Combinatorial Optimization Project
Steiner Tree Problem

Raymond Lochner

August 2017

1 Introduction

The Steiner Tree Problem is a well known combinatorial optimization problem that requires a set of nodes in a weighted graph to be connected at minimum cost. Given a network $G = (V, E, c)$ with vertices $V = v_1, \ldots, v_n$, edges E, edge weights $cij = c((v_i, v_j)) > 0$ and a set $R, \emptyset \neq R \subseteq V$ or required vertices (or terminals), find a minimum weight tree in G that spans R. These are also called Steiner Points.

This project studies three different formulations to solve the Steiner Tree Problem. The Julia programming language combined with the JuMP package is being used to implement these formulations. Finally, we will finish the report by comparing the computational results of the implementations.

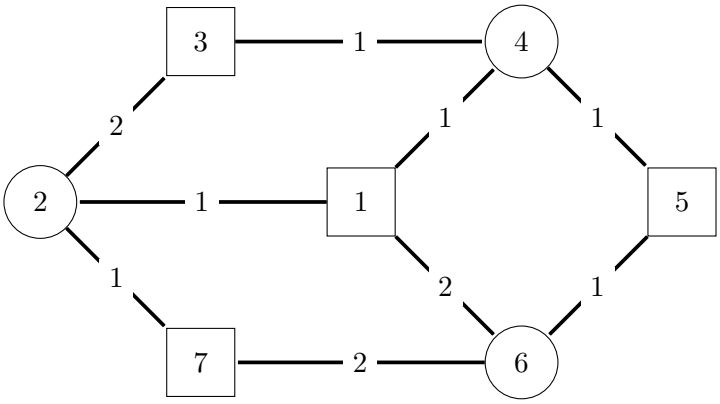
1.1 Further definitions

The following formulations make use of the notation z_i, if it has to be stressed that v_i is a terminal. Reformulations of this problem make also use of a directed graph: $\overrightarrow{G} = (V, A, c), (A := \{[v_i, v_j], [v_j, v_i] | (v_i, v_j) \in E\}, c$ defined accordingly) with a terminal (say z_1) as the root that spans $R_1 := R \setminus \{z_1\}$.

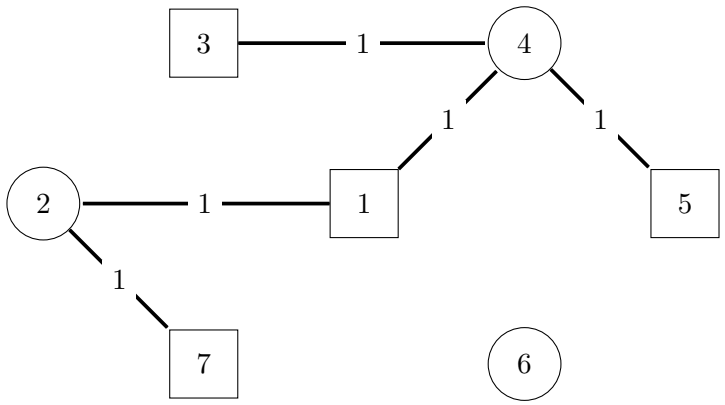
In the integer programming formulations, a binary variable x_{ij} is used to indicate whether an arc is in the solution ($x_{ij} = 1$) or not ($x_{ij} = 0$) for each arc $[v_i, v_j] \in A$.

1.2 Simple example

We shall demonstrate the Steiner Tree Problem by the means of a simple example. Round nodes denote a normal node and a boxed node a Steiner Point. All boxed nodes have to be connected together with a minimum weight.



The graph spanning all Steiner Points with the minimum weight of 5 is shown below:



2 Formulations

Several integer programming formulations for solving this problem are available in the literature. As part of this project, we will be exploring three different approaches:

2.1 Multicommodity Flow Formulation

The Steiner problem can be formulated as a multicommodity flow problem which is stated in [1] by Wong (1984). The general idea behind this formulation is to chose a terminal node z_1 and to find a path from this node to all other Steiner Points $z_t \in R_1$ by the means of defining a flow.

$$\boxed{P_F} \qquad \text{minimize} \sum_{[v_i,v_j]\in A} c_{ij}x_{ij}$$

$$\text{subject to} \sum_{[v_j,v_i]\in A} y^t_{ji} - \sum_{[v_i,v_j]\in A} y^t_{ij} = \begin{cases} 1 & (z_t \in R_1; v_i = z_t) \\ 0 & (z_t \in R_1; v_i \in V \setminus \{z_1, z_t\}) \end{cases} \tag{1.1}$$

$$x_{ij} \geq y^t_{ij} \qquad (z_t \in R_1; [v_i, v_j] \in A) \tag{1.2}$$

$$y^t_{ij} \geq 0 \qquad (z_t \in R_1; [v_i, v_j] \in A) \tag{1.3}$$

$$x_{ij} \in \{0, 1\}, ([v_i, v_j] \in A) \tag{1.4}$$

Variable x_{ij} is binary (4) and denotes if the edge $[v_i, v_j]$ (between node i and j) was chosen in the solution. The cost of each edge is represented by c_{ij}. The variable y^t_{ij} denotes the quantity of commodity t flowing through the edge $[v_i, v_j]$. Constraints (1) and (3) guarantee that for each terminal $z_t \in R_1$, there is a flow of one unit of commodity t from z_1 to z_t.

In other words, we choose a root node z_1 and try to find a path to each terminal z_t. The path is created by the means of sending a flow of one unit from z_1 to z_t for each t. The flow can only be *consumed* by the current terminal node z_t - i.e. have a incoming flow of one and a outgoing flow of 0. This is described in (1). All other nodes, with the exception of the root node z_1, have the same incoming flow as they have a outgoing flow, meaning that if they have an incoming flow, they have to send that flow to some other node. This passively creates a constraint in equation (1) for which the left hand side is -1 when $t = 1$. Equation (2) sets the binary variable to one if there exist a flow between two nodes.

2.2 Two-Terminal Formulation

The Two-Terminal Formulation P_{2T} was proposed by Liu (1990)[2]. It is based on the previous P_F formulation where $|R_1| = 1$. The P_{2T} formulation is based on the case where $|R_1| = 2$. For any two terminals $z_k, z_l \in R_1$, there is a two-terminal tree consisting of a path from z_1 to a splitter node v_s and two paths from v_s to v_k and v_l.

$$\boxed{P_{2T}} \qquad \text{minimize} \sum_{[v_i,v_j]\in A} c_{ij}x_{ij}$$

$$\text{subject to} \sum_{[v_j,v_i]\in A} \check{y}_{ji}^{kl} - \sum_{[v_i,v_j]\in A} \check{y}_{ij}^{kl} \geq \begin{cases} -1 & (\{z_k, z_l\} \subseteq R_1; v_i = z_t) \\ 0 & (\{z_k, z_l\} \subseteq R_1; v_i \in V \setminus \{z_1\}) \end{cases} \tag{2.1}$$

$$\sum_{[v_j,v_i]\in A} (\check{y}_{ji}^{kl} + \grave{y}_{ji}^{kl}) - \sum_{[v_i,v_j]\in A} (\check{y}_{ij}^{kl} + \grave{y}_{ij}^{kl}) = \begin{cases} 1 & (\{z_k, z_l\} \subseteq R_1; v_i = z_t) \\ 0 & (\{z_k, z_l\} \subseteq R_1; v_i \in V \setminus \{z_1\}) \end{cases} \tag{2.2}$$

$$\sum_{[v_j,v_i]\in A} (\check{y}_{ji}^{kl} + \acute{y}_{ji}^{kl}) - \sum_{[v_i,v_j]\in A} (\check{y}_{ij}^{kl} + \acute{y}_{ij}^{kl}) = \begin{cases} 1 & (\{z_k, z_l\} \subseteq R_1; v_i = z_t) \\ 0 & (\{z_k, z_l\} \subseteq R_1; v_i \in V \setminus \{z_1\}) \end{cases} \tag{2.3}$$

$$\check{y}_{ij}^{kl} + \acute{y}_{ij}^{kl} + \grave{y}_{ji}^{kl} \leq x_{ij} \qquad (\{z_k, z_l\} \subseteq R_1; [v_i, v_j] \in A) \tag{2.4}$$

$$\check{y}_{ij}^{kl}, \acute{y}_{ij}^{kl}, \grave{y}_{ji}^{kl} \geq 0 \qquad (\{z_k, z_l\} \subseteq R_1; [v_i, v_j] \in A) \tag{2.5}$$

$$x_{ij} \in \{0, 1\}, ([v_i, v_j] \in A) \tag{2.6}$$

The three various flow variables $\check{y}$,$\grave{y}$ and $\acute{y}$ describe the flow from z_1 to v_s, from v_s to z_k and from v_s to z_l respectively. In (1) the outgoing flow from the root r_1 is described. (2) and (3) describe the incoming flow of one unit of commodity to z_k and z_l. This mimics the formulation from P_f whereas z_1 *sends* flow into the graph and only the nodes z_k and z_l are able to consume the flow, meaning that there is a connection between the nodes z_1, z_k and z_l. (4) is similar to equation (2) in P_F, where we set the binary variable x of the adjacency matrix to one if there is a flow on an arc.

2.3 Common-flow Formulation

The common-flow formulation proposed P_{F^2} by Polzin and Daneshmand (2001) [3] embeds additional variables into the multicommodity flow formulation P_F. Somewhat similar to formulation P_{2T}, it uses the variable $\check{y}^{kl}$ to describe a flow between the nodes z_k and z_l.

$$\boxed{P_{F^2}} \qquad \text{minimize} \sum_{[v_i,v_j]\in A} c_{ij}x_{ij}$$

$$\text{subject to} \sum_{[v_j,v_i]\in A} y_{ji}^{t} - \sum_{[v_i,v_j]\in A} y_{ij}^{t} = \begin{cases} 1 & (z_t \in R_1; v_i = z_t) \\ 0 & (z_t \in R_1; v_i \in V \setminus \{z_1, z_t\}) \end{cases} \tag{3.1}$$

$$\sum_{[v_j,v_i]\in A} \check{y}_{ji}^{kl} - \sum_{[v_i,v_j]\in A} \check{y}_{ij}^{kl} \geq \begin{cases} -1 & (\{z_k, z_l\} \subseteq R_1; v_i = z_t) \\ 0 & (\{z_k, z_l\} \subseteq R_1; v_i \in V \setminus \{z_1\}) \end{cases} \tag{3.2}$$

$$\check{y}_{ij}^{kl} \leq y_{ij}^{k} \qquad (\{z_k, z_l\} \subseteq R_1; [v_i, v_j] \in A) \tag{3.3}$$

$$\check{y}_{ij}^{kl} \leq y_{ij}^{l} \qquad (\{z_k, z_l\} \subseteq R_1; [v_i, v_j] \in A) \tag{3.4}$$

$$y_{ij}^{k} + y_{ij}^{l} - \check{y}_{ij}^{kl} \leq x_{ij} \qquad (\{z_k, z_l\} \subseteq R_1; [v_i, v_j] \in A) \tag{3.5}$$

$$\sum_{[v_j,v_i]\in A} x_{ji} - \sum_{[v_i,v_j]\in A} x_{ji} \leq 0 \qquad (v_i \in V \setminus R) \tag{3.6}$$

$$\check{y}_{ij}^{kl}, y_{ji}^{k} \geq 0 \qquad (\{z_k, z_l\} \subseteq R_1; [v_i, v_j] \in A) \tag{3.7}$$

$$x_{ij} \in \{0, 1\}, ([v_i, v_j] \in A) \tag{3.8}$$

Equation (1) is identical to P_F - It describes the flow from the root z_1 to z_t, where one unit of flow is *consumed* by node z_t, forcing it to be connected to z_1 by a flow in the graph. (2) states that the common flow is non-increasing. (3) and (4) force the common flow to be less or equal than the normal flow; (5) describes the capacity of one arc: is has to be at least the sum of the sum of the common flow to both k and l through an arc minus the common flow y. Equation (6) describes a flow-balance constraint.

3 Implementation

The previous presented formulations have been implemented using the **Julia**[1] programming language in combination with the **JuMP**[2] package. The Steiner instance files[3] can be interpreted by using the *readArgumentFile(ARGS)* function, which is defined in the *stpInterpreter.jl* file. It creates an adjacency matrix consisting of the given weights, as well as the various global variables like lists of nodes, edges and terminals.

It is possible to choose from two different solvers: **GLPK** and **Cbc**. It is necessary to install both before being able to execute the program in julia:

```
Pkg.add("Cbc")
Pkg.add("GLPK")
Pkg.add("GLPKMathProgInterface")
```

Due to the way the program is written, however, it is possible to add additional solvers with very few lines - by adding the solver package at the beginning in the *main.jl* file and by adding 3 lines of an additional if statement in file *stpInterpreter.jl* at line 53.

As all given instances are undirected graphs, we are able to adapt the variable declaration slightly to achieve much faster results. For the formulations P_{2T} and P_{F^2}, z_k and z_l denote the terminals in the current configuration with k and l. It is possible to reduce the number of configurations by only considering each element in R_1 once. Let $R_1 = \{1, 2, 3, 4\}$, then it is possible to consider only the pairs $(\{k = 1, l = 2\}, \{k = 3, l = 4\})$ as they are all connected to the root node z_1. To achieve better computational results, it is therefore advantageous to let each element in R_1 be in as few (l, k) configurations as possible. Indeed, we can let a element occur a maximum of 2 times in all resulting configurations - when $|R_1|$ is uneven. Let $R_1 = \{1, 2, 3\}$, then the only necessary configurations are $(\{k = 1, l = 2\}, \{k = 1, l = 3\})$ which result in the graph with connected nodes $(\{z_1, k = 1, l = 2\}, \{z_1, k = 1, l = 3\})$. The same graph would be calculated twice if we let $(\{k = 1, l = 2\}, \{k = 2, l = 1\})$ for example.

3.1 Executing the program

```
julia main.jl
    -f | --file          [filename]
       --formulation     [PF|P2T|PF2]
    -m | --model         [GLPK|Cbc]
    -v | --verbose       [true|false]
    -s | --saveresult    [filename]
    -h | --help
```

The various options in detail:

- −−file [filename]; Path to the problem instance to be used. This is the only mandatory input parameter.
- −−formulation [PF|P2T|PF2]; Select the specific formulation to solve a problem instance. The three choices are the formulations studied previously. (Default: **PF**)

[1] `https://julialang.org/`
[2] `https://jump.readthedocs.io/`
[3] As provided from `http://steinlib.zib.de/showset.php?B`

- −−model [GLPK|Cbc]; Selects the model to be used to solve the linear program constraints. (Default: **Cbc**)
- −−verbose [true|false]; True: Don't show information to the user in the CLI. (Default: **false**)
- −−saveresult [filename]; Appends to [filename] the information in the following order: instance file name, formulation name, execution time, number of constraints, found objective value. (Default: **Don't save**)
- −−help ; Displays the previously shown information to the user.

4 Computational Experiments

A bash script *runTests.sh* was created to run the three formulations over each instance ten times to receive better average results by reducing possible OS influences. The script saves the results in a file called *results.txt*. Only the Cbc solver with default parameters was used to compute the results as preliminary testing showed significantly faster results for both easy and complex problems using Cbc.

4.1 Hardware information

The following hardware was used during the computational experiments:

- Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
- 16GiB DIMM Synchronous 2133 MHz (0,5 ns)
- B150M MORTAR (MS-7972)

The following commands under Linux were used to obtain more specific details about the used system. Their results are available in the *hardware/* folder:

```
$ lscpu
$ lshw
$ lshw -short
```

4.2 Experimenting

From `http://steinlib.zib.de/showset.php?B` we can verify the known optimal value of a problem instance to verify the correctness of our solution as well as the difficulty of the instance, which is specified in the DC column:

- **L** - Solvable by usage of local preprocessing. Typical examples are the SD-Test, BD-n Tests and FST computations. Neither a global upper nor lower bound needs to be computed.
- **P** - Solvable by polynomial time algorithms, like dual ascent in combination with primal heuristic, a integral LP formulation or advanced preprocessing like reduced cost criteria or the RCR-Test.

The letter after class gives an impression how long it takes to solve the problem using state-of-the-art soft- and hardware. Seconds **(s)** means less than a minute (this includes instances which can be solved in fractions of a second). Minutes **(m)** means less than an hour.This means that we should be able to solve all instances in less than a minute, as they are either Ls or Ps.

The results with averaged computing time, the number of constraints needed and the found optimum can be found in Table 1 on page 6. The graph of the average execution time per formulation per instance is displayed in Figure 1 on page 7. The graph with the varying number of constraints is displayed can be found in Figure 2 on page 8.

Table 1: Execution results of the three formulations

						P_F			P_{2T}			P_{F^2}		
Name	$\|V\|$	$\|E\|$	$\|T\|$	DC	Opt	Avg. Exec. Time	Constraints	Opt	Avg. Exec. Time	Constraints	Opt	Avg. Exec. Time	Constraints	Opt
b01	50	63	9	Ls	82	2.25	1400	82	2.606	1096	82	2.259	2145	82
b02	50	63	13	Ls	83	2.279	2100	83	2.543	1644	83	2.416	3193	83
b03	50	63	25	Ls	138	2.406	4200	138	2.665	3288	138	3.478	6337	138
b04	50	100	9	Ls	59	2.28	1992	59	2.46	1392	59	2.83	3033	59
b05	50	100	13	Ls	61	2.475	2988	61	2.805	2088	61	3.394	4525	61
b06	50	100	25	Ps	122	3.565	5976	122	5.599	4176	122	21.957	9001	122
b07	75	94	13	Ls	111	2.338	3144	111	2.586	2466	111	2.982	4784	111
b08	75	94	19	Ls	104	2.522	4716	104	2.92	3699	104	3.29	7139	104
b09	75	94	38	Ls	220	2.864	9694	220	4.737	7809	220	5.175	14916	220
b10	75	150	13	Ps	86	3.152	4488	86	4.85	3138	86	6.214	6800	86
b11	75	150	19	Ls	88	5.383	6732	88	10.962	4707	88	18.626	10163	88
b12	75	150	38	Ls	174	9.166	13838	174	32.936	9937	174	119.728	21300	174
b13	100	125	17	Ps	165	2.693	5584	165	3.557	4384	165	6.03	8467	165
b14	100	125	25	Ps	235	3.147	8376	235	5.379	6576	235	7.909	12651	235
b15	100	125	50	Ps	318	3.88	17101	318	11.452	13700	318	16.402	26151	318
b16	100	200	17	Ps	127	7.484	7984	127	16.545	5584	127	50.725	12067	127
b17	100	200	25	Ps	131	10.827	11976	131	40.226	8376	131	101.536	18051	131
b18	100	200	50	Ps	218	21.047	24451	218	154.118	17450	218			

Figure 1: Running time of the formulations

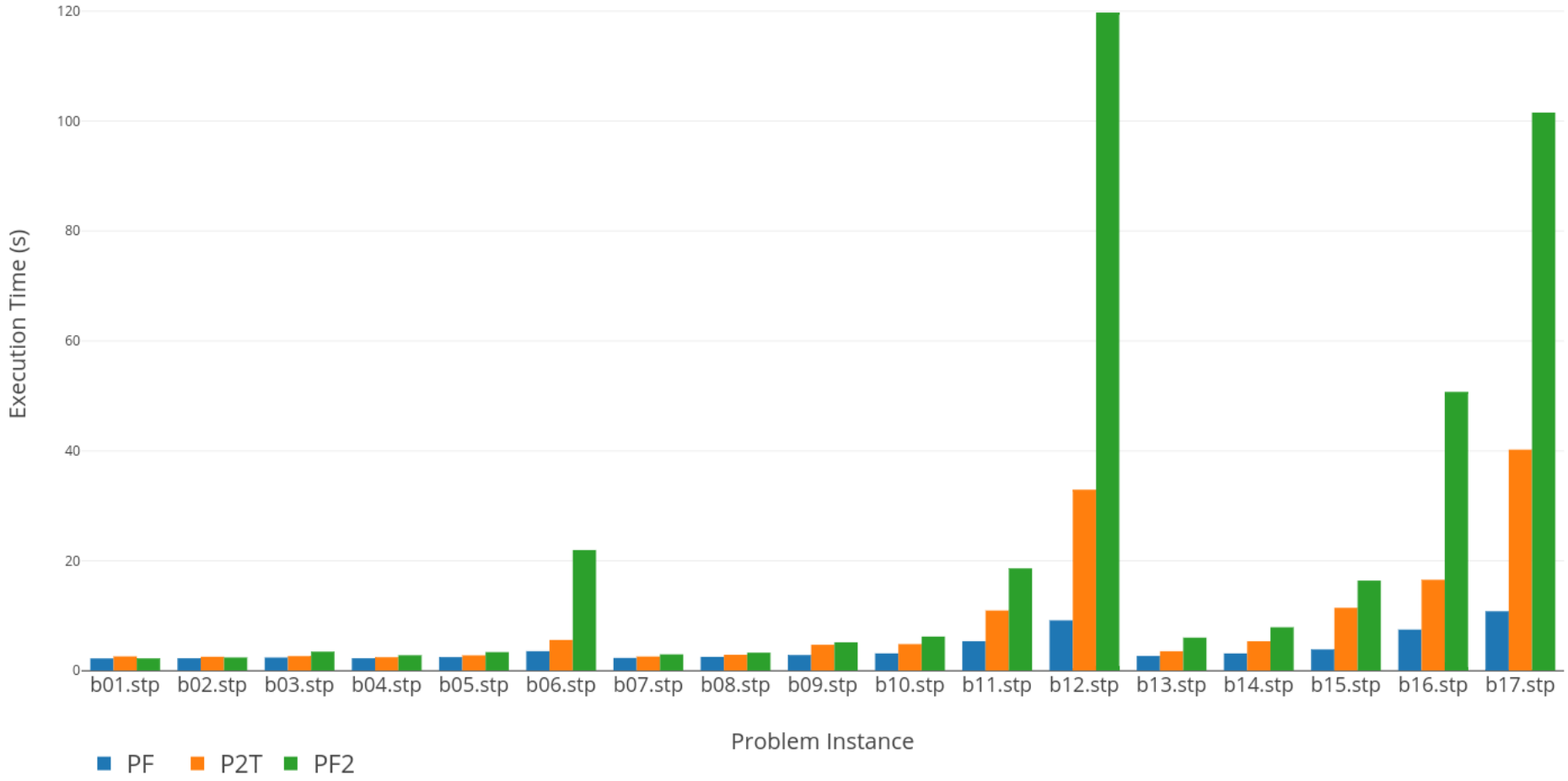


Figure 2: Number of constraints for each of the formulations

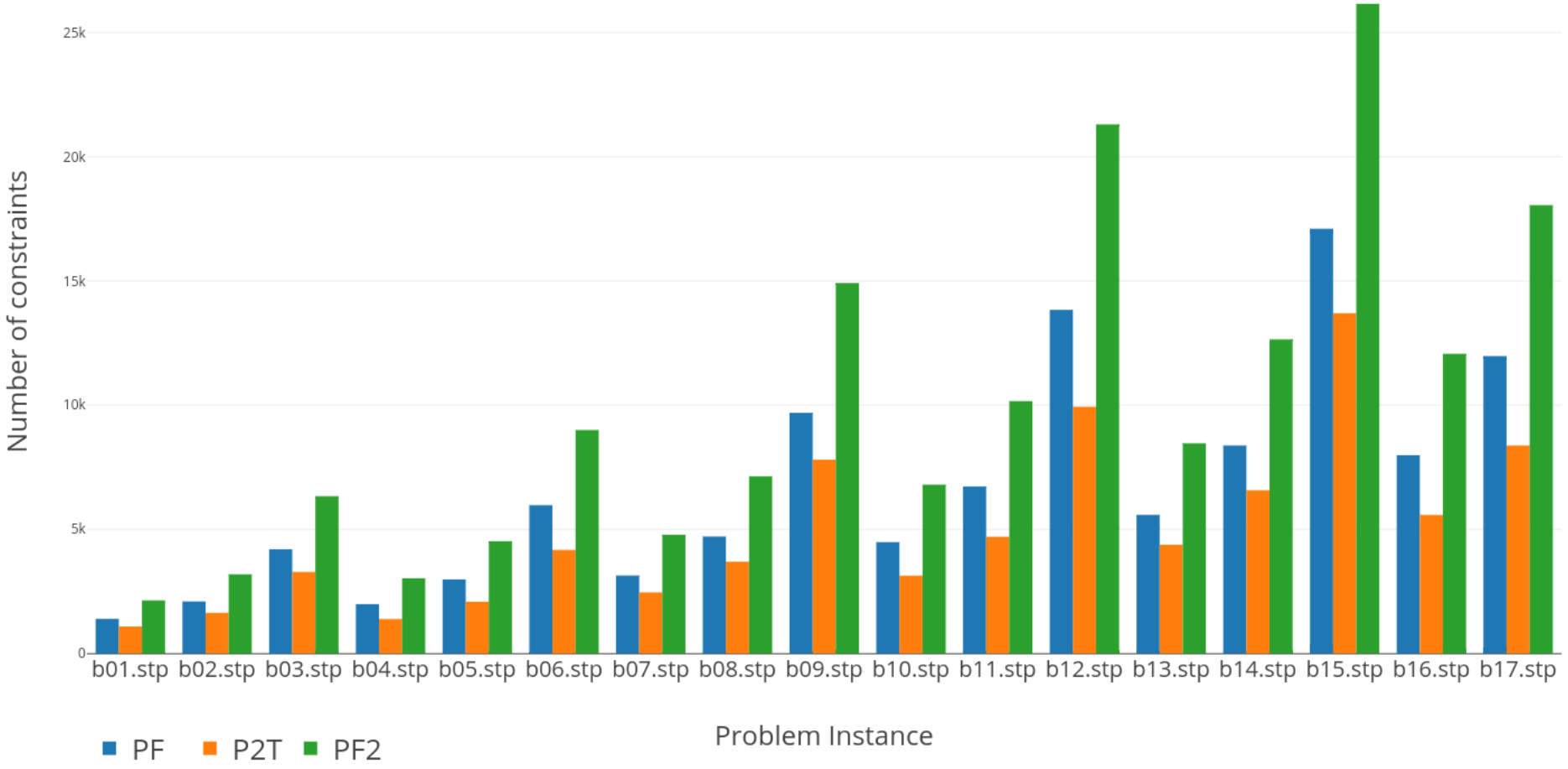
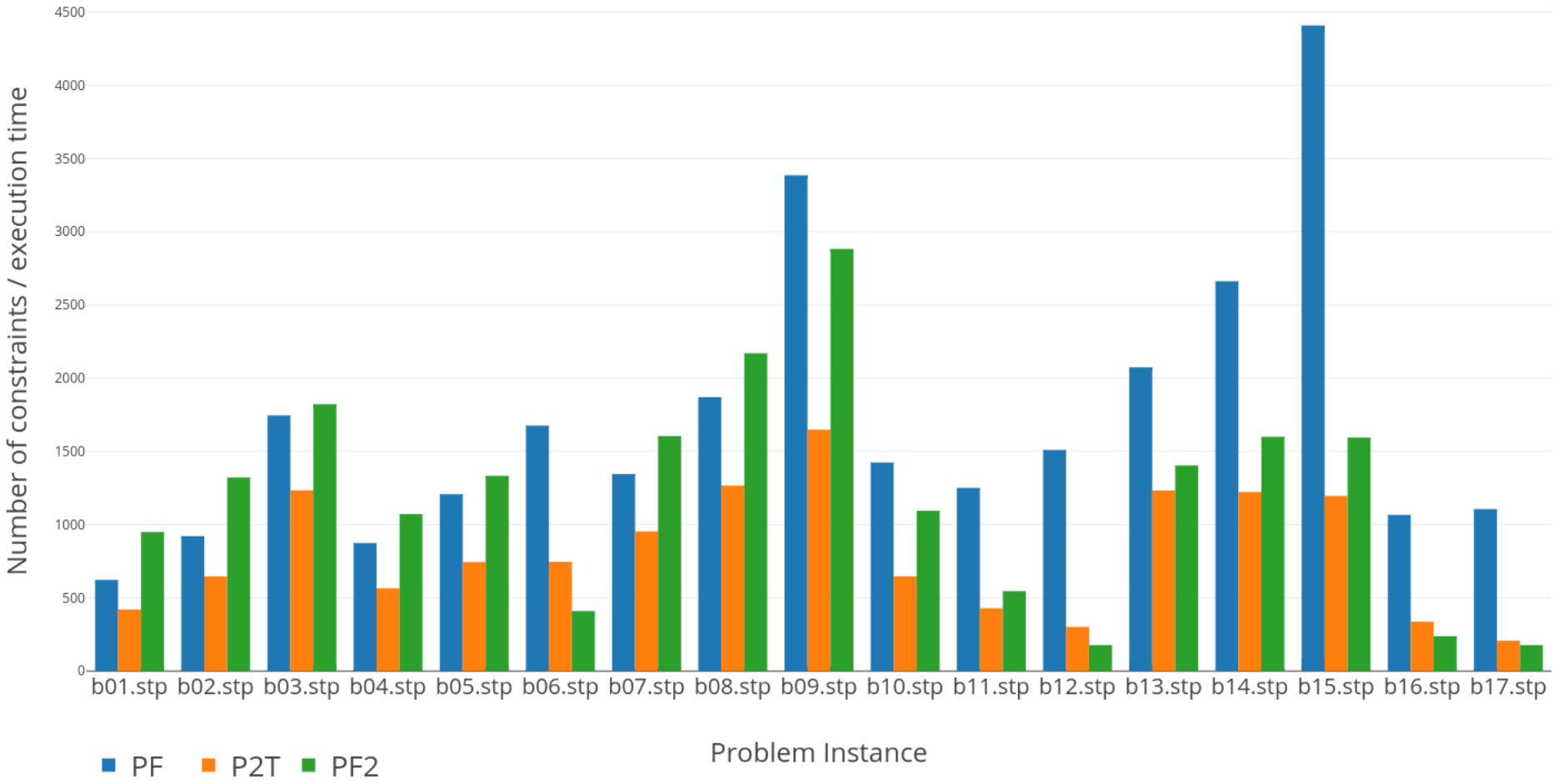


Figure 3: Number of constraints per second to solve



4.3 Results

We can observe that the first formulation P_F arrives at the optimal result significantly faster than the other two, for problems with both P and L difficulty. Only P_F manages to solve all problem instances under a minute and P_{F^2} did not compute a result for problem instance 18 under an hour. In Figure 2 we can see that P_{2T} uses the least number of constraints out of all the formulations and P_{F^2} the most. Formulation P_{F^2} was not able to compute a value for problem instance b18 under an hour, which is why I chose not to include b18 in the graphs. Besides that, every other instances optimal solution was found by all formulations. In Figure 3 on page 9 we can see the *effectiveness* of the constraints to a problem, by dividing the number of constraints by the execution time. From this figure we can derive that most of the time, the formulation P_F is able to *solve* the most constraints per second. Especially for more complex instances (11-17), P_F seems to be a much better choice than the other two formulations.

From these results we can conclude that the first and easiest formulation P_F seems to be the better choice for solving Steiner Tree Problem instances - for easy and harder instances. It should be noted, however, that the problem instances used in this report are all undirected graphs. This means that the result for directed graphs could vary a great deal, as the implemented formulations have been optimized for the undirected case.

References

[1] Richard T. Wong. A dual ascent approach for steiner tree problems on a directed graph. *Mathematical Programming*, 28(3):271–287, Oct 1984.

[2] Weiguo Liu. A lower bound for the steiner tree problem in directed graphs. *Networks*, 20(6):765–778, 1990.

[3] Tobias Polzin and Siavash Vahdati Daneshmand. A comparison of steiner tree relaxations. *Discrete Applied Mathematics*, 112(1):241 – 261, 2001. Combinatorial Optimization Symposium, Selected Papers.