Finding Optimal Size TDMA Schedules using Integer Programming

Mid Sweden University

Felix Dobslaw

felix.dobslaw@miun.se

May 17, 2013

Abstract

The problem of finding minimal size TDMA schedules is formally described as an Integer Program (IP). A brief user manual explains how the attached implementation can be used to find minimal size TDMA for any given WSN and routing table, fulfilling the validity criteria.

1 TDMA scheduling

The TDMA problem considers a given routing for a (weighted) Wireless Sensor Network (WSN) with a $n \times n+q$ connectivity matrix c and a $n \times n+q$ routing matrix r, with n being the amount of sending sensors, and q the amount of sinks. The problem is formulated as the maximization of $f(c,r) = \sum_i E_i, i \in \{1,\ldots,m\}, E_i \in \{0,1\}$, the amount of empty slots, given the upper bound

$$m = \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

for the case that each sensor has one packet waiting for delivery.

The formulation can be found in Table 1. It is restricted to single parent routing (assuming r-r conflict free schedules), and expects the given graph r to be a strict order with the sinks being modeled in position sensor $n+1,\ldots,n+q$. E_i is the amount of empty slots in a TDMA of length m. The formulation maximizes E, meaning that it seeks to find a slot table with a maximum of empty numbers which is equal to finding the shortest TDMA. The result is the $m \times n$ matrix S (optimal size TDMA).

maximize	$\sum_i E_i$	amount of empty slots
subject to	$\sum_j S_{ij} - Y_{1i} \leq 0$	c0. empty slot definition
	$E_i - (1 - Y_{1i}) \leq 0$	$(\sum_j S_{ij} > 0 \Rightarrow E_i = 0)$
	$B_{1j} = 1$	init: buffers start with 1 waiting packet
	$B_{ij} \geq 0$	buffers are non-negative
	$S_{ij} - amountSensors * Y_{2ijl} \leq 0$	c1. t-r, r-t-r conflicts
	$c_{lj} S_{il} - amountSensors * (1 - Y_{2ijl}) \leq 0$	$(S_{ij} = 1 \Rightarrow c_{lj} S_{il} = 0)$
	$\sum_j S_{ij} * C_{jl} \leq 1$	c2. t-t, t-r-t conflicts
	$-\sum_j S_{ij} - amountSensors * Y_{3i} \leq -1$	all buffers are empty after last delivery
	$\sum_j B_{ij} - amountSensors * (1 - Y_{3i}) \leq 0$	
	$\sum_j R_{ij} = 1$	single parent routing
	$S_{ij} - amountSensors * Y_{4ij} \leq 0$	empty buffer $\rightarrow$ no submission
	$-B_{ij} - amountSensors * (1 - Y_{4ij}) \leq -1$	
	$B_{ij} = B_{(i-1)j} - S_{(i-1)j} + \sum_l X_{ijl}$	buffer depends on former transitions
	$R_{lj} + S_{(i-1)l} - Y_{1ijl} \leq 1$	$R_{il} + S_{(i-1)l} = 2 \Leftrightarrow X_{ijl} = 1$
	$X_{ijl} + (1 - Y_{1ijl}) \geq 1$	slack variable X models arrivals
	$X_{ijl} - Y_{2ijl} \leq 0$	
	$R_{lj} + S_{(i-1)l} + 2 * (1 - Y_{2ijl}) \geq 2$	
	$X_{ijl} + Y_{3ijl} \geq 1$	
	$R_{lj} + S_{(i-1)l} - (1 - Y_{3ijl}) \leq 1$	

Table 1: Parameters are written with small letters, variables (to be optimized) with capital letters. The formulation could possibly be improved with regards to efficiency.

2 Program Description

In order for the program to work you have to have Java JRE 1.5 and the gurobi solver in version 5 or higher installed [1]. Once you have registered your installation and set your gurobi environment variables appropriately, you can start the tdmaSolver.jar by:

- `java -jar tdmaSolver.jar`.

The program expects a properties file by name '*wsnSetup*' in the same folder, which defines three parameters:

wsn the path to the network to be solved (mandatory)

routing the path to a routing table for the network (optional)

buffer the initial buffer setting for the network (optional)

The file content could look as in:

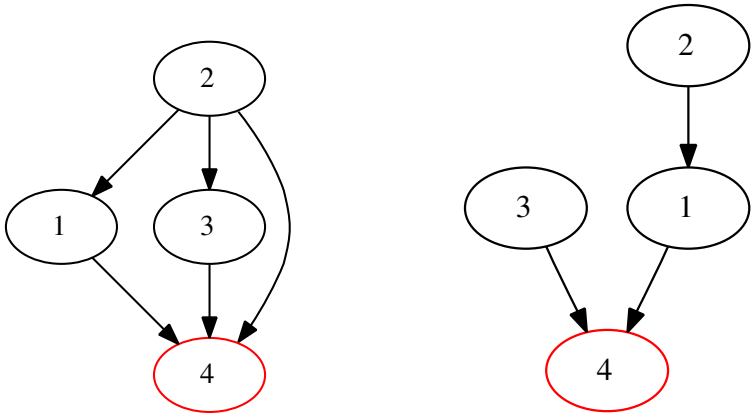
[1] Take a look at www.gurobi.com for an howto on installing it (you can attain a academic license for scientific purposes)

```
wsn=myWSN.txt
routing=myRouting.txt
buffer=myBuffer.txt
```

Solely defining the network potentially leads to better results (and substantially longer running times), because of less restrictions (no extra routing constraints). The upper bound m takes the amount of initial packets in the buffer into consideration.

3 Example

Given are the network in Figure 1a and the valid routing 1b.



(a) A trivial example WSN with 4 nodes. (b) A routing for the trivial example WSN in 1a.

The equivalent in-file decsription is formulated as in:

```
WSN:
00000000100000000000
00100000001000000010
00010001000000000100
00100000000010000000
00000000000000010000
10000000000000000000
01000000000000000000
00000000001000000001
00000000000000010000
00100000000000000000
00010000000000100000
01000000100001000000
01000000001000010000
```

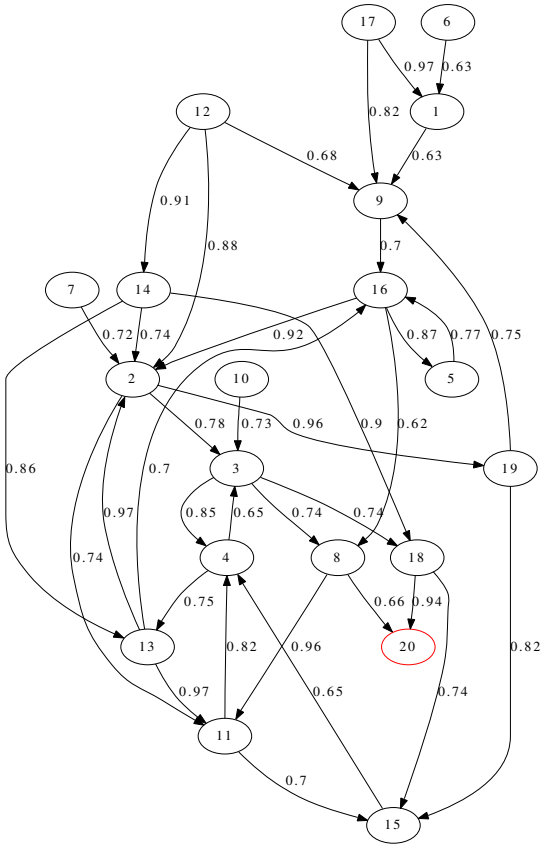


Figure 1: The test network to be optimized, consisting of 20 nodes.

```
01000000000010000100
00010000000000000000
01001001000000000000
10000000100000000000
00000000000000100001
00000000100000100000
```

1s stand for positive connectivity, 0s for no connectivity in the WSN file. In the routing file, 0s stand for no route, 1s for given route. Currently, based on the former problem formulation and this syntax, only non-weighted graphs are supported for optimization. The program will print the length of the optimal TDMA via system out, followed by one[2] optimal size schedule. Important note:

- **The algorithm gives only appropriate solutions for graphs that connect each sensor to a sink!**

The output for the example looks as follows:

```
Optimize a model with 553501 rows, 506880 columns and 1249270 nonzeros
Presolve removed 410499 rows and 458901 columns (presolve time = 6s) ...
Presolve removed 410499 rows and 458901 columns (presolve time = 10s) ...
Presolve removed 461604 rows and 458903 columns
Presolve time: 11.61s
Presolved: 91897 rows, 47977 columns, 324651 nonzeros
Variable types: 0 continuous, 47977 integer (45621 binary)
Found heuristic solution: objective -0.0000000

Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
    6448    1.2025872e+02   1.353778e+04   0.000000e+00      5s
    9200    1.2025797e+02   1.434810e+04   0.000000e+00     10s
   11264    1.2025670e+02   3.295059e+05   0.000000e+00     15s
   12812    1.2024009e+02   1.500940e+07   0.000000e+00     20s
   14360    1.1783776e+02   1.785668e+06   0.000000e+00     25s
   15736    1.1633980e+02   1.913384e+05   0.000000e+00     30s
   17112    1.1570554e+02   3.282639e+06   0.000000e+00     36s
   18488    1.1508934e+02   4.145550e+05   0.000000e+00     40s
Warning: Markowitz tolerance tightened to 0.03125
   19864    1.1433710e+02   7.922502e+05   0.000000e+00     45s
   21412    1.1411626e+02   1.050797e+05   0.000000e+00     50s
   22788    1.1380968e+02   8.141415e+06   0.000000e+00     55s
   23992    1.1367013e+02   1.728571e+05   0.000000e+00     60s
   25368    1.1348785e+02   6.207536e+04   0.000000e+00     65s
   26744    1.1324074e+02   5.421782e+04   0.000000e+00     70s
```

[2]There could be multiple, but the program stops after finding one.

```
   28120    1.1317063e+02   1.029490e+04   0.000000e+00     75s
   29324    1.1311170e+02   9.164295e+03   0.000000e+00     80s
   30872    1.1298886e+02   6.719229e+02   0.000000e+00     85s
   32420    1.1293671e+02   4.795785e+04   0.000000e+00     91s
   33624    1.1293410e+02   3.400464e+03   0.000000e+00     95s
   34353    1.1293510e+02   0.000000e+00   0.000000e+00     98s

Root relaxation: objective 1.129351e+02, 34353 iterations, 98.18 seconds
Total elapsed time = 115.54s

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  112.93510    0 3339   -0.00000  112.93510     -      -  118s
     0     0   99.38295    0 3190   -0.00000   99.38295     -      -  271s
     0     0   99.22089    0 3338   -0.00000   99.22089     -      -  288s
     0     0   99.16583    0 3339   -0.00000   99.16583     -      -  305s
     0     0   99.13332    0 3372   -0.00000   99.13332     -      -  315s
     0     0   99.11349    0 3415   -0.00000   99.11349     -      -  324s
     0     0   99.10528    0 3342   -0.00000   99.10528     -      -  332s
     0     0   99.10147    0 3314   -0.00000   99.10147     -      -  563s
     0     0   99.09358    0 3382   -0.00000   99.09358     -      -  568s
     0     0   99.08600    0 3373   -0.00000   99.08600     -      -  574s
     0     0   99.08265    0 3382   -0.00000   99.08265     -      -  580s
     0     0   99.07641    0 3183   -0.00000   99.07641     -      -  592s
     0     0   99.06997    0 3322   -0.00000   99.06997     -      -  605s
     0     0   99.06845    0 3247   -0.00000   99.06845     -      -  619s
     0     0   99.06437    0 3324   -0.00000   99.06437     -      -  630s
     0     0   99.05293    0 3315   -0.00000   99.05293     -      -  644s
     0     0   99.05145    0 3469   -0.00000   99.05145     -      -  653s
     0     0   99.04969    0 3270   -0.00000   99.04969     -      -  663s
     0     0   99.04928    0 3314   -0.00000   99.04928     -      -  675s
     0     0   99.04839    0 3294   -0.00000   99.04839     -      -  688s
     0     0   99.04681    0 3338   -0.00000   99.04681     -      -  699s
     0     0   99.04662    0 3290   -0.00000   99.04662     -      -  711s
     0     0   99.04589    0 3357   -0.00000   99.04589     -      -  721s
     0     0   99.00354    0 3361   -0.00000   99.00354     -      -  732s
     0     0   99.00354    0 3360   -0.00000   99.00354     -      -  734s
H    0     0                       94.0000000   99.00354  5.32%     -  782s
     0     2   99.00354    0 3360   94.00000   99.00354  5.32%     -  791s
     1     3   98.92936    1 3341   94.00000   98.92936  5.24%  2901  795s
     3     3     cutoff    3        94.00000   98.82752  5.14%  7364  878s
     4     4   98.73492    3 3238   94.00000   98.73492  5.04%  5848  883s
     5     2     cutoff    4        94.00000   98.73126  5.03%  6550  925s
     6     3   97.93249    4 3145   94.00000   97.93249  4.18%  7461  988s
     8     1     cutoff    5        94.00000   97.89818  4.15%  6423 1016s
```

```
     9     2   97.09808    5 3215   94.00000   97.09808  3.30%  6102 1031s
    12     2     cutoff    7        94.00000   97.09642  3.29%  6727 1186s
    13     3   96.82057    6 3132   94.00000   96.97763  3.17%  6741 1214s
    14     4   96.23050    7 2939   94.00000   96.82057  3.00%  6452 1223s
    15     3     cutoff    8        94.00000   96.82057  3.00%  7037 1299s
    16     4   95.33293    8 2944   94.00000   96.82057  3.00%  6696 1304s
    17     2   95.33293    9 2944   94.00000   96.82057  3.00%  6302 1345s
    21     3   95.06702    9 2950   94.00000   96.82057  3.00%  5657 1366s

Cutting planes:
  Gomory: 4
  Cover: 980
  Implied bound: 1365
  Clique: 1301
  MIR: 5
  GUB cover: 166
  Zero half: 27

Explored 26 nodes (258052 simplex iterations) in 1367.17 seconds
Thread count was 2 (of 2 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 9.400000000000e+01, best bound 9.400000000000e+01, gap 0.0%
Routing:

A result of minimal size 31 could be found in 1367406 ms.

Routing:
00000000100000000000
00000000000000000010
00010000000000000000
00000000000010000000
00000000000000010000
10000000000000000000
01000000000000000000
00000000000000000001
00000000000000010000
00100000000000000000
00010000000000000000
00000000000001000000
01000000000000000000
00000000000000000100
00010000000000000000
00000001000000000000
10000000000000000000
00000000000000000001
```

```
00000000100000000000

Schedule and buffer:
0100000111100000000   1111111111111111111
0000000000001111001   1022111000011112112
1001000000000001010   1123111110010001121
0001010110000000000   0122111220011000111
0010000110000000100   1121101110012001111
1010001000000001000   2112101000012002011
1000100100000000000   1203100110012001011
0000000000001001010   0203000020012002011
0100000110000000000   0303000120011001001
0100000000000001000   0203000010011002002
0101000110000000000   0103000110011001003
0000000000001001000   0002000000012002004
0001000000010001001   0102000100011001004
0000000110000000000   0101000210002100003
0000000100001000001   0101000100002101003
0000000000001001001   0201000010001101002
0101000110000100000   0301000120000100001
0100000000000001010   0200000010001001012
0100000110000000000   0100000110001000003
0000000000000001001   0000000000001001004
0000000110001000000   0000000110001000003
0000000000000001001   0100000000000001003
0100000110000000000   0100000110000000002
0000000000000001001   0000000000000001003
0000000110000000000   0000000110000000002
0000000000000001001   0000000000000001002
0000000110000000000   0000000110000000001
0000000000000001001   0000000000000001001
0000000110000000000   0000000110000000000
0000000000000001000   0000000000000001000
0000000100000000000   0000000100000000000
```