Approximation Algorithms for Loop-Free Updates in Software-Defined Networks

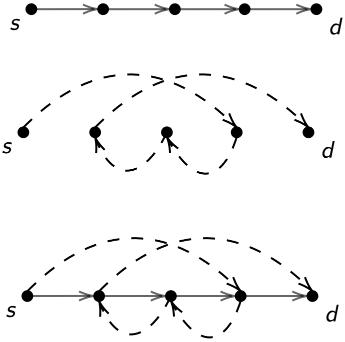
Radu Vintan

Technische Universität München

August 16, 2022

Problem Definition

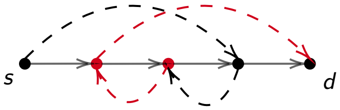
Two paths/permutations from s to d. Change forwarding tables of all routers.



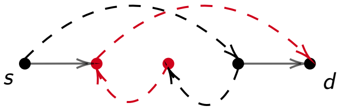
Asynchronous Updates

Cannot control order of updates in one **round**. E.g. if we update $\{2, 3\}$.

During round both new and old edges can be used:

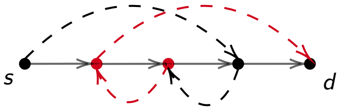


After round only new edges remain:



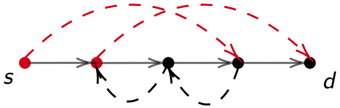
Strong Loop Freedom

But... no loops allowed at any time.
Strong Loop Freedom (SLF)

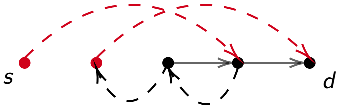


not allowed... How to solve this instance?

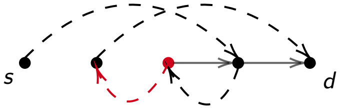
Round 1



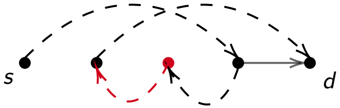
Round 1



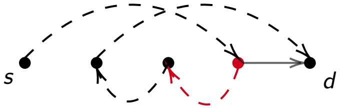
Round 2



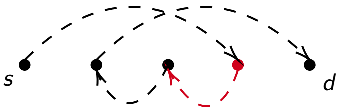
Round 2



Round 3



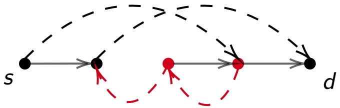
Round 3



Relaxed Loop Freedom

Cycles allowed as long as not reachable from s.
Relaxed Loop Freedom (RLF)

E.g. this is allowed (2 already updated, update $3, 4$):



Optimization Problem

Solve SLF/RLF instance with n nodes using minimal number of rounds. Known facts:

- For SLF: instances requiring $\Omega(n)$ rounds exist.
- For RLF: there is always a solution with $\mathcal{O}(\log n)$ rounds [Foerster et al., 2018].
- NP-hardness proved only for SLF [Foerster et al., 2018].

SLF: LP Approach

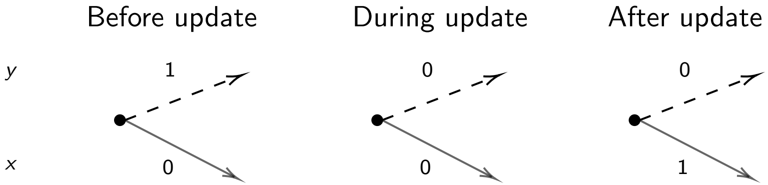
Solve decision version: can I solve instance in T rounds?

Encode using linear constraints.
For now, think of the variables as 0 or 1.

1 encodes forbidden edges, 0 encodes allowed edges.

y_{ti} for new edge of i, x_{ti} for old edge of i

Relation between x and y



$$x_{ti} = 1 - y_{t-1,i}, \ \forall t \ \forall i$$

Other linear constraints

Initially, we use old edges:

$$y_{0i} = 1, \ \forall i$$

At end, we use new edges:

$$y_{Ti} = 0, \ \forall i$$

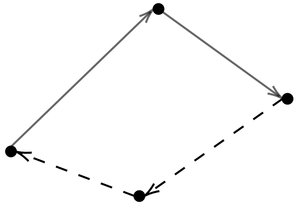
Cannot go from new edge back to old edge:

$$y_{t-1,i} \geq y_{ti}, \ \forall t \ \forall i$$

No cycles

For any cycle C:

$$\sum_{i \in C_{\text{old}}} x_{ti} + \sum_{y \in C_{\text{new}}} y_{ti} \geq 1, \quad \forall t$$



Separation oracle can be implemented using Floyd-Warshall.

Result

$$
\begin{aligned}
y_{0i} &= 1, \quad \forall i \\
y_{Ti} &= 0, \quad \forall i \\
x_{ti} + y_{t-1,i} &= 1, \quad \forall t\ \forall i \\
y_{t-1,i} - y_{ti} &\geq 0, \quad \forall t\ \forall i \\
\sum_{i \in C_{\text{old}}} x_{ti} + \sum_{y \in C_{\text{new}}} y_{ti} &\geq 1 \quad \forall t\ \forall C \text{ cycle}
\end{aligned}
$$

To solve SLF optimally: find minimal T for which above contains *integral feasible point*.

But how to find an approximation algorithm, i.e. poly-time?
Relax above to $x, y \in [0, 1]$, solve *fractionally* and *round* (how?)

Important Implementation Detail

Add cycle constraint *lazily*:

- Start with no cycle constraints at all.
- Find (integral/fractional) point in current version.
- Check using the separation oracle if any cycle is too short.
- Add such cycles to constraints if they exist and repeat.

Leads to excellent performance.

Rounding

Computed optimal fractional solution for $n = 6$ nodes and $T = 3$.

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
$t = 0$	1.0	1.0	1.0	1.0	1.0
$t = 1$	1.0	1.0	0.5	0.5	0.0
$t = 2$	1.0	0.5	0.0	0.5	0.0
$t = 3$	0.0	0.0	0.0	0.0	0.0

Integralizing row 1 and obtaining new feasible solution with $T' \in \{3, 4\}$

Rounding

E.g. now get:

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$
$t=0$	1.0	1.0	1.0	1.0	1.0
$t=1$	1.0	1.0	1.0	1.0	0.0
$t=2$	1.0	1.0	0.0	0.5	0.0
$t=3$	0.0	0.5	0.0	0.5	0.0
$t=4$	0.0	0.0	0.0	0.0	0.0

Integralizing row 1 and obtaining new feasible solution with $T' \in \{3,4\}$

Rounding

E.g. now get:

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$
$t=0$	1.0	1.0	1.0	1.0	1.0
$t=1$	1.0	1.0	1.0	1.0	0.0
$t=2$	1.0	1.0	0.0	0.5	0.0
$t=3$	0.0	0.5	0.0	0.5	0.0
$t=4$	0.0	0.0	0.0	0.0	0.0

Integralizing row 2 and obtaining new feasible solution with $T' \in \{4, 5\}$

Rounding

E.g. now get:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
$t = 0$	1.0	1.0	1.0	1.0	1.0
$t = 1$	1.0	1.0	1.0	1.0	0.0
$t = 2$	1.0	1.0	0.0	1.0	0.0
$t = 3$	0.0	1.0	0.0	0.0	0.0
$t = 4$	0.0	0.0	0.0	0.0	0.0

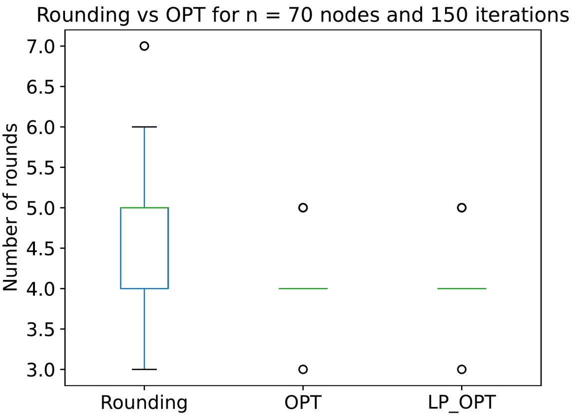
Integralizing row 2 and obtaining new feasible solution with $T' \in \{4, 5\}$

Rounding

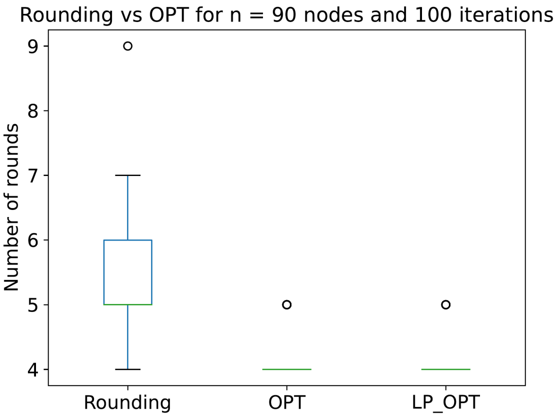
	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
$t = 0$	1.0	1.0	1.0	1.0	1.0
$t = 1$	1.0	1.0	1.0	1.0	0.0
$t = 2$	1.0	1.0	0.0	1.0	0.0
$t = 3$	0.0	1.0	0.0	0.0	0.0
$t = 4$	0.0	0.0	0.0	0.0	0.0

Done.

Plots

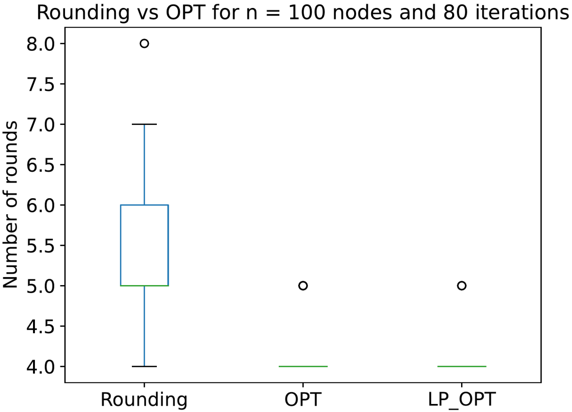

Rounding vs OPT for n = 70 nodes and 150 iterations
Number of rounds
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
Rounding
OPT
LP_OPT

Plots

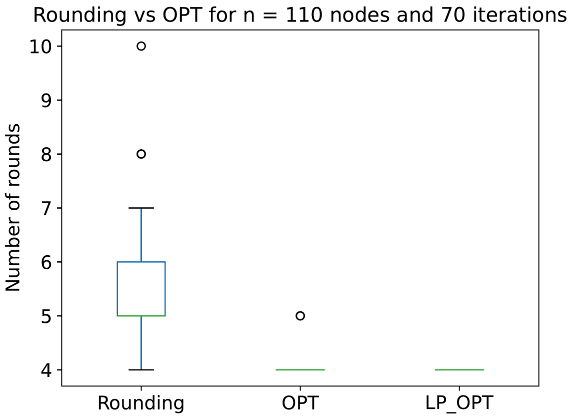


Rounding vs OPT for n = 90 nodes and 100 iterations
Number of rounds
9
8
7
6
5
4
Rounding
OPT
LP_OPT

Plots


Rounding vs OPT for n = 100 nodes and 80 iterations
Number of rounds
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
Rounding
OPT
LP_OPT

Plots



Rounding vs OPT for n = 110 nodes and 70 iterations
Number of rounds
10
9
8
7
6
5
4
Rounding
OPT
LP_OPT

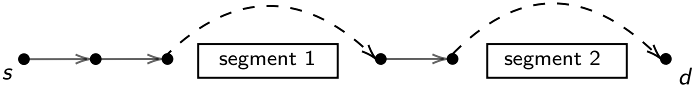
RLF Problem

Recall relaxed version: cycles allowed if not reachable from s.

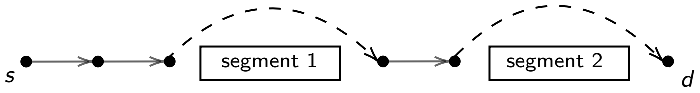
Can adapt previous LP approach easily. How about other methods?

RLF Other Approach

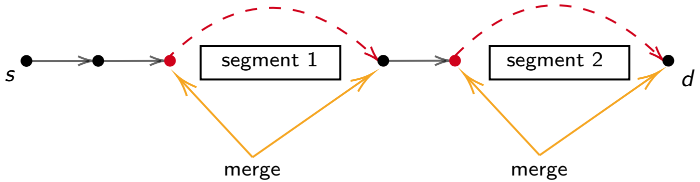
Forward path: s-d path containing old edges and forward new edges.



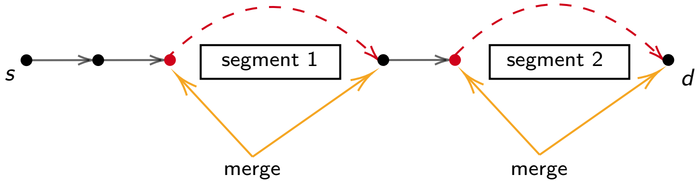
Reduced Instance

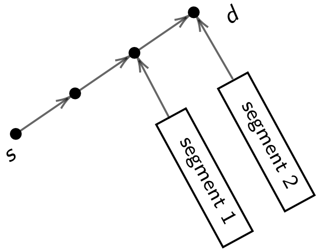


Update forward edges in one round.



Reduced Instance

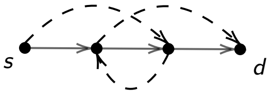




Can update everything jumped over in next round.

Reduced Instance

Obtain *Reduced Instance* with new $n := 1 +$ number old edges on path.



Reduced Instance

Peacock introduced in [Foerster et al., 2018].

Construct *forward path* in the following **greedy** way:

- Sort forward edges in descending order to form list L.
- $F := \emptyset$
- Iterate through L.
- Add $f \in L$ to F if $F \cup \{f\}$ can be used to form a *forward path*.

Reduced Instance

Peacock introduced in [Foerster et al., 2018].

Construct *forward path* in the following **greedy** way:

- Sort forward edges in descending order to form list L.
- $F := \emptyset$
- Iterate through L.
- Add $f \in L$ to F if $F \cup \{f\}$ can be used to form a *forward path*.

Lemma ([Foerster et al., 2018])

Reduced instance has $\leq 2n/3$ *nodes.*

$\implies$ Peacock solves any instance in $\mathcal{O}(\log n)$ rounds.

Lower Bound

Lemma ([Foerster et al., 2018])

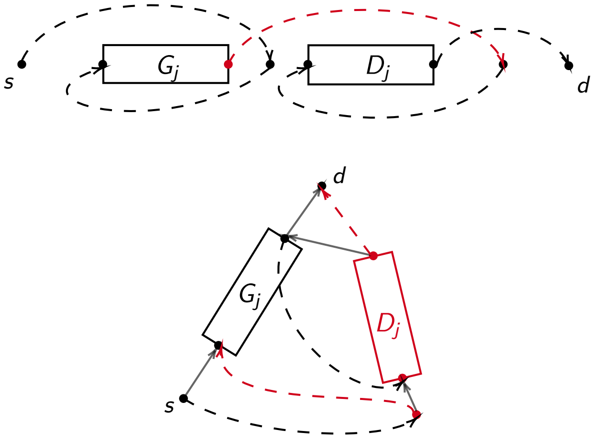
Graphs G_j exist with $n := 2^j$ nodes such that Peacock needs $\Omega(\log n)$ rounds.

Construction in [Foerster et al., 2018] contained mistake.
We fixed it.

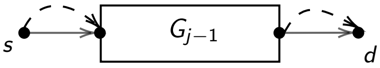
Theorem

Peacock is not an $o(\log n)$ approximation.

Proof by picture:



Peacock Approximation Factor



Instance is hard.

$\square$

Alternative Heuristic

Local Search:

- Start with some *forward path* P containing forward edges F. Solve reduced instance recursively in r rounds.

Alternative Heuristic

Local Search:

- Start with some *forward path* P containing forward edges F. Solve reduced instance recursively in r rounds.
- Select unused forward edge f and add it to solution. Obtain F'.

Alternative Heuristic

Local Search:

- Start with some *forward path* P containing forward edges F. Solve reduced instance recursively in r rounds.
- Select unused forward edge f and add it to solution. Obtain F'.
- Remove edges from F' is this is required to make it feasible.

Alternative Heuristic

Local Search:

- Start with some *forward path* P containing forward edges F. Solve reduced instance recursively in r rounds.
- Select unused forward edge f and add it to solution. Obtain F'.
- Remove edges from F' is this is required to make it feasible.
- Solve reduced instance recursively in r' rounds.

Alternative Heuristic

Local Search:

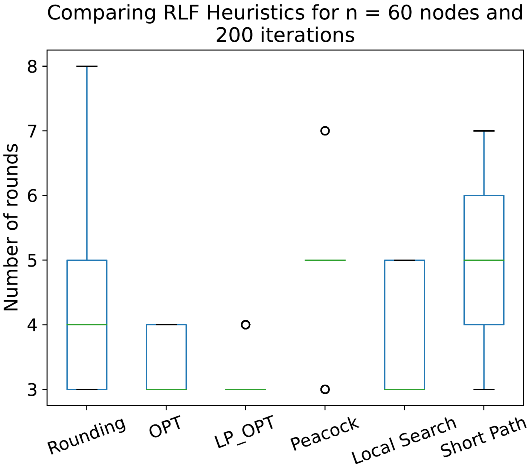
- Start with some *forward path* P containing forward edges F. Solve reduced instance recursively in r rounds.
- Select unused forward edge f and add it to solution. Obtain F'.
- Remove edges from F' is this is required to make it feasible.
- Solve reduced instance recursively in r' rounds.
- If $r' < r$: $F \leftarrow F', \ P \leftarrow P'$

Alternative Heuristic

Local Search:

- Start with some *forward path* P containing forward edges F. Solve reduced instance recursively in r rounds.
- Select unused forward edge f and add it to solution. Obtain F'.
- Remove edges from F' is this is required to make it feasible.
- Solve reduced instance recursively in r' rounds.
- If $r' < r$: $F \leftarrow F', \; P \leftarrow P'$
- Repeat until locally optimal.

Plots


Comparing RLF Heuristics for n = 60 nodes and 200 iterations
Number of rounds
8
7
6
5
4
3
Rounding
OPT
LP_OPT
Peacock
Local Search
Short Path

Conclusion

- LP Approach allows to solve SLF and RLF optimally very well in practice.
- LP Approach also good for approximation in SLF case. Open if there is any non-trivial approximation.
- Peacock is not $o(\log n)$ approximation. Open if there is such an approximation for RLF.
- Local Search best heuristic for RLF.

References I

Foerster, K.-T., Ludwig, A., Marcinkowski, J., and Schmid, S. (2018).
Loop-free route updates for software-defined networks.
IEEE/ACM Transactions on Networking, 26(1):328–341.