, border and fill, length, width, distance, font, alignment…

SVG

- The SVG extension for Scalable Vector Graphics is a vector drawing format. It is not strictly speaking an image like PNG or JPEG files (also called raster image), but a text file describing each element of the graph, diagram, icon, map... by its vector coordinates. It should be noted that SVG files are recommended because they allow graphics to be resized (scalable) without deteriorating the image and they can be reworked. The format works on the principle of XML.

PITFALLS

Unless the creation of the diagram responds to a one-time request, the problems that the diagram author may face are similar to those that a programmer encounters when they decide to switch from a program to a macro.

If the data change, it must be possible to do this automatically in the graph, whether it concerns population sizes "(n=…)", codelist elements, variable values, modified texts, etc... The general layout of the graphic on the page will depend on the number of nodes to be positioned vertically or horizontally, but also on the text to be included in each node, which impacts the node's dimensions. The attributes of the different elements must be easily customizable. It is also essential to think about reusing the graph on another study and therefore, to anticipate design changes (periods, treatment arms, etc...). When we talk about "reuse", we must also anticipate reuse by someone else. In this situation, solid documentation and good readability of any code is crucial. Finally, from experience, the programmer keeps in mind that the specifications fluctuate with time or people, and very often this happens near the delivery.

Therefore, the solution used must not only take account of the requirements but also be apprehended in a more global manner.

SOLUTIONS

First, let us focus on some tools for producing manual CONSORT diagrams, then look at ways previously covered in PHUSE papers to achieve flow diagrams with programs.

MICROSOFT WORD AND MICROSOFT POWERPOINT

The interest of these 2 office applications is that everyone knows them, masters them at a reasonable level and that they are available almost everywhere. Consequently, a person without any programming knowledge can support the creation/modification of a CONSORT diagram.
On the other hand, this method is prone to errors and time consuming. Because the information displayed is disconnected from the source data, it does not update when refreshing the database. Moreover, as any programmer knows, the famous "copy/paste" must be avoided as much as possible. In the case of a somewhat complex study design, the main difficulty lies in the positioning and adjustment of the elements (nodes, edges, texts) and therefore a simple modification of the appearance can lead to big changes. Finally, this method does not allow to quickly reproduce a diagram for another study having a different design.

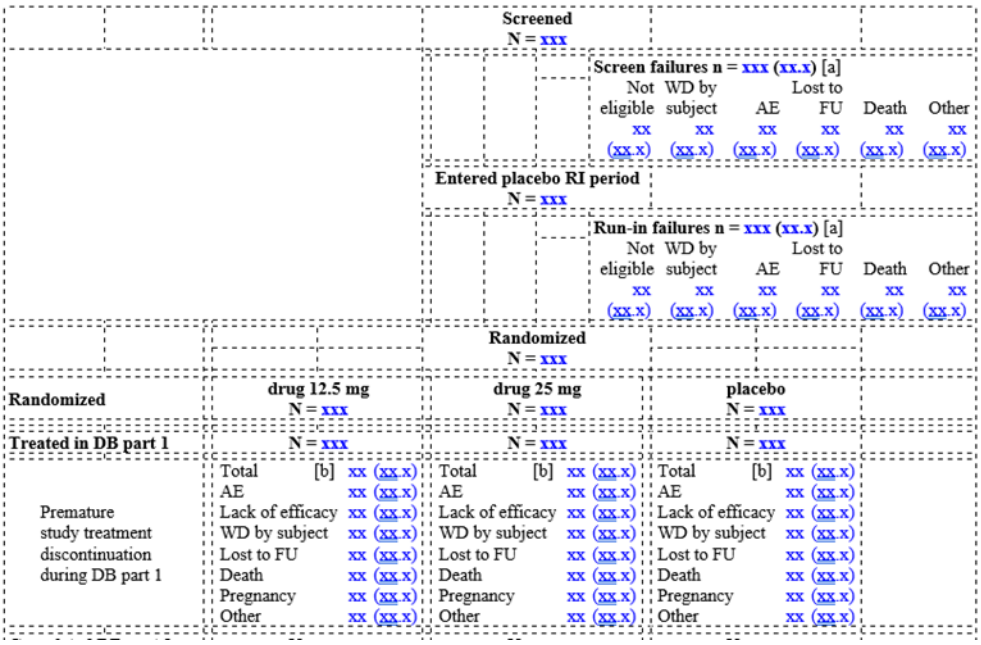


Figure 3: Diagram built with Microsoft Word

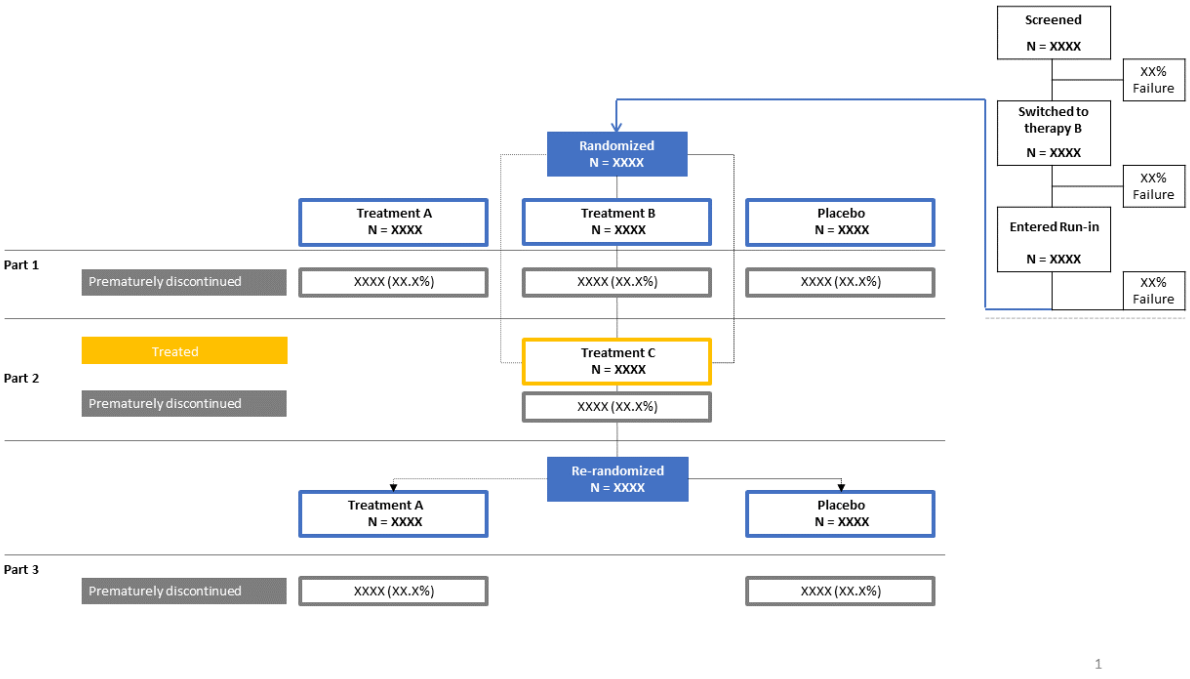


Figure 4: Diagram built with Microsoft PowerPoint

MICROSOFT VISIO

This is the main application for creating any type of diagram like flowcharts, network diagrams, Venn diagrams, floor plans and much more. With an age of more than 20 years, it is well established. The interface is structured in the same way as other Office applications (e.g., a ribbon with FILE, HOME, INSERT... tabs) which allows a very quick start for beginners. It is a collaborative tool and diagrams can be included in other Office files. With an add-in, it is also possible to create data-driven diagrams from Excel tables.

DIAGRAMS.NET™ / DRAW.IO ®

This is the other “main” application if you do not have Visio or if you are using Visio in Microsoft 365.

diagrams.net/draw.io is an open-source technology stack for building diagramming applications, and the world’s most widely used browser-based end-user diagramming software [3].

That being said, this tool is equivalent in terms of GUI and capabilities to Visio.

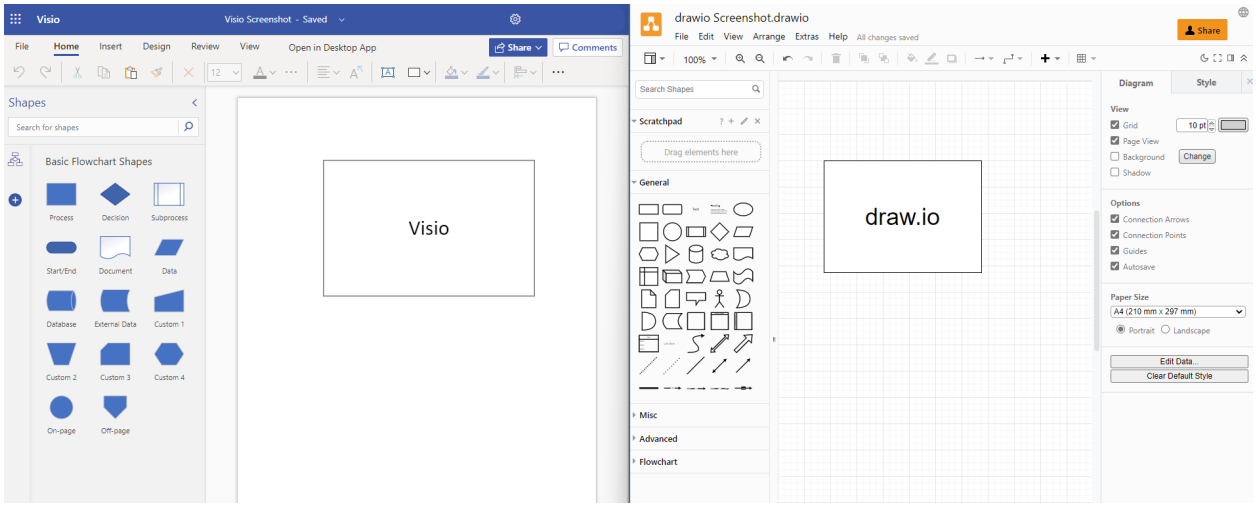


Figure 5: Visio and draw.io interfaces

UPDATING RTF FILES WITH SAS

In 2012, Carpenter and Fisher [4] suggested to use an RTF template “prepared for use by SAS by filling in each of the individual fields using unique codes”. The RTF file has sections common from document to document and placeholders (tokens) where a text has to be substituted, typically a count. A SAS macro reads the RTF file, compute the text to insert and performs the replacement before sending it back to the RTF file.

In 2017, Hinson [5] went a step further by showing how to execute a SAS macro whose name would be typed in the RTF template thanks to the `DOSUBL` function and `Stream` procedure. An example of its usage is given here.

SGPLOT DRAW STATEMENTS / ANNOTATE DATASET

With SAS/GRAPH, the annotate facility enables to set up a special “annotate dataset” which serves to add graphical elements such as boxes, arrows, texts and so on. This functionality is still available with SG procedures but the draw statements from the PROC TEMPLATE do the same.
As there is no plot statement to create a flow chart in the SG procedures, the idea is to annotate/draw each element and many publications describe how this can be achieved.

- For old-school programmers familiar with %ANNOMAC (%SGANNO), a flowchart macro [6] using annotate has been submitted in 2008, before CONSORT flow diagrams were launched.
- In the same vein, another approach [7] suggested to split up individual elements of the diagram and import, arrange and link them in Excel.
- A more modern programming technic based on SG procedures was introduced in 2016 by Sanjay Matange on Graphically speaking [8] and bit by bit improved [9] [10] [11]. Some papers proposed solutions to fully automate the creation of the Diagram [12].

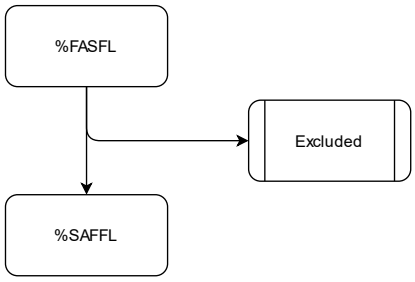
APPRAISAL

To conclude this journey through the different methods for creating a CONSORT flow diagram, Microsoft VISIO and DIAGRAMS.NET™ are positioned as the most efficient and fastest tools for creating it or modifying some of its existing elements.
From the moment when we must consider a routine creation approach of these diagrams, a switch to a technique involving a programming language (SAS) is necessary because of its connection to the data and also because it remains GCP compliant.
However, if the user wants to retain the ability to be flexible with respect to the study designs, the relationship between the programming complexity and the time it takes on the one hand and the frequency of the need for these diagrams on the other hand may not be judicious.

GOING FURTHER

At this level, wouldn't there be a method that would combine the advantages of some points mentioned above to create a flow diagram?

1. Select a tool to manually draw the diagram and save the file as SVG, say DRAW.IO.



2. When opening the file with a text editor, XML like code is showing up, from where we can easily identify tokens to replace.

```
<text x="59.5" y="34.5">%FASFL</text>
```

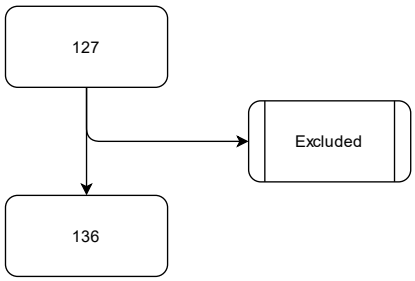
3. Build a SAS macro which name matches the token and uses the DOSUBL function to return text.

```
%macro fasfl;
%global cnt1;
%let rc = %sysfunc(dosubl('
    proc sql noprint;
      select count (*) into:cnt1 trimmed
      from adam.adsl
    where fasfl eq "Y";
    quit;
'));
&cnt1.
%mend;
```

4 With PROC STREAM, reads in the SVG file and performs text replacement

```
filename svg_in "&path./&image..svg" lrecl = 32755;
filename svg_out "&path./&image._out.svg" lrecl = 32755;
proc stream outfile=svg_out RESETDELIM="GOTO" quoting=DOUBLE;
BEGIN
GOTO; %include svg_in;
;;;;
```

5. The new SVG is updated with values.



6. Unsatisfied with the rendering? The SVG can be manually edited with Scalable Vector Graphics Editors e.g., Inkscape©. With such tools, fine tuning graphics, like adding, (re)moving, filling and so on is made easy. It is also possible to export the graph as a PNG file and it then make it possible to insert the image in a PDF document.

DOT LANGUAGE

The DOT Language is a syntax for describing graphs. In mathematics, graphs are the representation of a set of objects of which certain pairs are related, hierarchical and directional (but not always). This syntax is then parsed by a program to render the graph. Getting started with the language is simple, fast and learning the essential features can be covered in a very short time.
Graphviz [13] is such an open-source program which comes with all the documentation. DOT Language viewers [14] [15] allow to write and instantly visualize the graph.

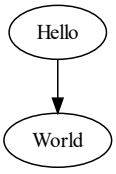
BASICS

The following examples serve to introduce the DOT grammar and can be tested in a viewer.

Hello World

```
/* Example 1 */
digraph PHUSE {
    rankdir="TB"
    Hello -> World
}
```

- Use `/* comment */` or `// comment` to insert a comment.
- `digraph` declares a directional graph, named `PHUSE`.
- `rankdir` orients the graph from Top to Bottom.
- An edge is created between 2 nodes, A and B.
- A and B are the node identifiers and, by default, their label.

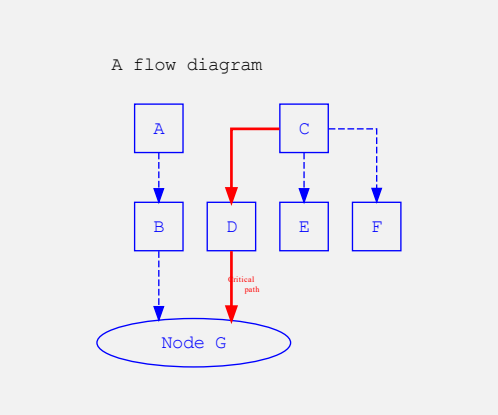


Defining elements attributes and linking them

```
digraph PHUSE {
       rankdir   = "TB";                    label       = "A flow diagram\n ";
       labelloc  = t;                       labeljust   = l;
       fontname  = "Courier New";           splines     = ortho;
       bgcolor   = grey95;                  pad         = "1,0.5";

 node [shape     = box                      color       = blue
```

```
       width     = 0.5                          fontcolor    = blue
       fontname  = "Courier New"]

 G [   shape     = oval                         width        = 2
       label     = "Node G"]

 {rank=same A C}
 {rank=same B D E F}

 edge [color     = blue                         style        = dashed]
   A -> B -> G
   C -> {E F}

 edge [color     = red                          penwidth     = 2
       style     = filled]
   C -> D
   D -> G
      [label     = "      Critical\npath\r"     fontname     = "Arial"
       fontcolor = "red"                        fontsize     = 6]
}
```

- Global attributes for the graph are declared at the beginning.
- Similarly, global attributes can be assigned to nodes or edges: `node [attributes]` or `edge [attributes]`.
- If a specific style has to be applied, declare the node and list its attributes different from the default ones: *`node_id`* `[attributes]` and the same is done for edges: `node_id -> node_id [attributes]`.
- Forcing nodes to be aligned is a common constraint that is achieved with `rank` .
- There are different ways to specify edges and `node_id1 -> node_id2; nodeid1 -> node_id3;` is equivalent to `node_id1 -> {node_id2 node_id3};`
- Labels can be wrapped, aligned, padded using escape sequences or additional white spaces



Flow diagram

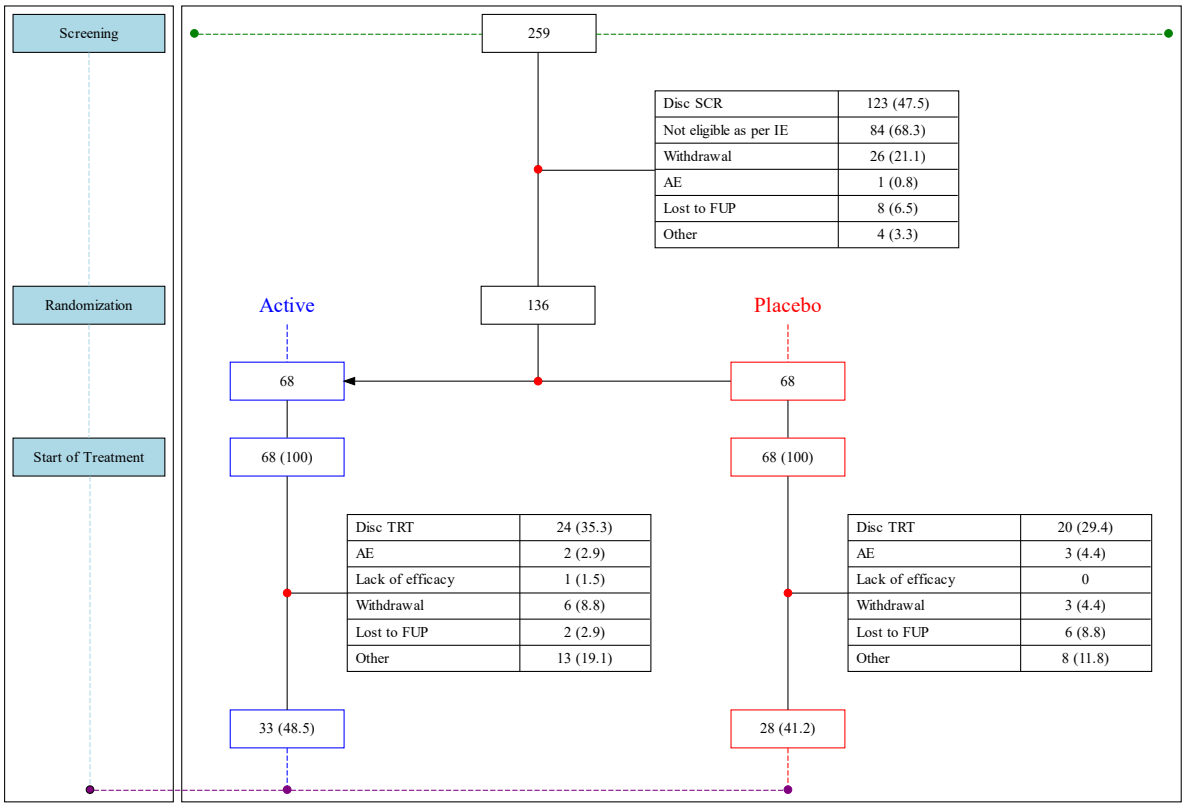
```
digraph CONSORT{

 splines = ortho;
 newrank = true;
 pad     = 1;

 subgraph cluster_periods{
  node [shape=box style=filled fillcolor="lightblue" width=2];
   period1 [label="Screening"];
   period2 [label="Randomization"];
   period3 [label="Start of Treatment"]

  node [shape=point width=2 height=0.1 label="" fillcolor=purple]
```

```
   invis0;

  edge [style=dashed arrowhead="none" color="lightblue"]
   period1 -> period2 -> period3 -> invis0;
 }

 subgraph cluster_plot{
  node [shape=box width=1.5];
   scr_all  [label="259"];
   rnd_all  [label="136"];

  node [color="blue"];
   rnd_act  [label="68" ]
   trt_act  [label="68 (100)"]
   eos_act  [label="33 (48.5)"]

  node [color="red"];
   rnd_pbo  [label="68"]
   trt_pbo  [label="68 (100)"]
   eos_pbo  [label="28 (41.2)"]

  node [shape=plaintext fontsize=20]
   lbl_act  [label="Active" fontcolor="blue"]
   lbl_pbo  [label="Placebo" fontcolor="red"]

  node [label="" shape=point width=.1 height=.1];
   invis1;  invis2;  invis3;  invis4;
   invis5 [color=green]; invis6 [color=green]
   invis7 [color=purple]; invis8 [color=purple];

  node [shape=record labeljust=l fontsize=14 width=4 color=black]
   tbl1[label="{Disc SCR\l|Not eligible as per IE\l|Withdrawal\l|AE\l|Lost to
FUP\l|Other\l}|{123 (47.5)|84 (68.3)|26 (21.1)|1 (0.8)|8 (6.5)|4 (3.3)}"]
   tbl2[label="{Disc TRT\l|AE\l|Lack of efficacy\l|Withdrawal\l|Lost to
FUP\l|Other\l}|{24 (35.3)|2 (2.9)|1 (1.5)|6 (8.8)|2 (2.9)|13 (19.1)}"]
   tbl3[label="{Disc TRT\l|AE\l|Lack of efficacy\l|Withdrawal\l|Lost to
FUP\l|Other\l}|{20 (29.4)|3 (4.4)|0|3 (4.4)|6 (8.8)|8 (11.8)}"]

  {rank=same invis1 tbl1};
  {rank=same invis3 tbl2};
  {rank=same invis4 tbl3};
  {rank=same rnd_act invis2 rnd_pbo};
  {rank=same trt_act trt_pbo};
  {rank=same invis5 scr_all invis6};
  {rank=same eos_act eos_pbo};

  edge [arrowhead="none"]
   scr_all -> invis1;
   rnd_all -> invis2;
   trt_act -> invis3;
   trt_pbo -> invis4;

  edge [style=dashed arrowhead="none"]
   lbl_act -> rnd_act [color=blue];
   lbl_pbo -> rnd_pbo [color=red];
   invis5  -> scr_all [minlen=15 color=green]
   scr_all -> invis6  [minlen=30 color=green];
   eos_act -> invis7  [color=blue];
   eos_pbo -> invis8  [color=red];
```

```
  edge [style=solid]
   invis1  -> rnd_all;
   invis3  -> eos_act;
   invis4  -> eos_pbo;
   rnd_act -> trt_act;
   rnd_pbo -> trt_pbo;
   invis2  -> rnd_pbo [minlen=10];
   rnd_act -> invis2  [dir=back minlen=10];
   invis1  -> tbl1    [minlen=6];
   invis3  -> tbl2    [minlen=3];
   invis4  -> tbl3    [minlen=3];
 }

 {rank=same period1 scr_all};
 {rank=same period2 rnd_all};
 {rank=same period3 trt_act trt_pbo };
 {rank=same invis0 invis7 invis8};
 invis0 -> invis7 [minlen=10 style=dashed arrowhead="none" color=purple]
 invis7 -> invis8 [minlen=26 style=dashed arrowhead="none" color=purple]
}
```

- Why dashed lines and filled circles?
  Using invisible edges and nodes allow to get everything in the desired position. In order to better understand how the grid should be setup, this example uses dashed lines and filled circles:
  - The top left and top right green points fix the width of the cluster.
  - Red circles create branches and edges headed to it have no arrow.
  - The purple system is used to stretch clusters down.
  - Light blue edges define the sequence of periods.

  Replacing the style of all the edges with `invis` and setting width/height of nodes named “invis” to 0 will make them transparent.
- Tables: The `record` shape is a suitable type of node for tables where columns and rows are controlled with braces and vertical bars.



DIAGRAMMER

`DiagrammeR` is an R Package which offers a large range of options for creating and viewing graphs.
grViz is a function of DiagrammeR which can make diagrams using viz.js, and so it works the same way as DOT Language viewers.
Once all necessary libraries are loaded, the R code is:

```
grViz("
  <DOT Language code>
")
```

DIAGRAMMERRSVG / RSVG

In the viewer pane, the rendered graph is an SVG file. It can be saved as a PNG, JPEG or BMP file. To automate this step, both `DiagrammeRsvg` and `rsvg` packages are loaded and the R code is:

```
mygraph <- "<DOT Language code>"
grViz(mygraph) %>%
    export_svg %>%
    charToRaw %>%
    rsvg_png("<path to png file>",width = xxx, height = xxx)
```

R IN SAS

For certain reasons, you might want to have SAS programs only to produce your deliverables. SAS/IML Software has a mechanism to call R programs and transfer data between R and SAS. Many papers discussing the topic can be referred to for settings (RLANG, R_HOME).

```
proc options option=RLANG;
run;
options set=R_HOME "C:\Program Files\R\R-4.1.1";
proc iml;
  submit / R;
  # R code starts here
    <R code>
  # R code ends here
  endsubmit;
quit;
```

Previously, it was showed how to save a graph as a raster image file in R.
The ODS style PREIMAGE is used to embed the raster image file in a PDF document which has titles, footnotes, template styles defined consistently with other figures. It can be placed in a SASITEM with Proc Document for later reuse.

```
ods escapechar = '^';
data get_raster;
    length text $500;
    content=1;
    text ='^S={preimage="<path to png file>"}';
run;

ods document name=work.dot (write);
    proc report data= get_raster nowd missing spacing=1 split="@" contents=""
                      style(report)={rules=none frame=void cellspacing=0  };
        column content text ;
        define content / order order=internal noprint;
        define text    / style(column)={asis=on } " " width=20;
        break before content / page contents='';
    run;
```

```
ods document close;
ods pdf file ="<path to pdf file>";
    proc document name=work.dot(read);
        replay / dest=(pdf);
    run; quit;
ods pdf close;
```

METADATA DRIVEN GRAPH

One can imagine conceptualizing the creation of a DOT plot through a dataset describing its elements without having to tweak with the layout of these elements thanks to the DOT Language, unlike the techniques for creating the CONSORT diagram that already exist.
The table below can be used as a starting point but in no way represents the method to follow.

Variable	Description	Example	
Node	An element (node) identifier. The ID would be constructed with a node shape, a style identifier, an increment.	Box_style1_1	Shape=box, style1 is default style (kind of printer style), 1 represents the first node.
		Tbl_style1_1	First table (shape=record) with same style as above.
		Title_style2_1	A specific box shape used to label a block and a different style, e.g., with a fill color.
Cluster	A name used to define the cluster the node may belong to.	Randomization	The current node is assigned to the Randomization node
Parent	If a node has a parent node, it refers the parent node identifier	Box_style1_1	If current node being defined is Box_style1_2, then it is linked to Box_style1_1, down to it.
Out	This flag is used to determine whether the node is used to represent a node edged to the right.	Y	For instance, a node describing the reasons for exclusion.
Rank	If the current node needs to be aligned with another one previously defined	Title_style2_1	If current node being defined is Box_style1_1, then it forces it to be aligned with Title_style2_1.
Text	A text convention which would allow to write a free text, a count, values of a variable, insert line breaks	“Screened”	Simply writes “Screened” in the node.
		“Screened and#randomized”	Split the text on 2 lines with a split character
		“Randomized (N=%RANDFL)”	Run %RANDFL macro to get number of randomized patients.
		“Randomized (n=%RANDFL(npct))”	Run the same as above and computes percentages based on the Parent node.
Where	A where clause to select records where a count has to be computed.	ADSL.RANDFL EQ 'Y'	This is not necessary if the %RANDFL macro is used.
Codelist	In the case the values of a codelist must be printed.	ADSL.DCSRS	If only a variable name is specified, prints only values of the codelist actually in the dataset
		ADSL.DCSRS#CL.NCOMPLT	Same as above except that it retrieves all values of the codelist, i.e., may result to n=0.
Options	Any text to be added when an edge is created (length, style, anchor…)	[minlen=6]	If current node being defined is Box_style1_2 and its parent is Box_style1_1, the edge length is forced to 6.

CONCLUSION

There is a wide range of solutions for producing CONSORT diagrams and, more generally, flow diagrams. As very often in computer science, the tool or method to adopt depends on what you want to do and how you want, or can, do it. For occasional use, outside the field of programming, Visio and DRAW.IO are remarkable and very flexible tools. As soon as you want to be part of a more « GxP » environment (traceability and accountability), adopting the DOT Language solves the tedious side of calculating the coordinates of each element of the graph. Beyond a fairly restricted grammar, it has, like any language, its little traps, tips and tricks to know, but it is quickly acquired. DOT Language editors also make it possible to prepare the graph and the corresponding code in a reactive way, which is very pleasant. Secondly, being able to run DOT Language from R or even being able to run R code from SAS opens the way to producing flow diagrams more in line with the usual methods and closer to the data. Note that the use of SAS is not necessary at this stage if you are comfortable enough with R programming.

Automatic creation of flow diagrams can also be used to represent a data flow or a programs hierarchy diagram (See PROC SCAPROC),

REFERENCES

1 Consort - Welcome to the CONSORT Website (consort-statement.org)
2 CONSORT 2010 Flow Diagram.doc (live.com)
3 Diagram Software and Flowchart Maker
4 Reading and Writing RTF Documents as Data: Automatic Completion of CONSORT Flow Diagrams (pharmasug.org)
5 Making Documents 'Intelligent' with Embedded Macro Calls, DOSUBL and Proc STREAM: An example with the CONSORT Flow Diagram (lexjansen.com)
6 074-2008: Creating Flowcharts Using the Annotate Facility (sas.com)
7 Alternative Approaches to Creating Disposition Flow Diagrams (pharmasug.org)
8 Outside-the-box: CONSORT diagram - Graphically Speaking (sas.com)
9 CONSORT Diagrams with SG Procedures (pharmasug.org)
10 Create Multi-Arm CONSORT Diagrams the Easy Way with SAS®
11 CONSORT Diagrams in SGPLOT: Adding Efficiencies (lexjansen.com)
12 Methods of a Fully Automated CONSORT Diagram Macro... - SAS Support Communities
13 Graphviz
14 Graphviz Visual Editor (magjac.com)
15 Viz.js (viz-js.com)

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Nicolas Guerro
Idorsia Pharmaceuticals Ltd – Hegenheimermattweg 91
4123 Allschwil – Switzerland
nicolas.guerro@dorsia.com