Process Discovery Contest 2016: Heuristic Alpha+ Miner (HAM)

Moshe Shteiner, Liat Bodaker, Arik Senderovich*

The process discovery contest[1], 2016, took place in Rio De Janeiro, as part of the Business Process Intelligence 2016 Workshop. Designing an algorithm for process discovery was an interesting and fascinating experience, and in many cases very challenging. Below, we briefly outline the essentials of the developed software and algorithm.

The heuristic alpha miner, or the HAM, have been created in the C# environment. We developed a full environment to read, parse and discover process models from event logs. The basic algorithm deploys the well-known Alpha+ algorithm [1]. The source code appears in http://bit.ly/2eQPgAw.

As Alpha+ has some limitations, we developed some workarounds, as well as a special *repair* package that works on a model by model basis. To emphasize the latter, each of the logs that were published in the contest came from different models with different characteristics, which were known in advance. Hence, we constructed the repair package to take advantage of that knowledge. As a basic idea, the log footprint matrices were used to repair some of the disadvantages or weaknesses of the Alpha+ algorithm. The Alpha+ footprints are essentially one-step relations between consecutive activities in order of their appearance in the log. Instead of looking only at the footprints we perform a two-step lookahead and look-back, and enable backtracking if we find complex relations between the different activities. This increases the ability to capture long-term dependencies that the Alpha+ algorithm cannot represent in the resulting model.

In some cases, the repair was not sufficient, and therefore we implemented ad-hoc solutions to tune the model according to the test log results. The modifications in repair are model specific and use activity names to re-connect or to de-attach activities from one another.

Our software has a built-in visualization package. However if GraphViz [2] is installed in the system, the program can display higher quality results. The resulting models are workflow nets, i.e. Petri nets with specific characteristics [3]. Further, we have implemented a *replay* package to enable log replay. The main limitation of the replay component is that it is greedy, and does not seek optimal alignments.

*e-mails: moshe@steiners.co.il, liat.bodaker@reali.org.il, sariks@campus.technion.ac.il

[1]

The running of our software is straightforward: it is possible to read the training log first, and then the test log file, for an output that would fit the contest format. It is also possible to read all pairs of train/test logs and run everything in a batch mode. Runtime is quite fast as most of the models run within a few seconds or less. The longest model runs for about 20 seconds, due to an extensive use of replay and repair. Important note: we rely on the model number $(1, \ldots, 10)$ to be part of the training log filename. We use it to tune the model discovery process. For example: training1.csv will be identified as model 1. In the appendix, we provide the 10 models that we mined from the 10 training event logs provided by the contest organizers.

References

[1] AK Alves de Medeiros, Boudewijn F van Dongen, Wil MP Van der Aalst, and AJMM Weijters. Process mining: Extending the α-algorithm to mine short loops. Technical report, BETA working paper series, WP 113, Eindhoven University of Technology, Eindhoven, 2004. 1

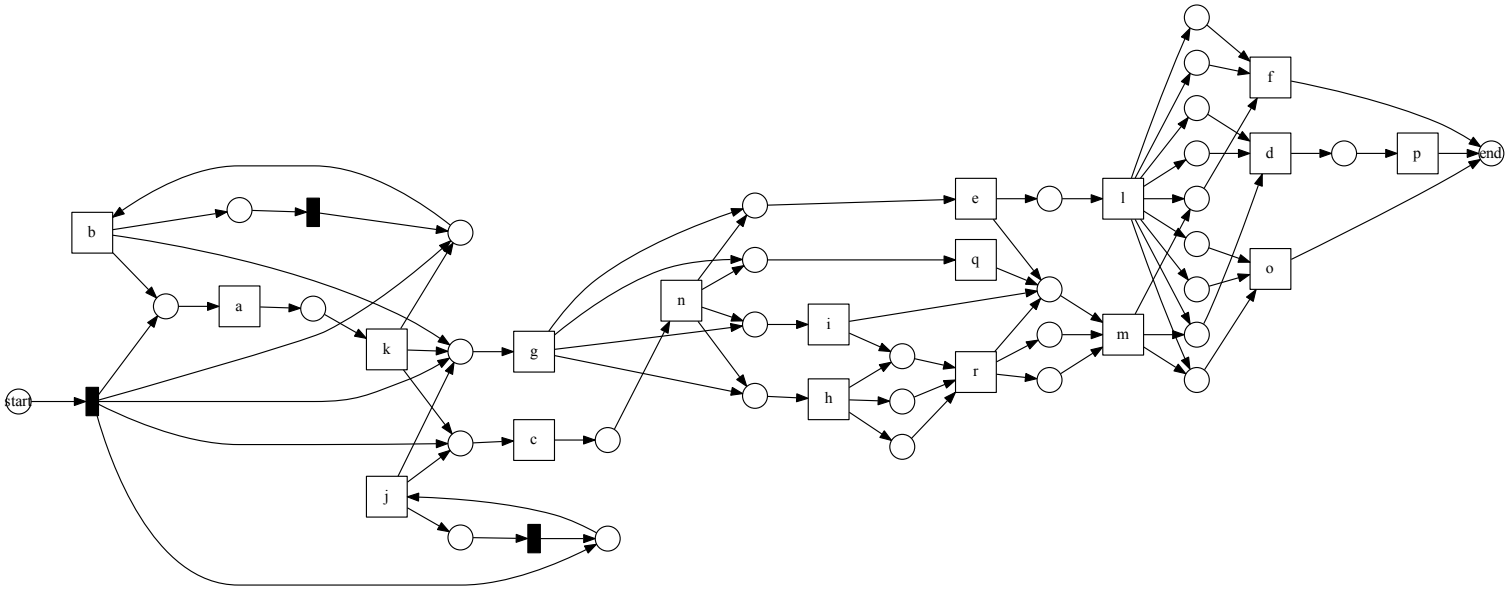
[2] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*, 30(11):1203–1233, 2000. 1

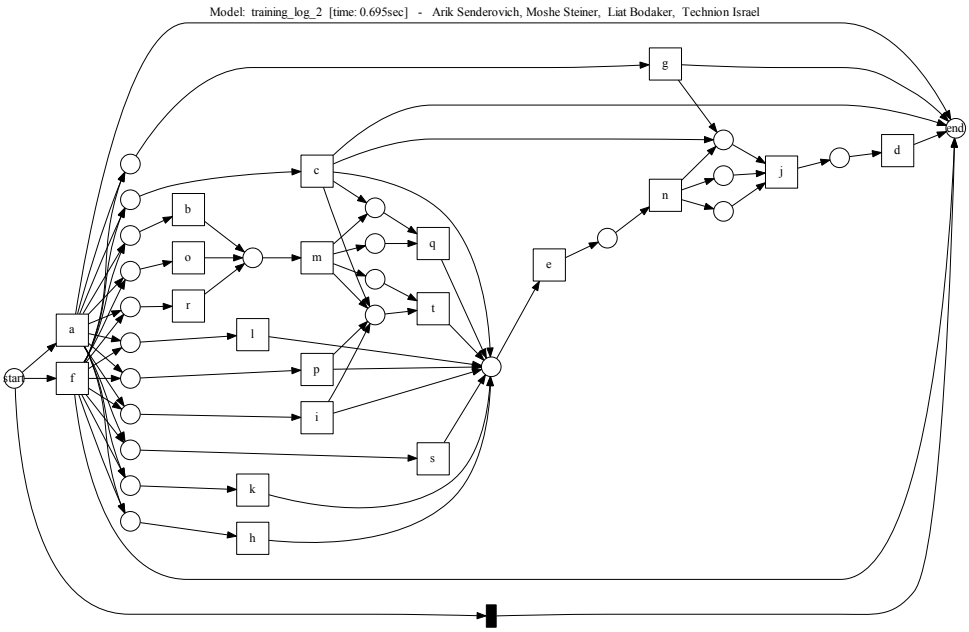
[3] Wil MP Van der Aalst. Verification of workflow nets. In *International Conference on Application and Theory of Petri Nets*, pages 407–426. Springer, 1997. 1

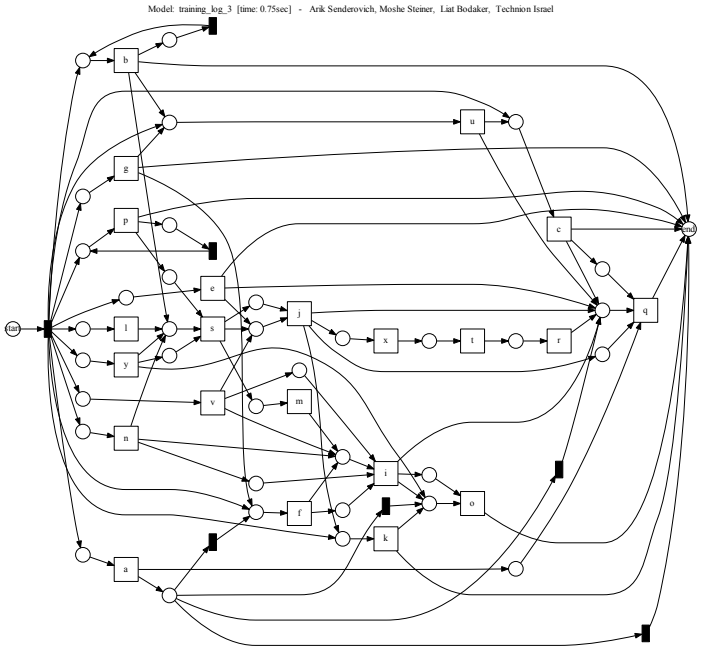
A Appendix: The Process Models

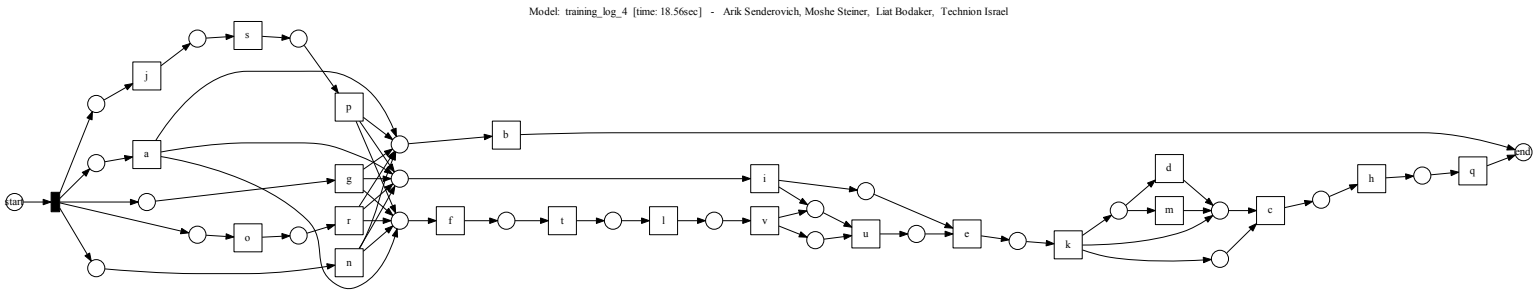
Below, we provide the 10 models that we mined from the 10 training event logs provided by the contest organizers.

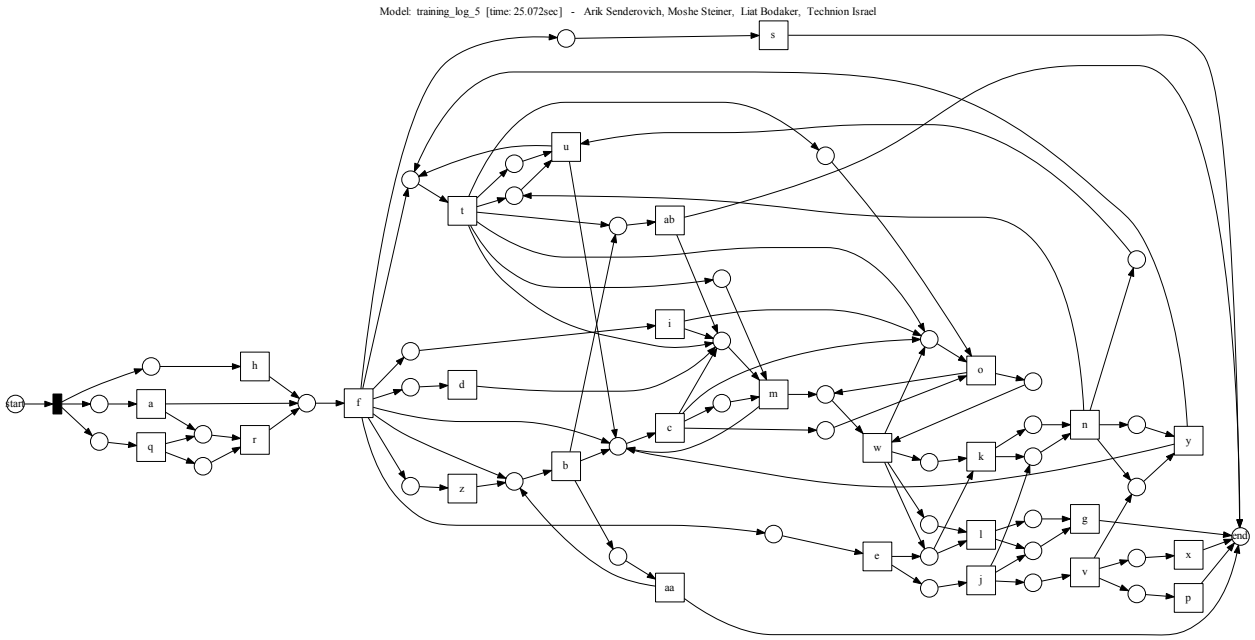
Model: training_log_1 [time: 0.724sec] - Arik Senderovich, Moshe Steiner, Liat Bodaker, Technion Israel

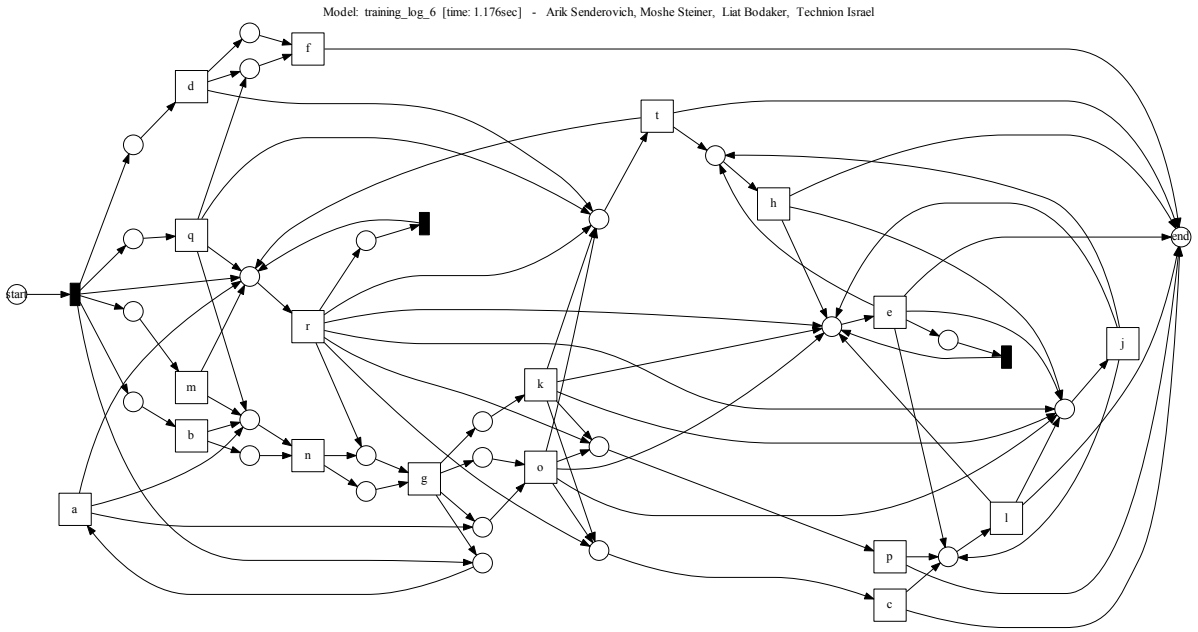


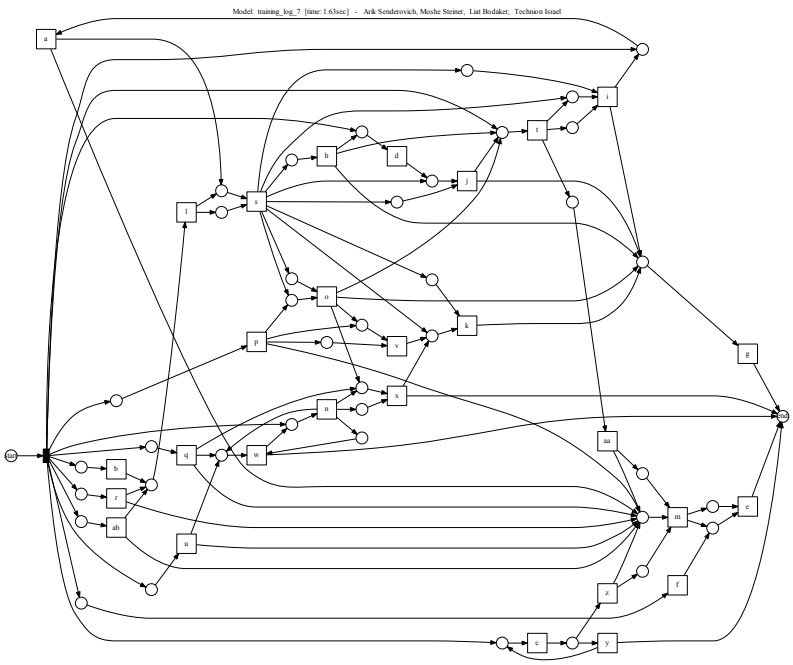

Model: training_log_2 [time: 0.695sec] - Arik Senderovich, Moshe Steiner, Liat Bodaker, Technion Israel
start
end
a
f
b
o
r
l
c
m
p
i
k
h
q
t
s
e
g
n
j
d

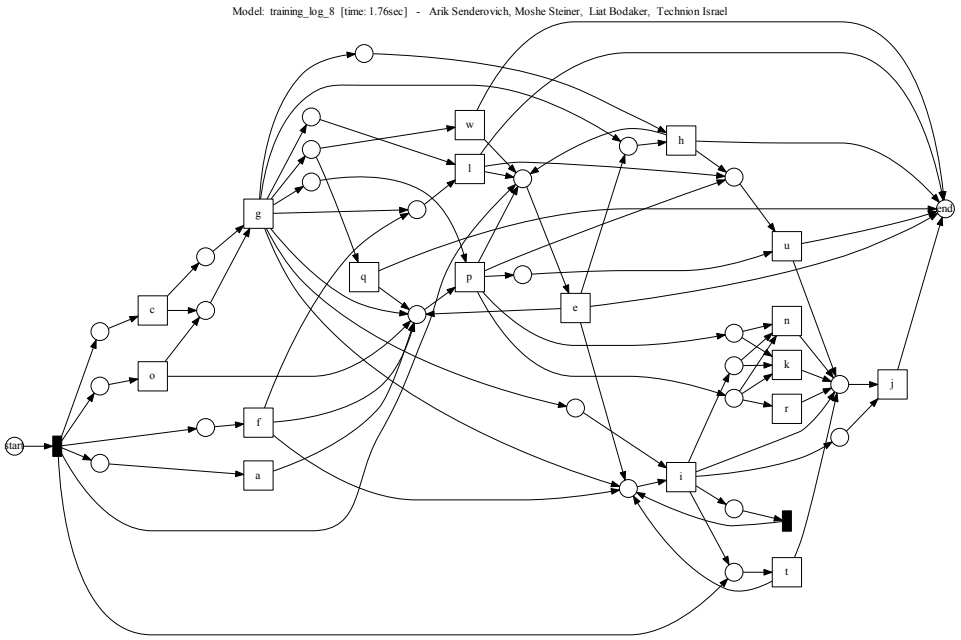

Model: training_log_3 [time: 0.75sec] - Arik Senderovich, Moshe Steiner, Liat Bodaker, Technion Israel
start
end
b
u
g
p
c
e
q
l
s
j
x
t
r
y
v
m
n
i
o
f
k
a


Model: training_log_4 [time: 18.56sec] - Arik Senderovich, Moshe Steiner, Liat Bodaker, Technion Israel
start
end
s
j
p
a
g
r
o
n
b
f
t
l
i
v
u
e
k
d
m
c
h
q

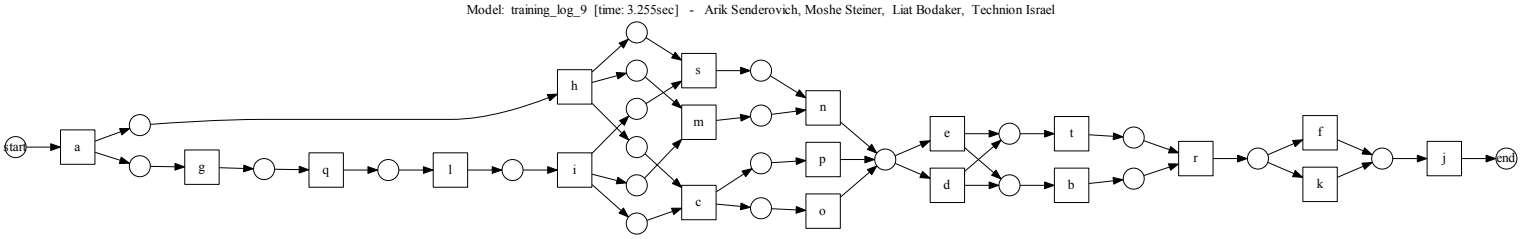

Model: training_log_5 [time: 25.072sec] - Arik Senderovich, Moshe Steiner, Liat Bodaker, Technion Israel
start
a
h
q
r
f
s
u
t
ab
i
d
m
o
c
w
k
n
y
b
z
l
g
e
j
v
x
p
aa
end

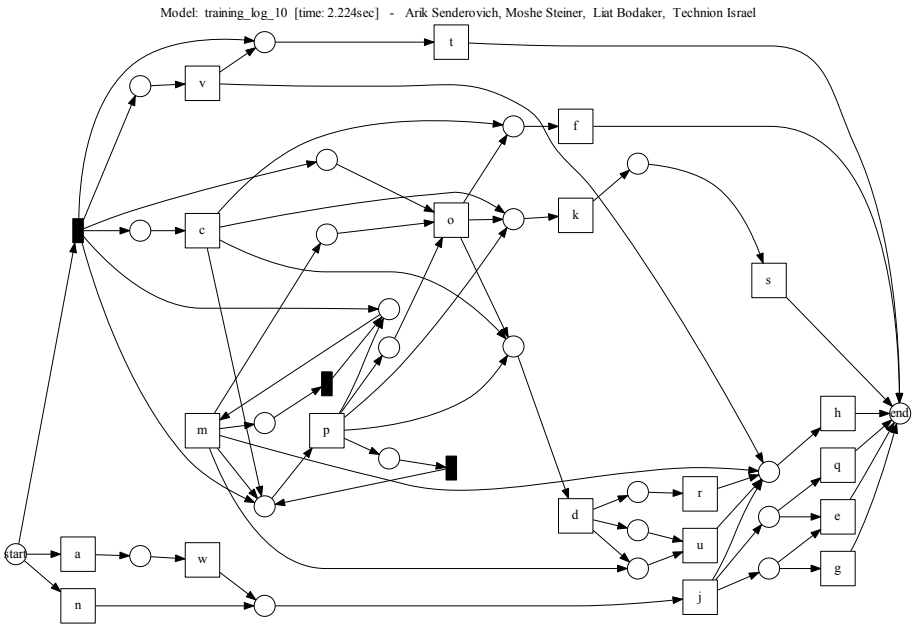

Model: training_log_6 [time: 1.176sec] - Arik Senderovich, Moshe Steiner, Liat Bodaker, Technion Israel
start
end
a
b
c
d
e
f
g
h
j
k
l
m
n
o
p
q
r
t


Model: training_log_7 [time: 1.63sec] - Arik Senderovich, Moshe Steiner, Liat Bodaker, Technion Israel
a
i
t
h
d
j
l
s
o
k
p
v
g
x
n
w
q
aa
b
r
ab
u
m
e
z
f
c
y


Model: training_log_8 [time: 1.76sec] - Arik Senderovich, Moshe Steiner, Liat Bodaker, Technion Israel
start
end
w
l
h
g
q
p
u
c
e
o
n
k
r
j
f
a
i
t

Model: training_log_9 [time: 3.255sec] - Arik Senderovich, Moshe Steiner, Liat Bodaker, Technion Israel




Model: training_log_10 [time: 2.224sec] - Arik Senderovich, Moshe Steiner, Liat Bodaker, Technion Israel
start
a
w
n
v
t
f
c
o
k
s
m
p
d
r
u
j
h
q
e
g
end