Coordinating SON Instances: Reinforcement Learning with Distributed Value Function

Ovidiu Iacoboaiea[†‡], Berna Sayrac[†], Sana Ben Jemaa[†], Pascal Bianchi[‡]

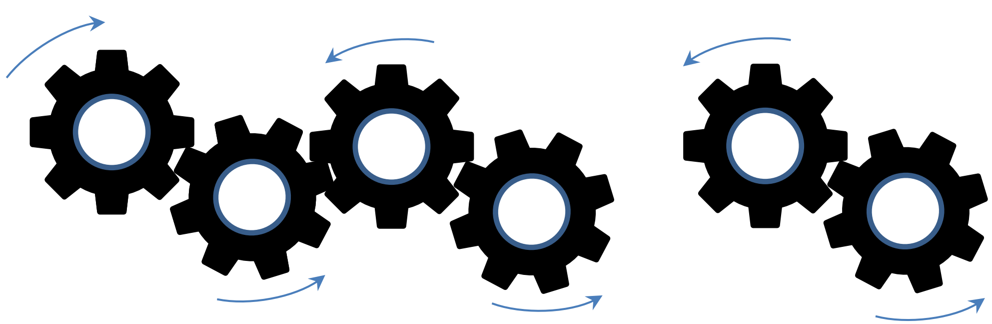
(†)Orange Labs, 38-40 rue du General Leclerc 92130, Issy les Moulineaux, France
(‡) Telecom ParisTech, 37 rue Dareau 75014, Paris, France

Presentation agenda:

- Introduction
- System Description: SONCO, network stability
- Reinforcement Learning
- State Aggregation
- Simulation Results
- Conclusions and Future Work

Introduction to SON & SON Coordination

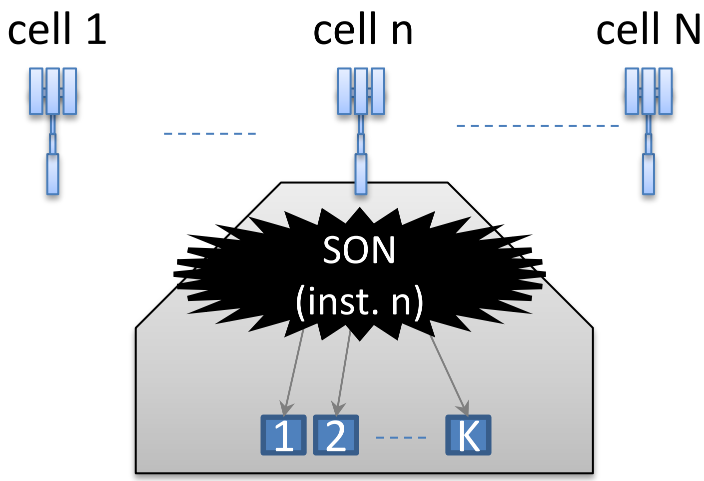
- SON functions are meant to automate network tuning (e.g. Mobility Load Balancing, (MLB),etc.) in order to reduce CAPEX and OPEX.
- A SON instance is a realization/instantiation of a SON function running on one (or several) cells.
- In a real network we may have several SON instances of the same or different SON functions, this can generate conflicts and instability.
- Therefore we need a SON COordinator (SONCO)



SON instance 1
(e.g. MLB instance on cell x)

SON instance 2
(e.g. MLB instance on cell y)

System description



We consider:

- N cells. (each sector constitutes a cell)
- 1 SON function (e.g. MLB*), black boxes
  - instantiated on every cell, i.e. N SON instances
- K parameters on each cell tuned by the SON functions (e.g. CIO*, HandOver Hysteresis)

❑ The network at time t:

$P_{t,n,k}$ - the parameter k on cell n

❑ The SON at time t:

$U_{t,n,k} \in [\pm 1; 0]$- the request of the SO instance n targeting $P_{t,n,k}$

- $U_{t,n,k} = -1$,$U_{t,n,k} = 1$ and $U_{t,n,k} = 0$ is a request to decrease, increase and maintain the value of the target parameter, respectively

❑ The SONCO at time t:

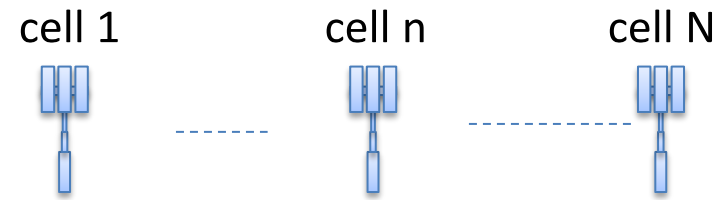
$A_{t,n,k} \in \{1,0\}$- the action of the SONCO

- if $A_{t,n,k} = 1$ / $A_{t,n,k} = 0$ means that we accept/deny the request $U_{t,n,k}$, i.e. $P_{t+1,n,k} = P_{t,n,k} + U_{t,n,k}\, A_{t,n,k}$

- targets to eliminate unnecessary parameter fluctuations

(*) MLB = Mobility Load Balancing; (*)

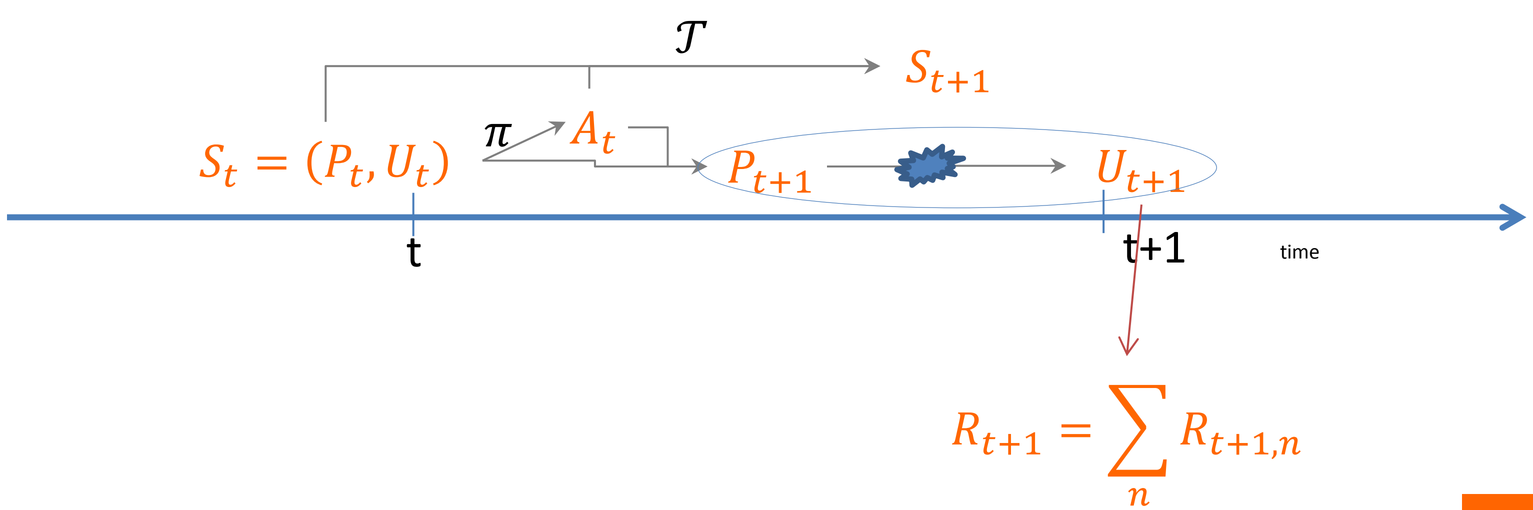
MDP formulation



- State: $S_t = (P_t, U_t)$

- Action: $A_t \in \{0,1\}^{NK}$

- Transition kernel:
  - $P_{t+1} = g(P_t, U_t, A_t)$ (where g is a deterministic function)
  - $U_{t+1} = h(P_{t+1}, \xi_{t+1})$, i.e. is a "random" function of P_{t+1}, and some noise ξ_{t+1}



Target: optimal policy, i.e. best A_t

- we define discounted sum reward (value function):

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_t \,|S_0 = s\right]$$

- the optimal policy π^* is the policy which is better or equal to all other policies:

$$V^{\pi^*}(s) \geq V^{\pi}(s), \qquad \forall s$$

- the optimal policy can be expressed as

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

where $Q^*(s,a)$ is the optimal action-value function:

$$Q^*(s,a) = \mathbb{E}_{\pi^*}\left[\sum_{t=0}^{\infty} \gamma^t R_t \,|S_0 = s, A_0 = a\right]$$

- As we only have partial knowledge of the transition kernel, we resort to Reinforcement Learning (RL) to **estimate** “on line” the Q^* function. For example we could use Q-learning.

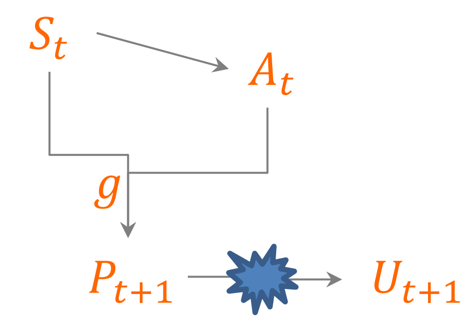
BUT: we have deal with the complexity issue

Towards a reduced complexity RL algorithm

Main idea : exploit the particular structure/features of the problem/model:

- Special structure of the transition kernel:

$$P_{t+1} = g(S_t, A_t)$$
$$U_{t+1} = h(P_{t+1}, \xi_{t+1})$$



- the reward:

$$R_{t+1} = \sum_{n \in \mathcal{N}} R_{t+1,n}$$

only depends on

The consequence is:

$$Q(s,a) = \sum_{n \in \mathcal{N}} W_n(p'), p' = g(s,a)$$

The complexity is reduced as now we can learn the W-function instead of the Q-function, (the domain of $(s,a) = ((p,u),a)$ is smaller than the domain of $g(s,a) = p$)

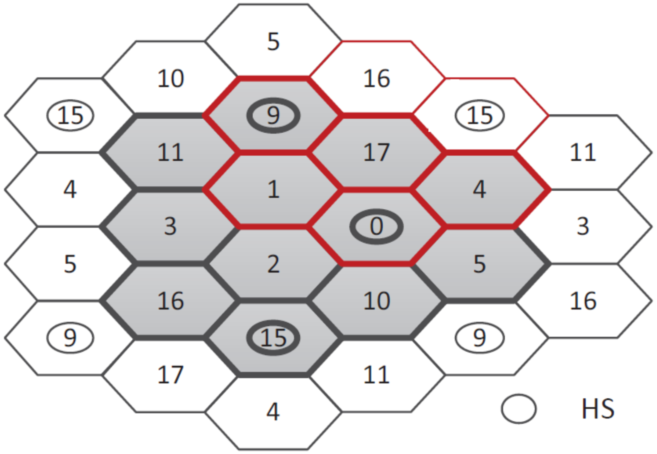
Still not enough, but…

- The complexity is still too large as the domain of $p' = g(s, a)$ scales exponentially with the number of cells.

➔ Use state aggregation to reduce complexity.

$$W_n(p) \approx \bar{W}_n(\bar{p}_n)$$

$\bar{p}_n$ contains the parameters of cell n and its neighbors.



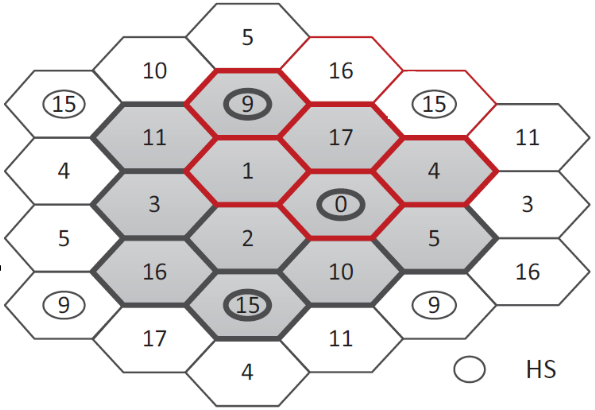
Application example

Some scenario details:

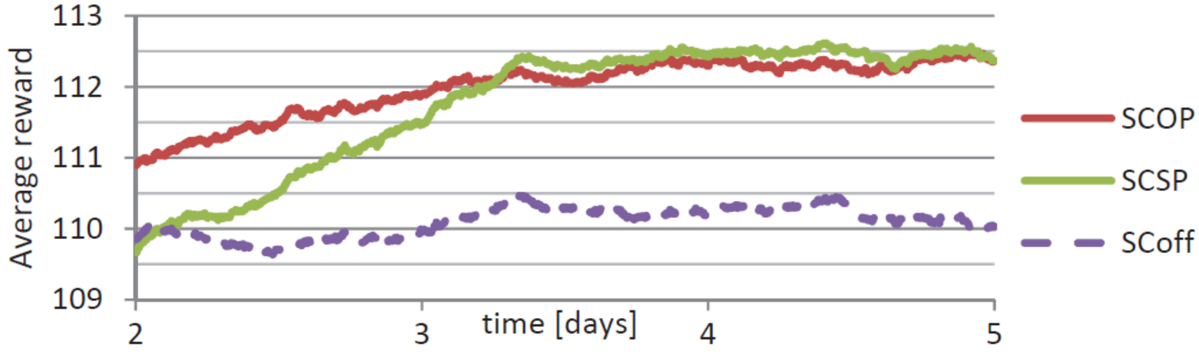
- ❑ 1 MLB instance on each and every cell tuning the CIO,
- ❑ we have an **instability** problem on the CIO,
- ❑ the reward is a sum of sub-rewards calculated per cell ➔ W_n $(n \in \mathcal{N})$,
- ❑ from $W_n(p)$ to $\bar{W}_n(\bar{p}_n)$: $\bar{p}_n$ contains the parameters of cell n and its neighbors,
- ❑ state space scales linearly with the no. of cells,
- ❑ reward: $R_{t,n} = 1{,}0 \cdot \mathbb{I}_{\{U_{t,n,CIO}=0\}} + 0{,}9 \cdot \mathbb{I}_{\{U_{t,n,CIO}=1\}}$,

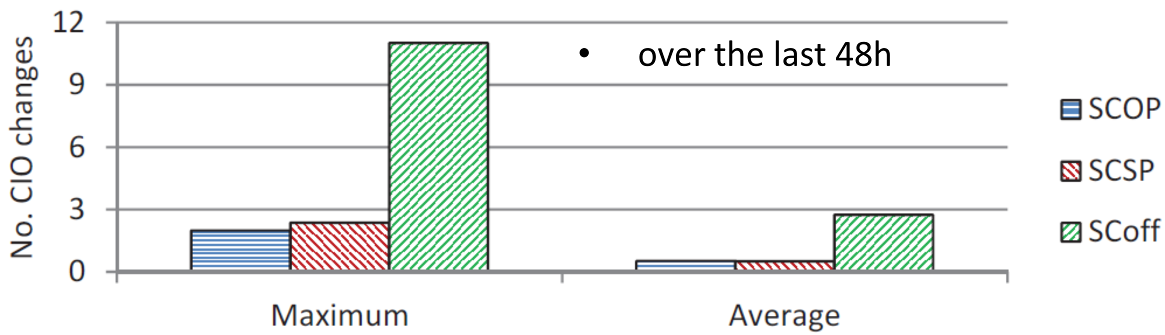
i.e. no reward when the request is “off-load”,

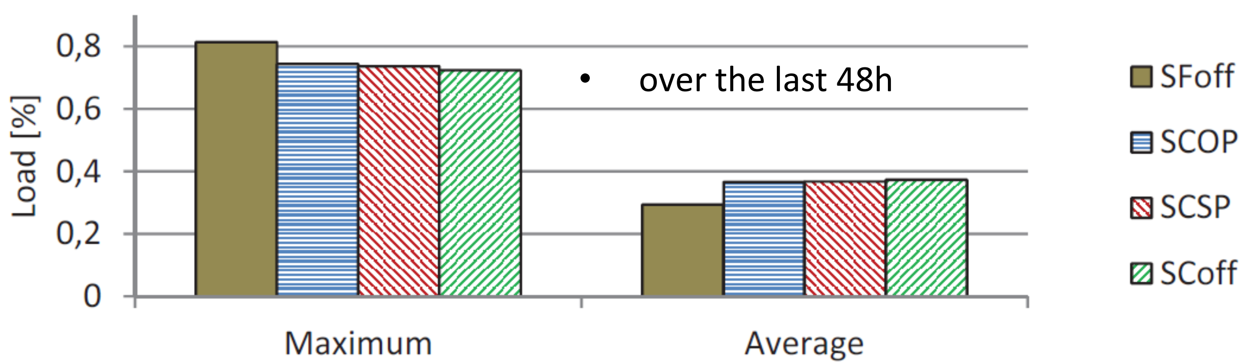
to say that we are unhappy when the cell is overloaded.



Simulation Results







4 scenarios:

1. **S**on**C**o **O**ptimal **P**olicy: uses and estimates W_n,

2. **S**on**C**o **S**ub-optimal **P**olicy: uses and estimates $\overline{W}_n$,

3. **S**on**C**o **off**

4. **S**on **F**unctions **off**

- stationary traffic

Conclusion and future work

- with RL we can improve the network stability, i.e. reduce the unnecessary parameter changes, without affecting the target of the SON (the load balancing)
- the solutions state space scales linearly with the number of cells

Future work:

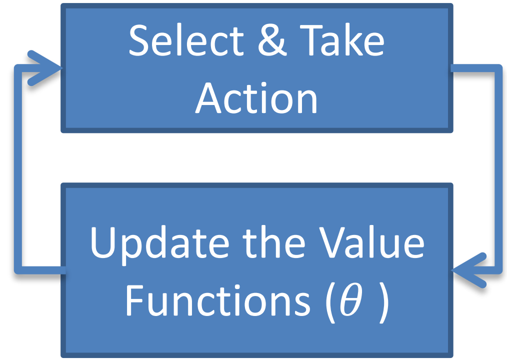
- analyzing **tracking** capability of the algorithm,
- **HetNet scenarios,**

Questions ?

ovidiu.iacoboaiea@orange.com

SONCO: Algorithm

So this is what has to be implemented:



Algorithm 1 SONCO

Function Init :

$\forall i \in \mathcal{I}_m$, $\forall j \in \mathcal{J}_i$ initialize $\theta_{i,j} = 0$

Function SONCO :

Observe current parameter configurations $p_{\mathcal{O}_m}$ and update requests $u_{\mathcal{O}_m,\mathcal{Z}}$, calculate regret $r_{\mathcal{I}_m}$,

  Compute $a^*_{\breve{\mathcal{O}}_m} = \arg\min_{a_{\mathcal{O}_m} \in \mathcal{A}_{\mathcal{O}_m}} \sum_{i \in \mathcal{I}_m} \bar{W}_i^* \left(\bar{g}_i\left(\left(\bar{p}_i, \bar{u}_i\right), \bar{a}_i\right)\right)$

  Compute (and overwrite) $a^*_{\breve{\mathcal{O}}_m} = \left(\mathbb{I}_{\{\exists z \in \mathcal{Z} \text{ s.t. } u_{n,k,z} > 0\}} - \mathbb{I}_{\{\exists z \in \mathcal{Z} \text{ s.t. } u_{n,k,z} < 0\}}\right)_{(n,k) \in \breve{\mathcal{O}}_m}$

  Compute $p^*_{\breve{\mathcal{O}}_m} = g_m\left(\left(p_{\mathcal{O}_m}, u_{\mathcal{O}_m,\mathcal{Z}}\right), a^*_{\breve{\mathcal{O}}_m}\right)$

  For all $i \in \mathcal{I}_m$,

    For all $j \in \mathcal{J}_i$,

      $\theta_{t+1,i,j} = \theta_{t,i,j} + \alpha\left[r_i + \gamma \sum_{i \in \mathcal{I}_m} \bar{W}_i^*\left(\bar{p}_i^*\right) - \sum_{i \in \mathcal{I}_m} \bar{W}_i^*\left(\bar{p}_i\right)\right] F_{i,j}\left(\bar{p}_i\right)$

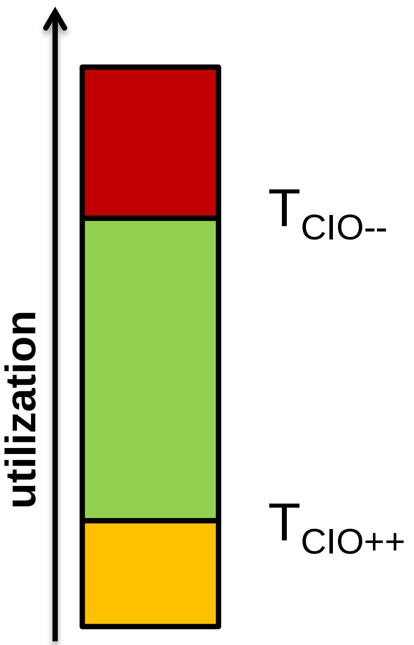
Choose action $a_{\mathcal{O}_m}$ using the ϵ-greedy policy $\pi_m\left(\left(p_{\mathcal{O}_m}, u_{\mathcal{O}_m,\mathcal{Z}}\right), \{a_{\mathcal{O}_m}\}\right) = \left((1-\epsilon)\,\mathbb{I}_{\{a_{\mathcal{O}_m} = a^*_{\breve{\mathcal{O}}_m}\}} + \frac{\epsilon}{3^{NK}}\right)\mathbb{I}_{\{a_{\breve{\mathcal{O}}_m} = a^*_{\breve{\mathcal{O}}_m}\}}$,

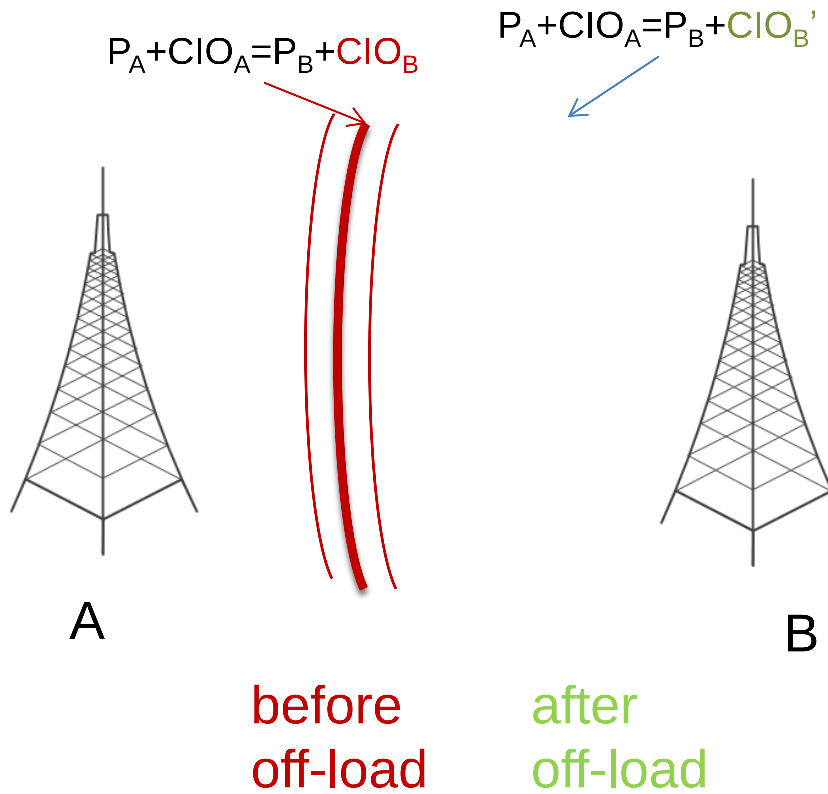
Take action $a_{\mathcal{O}_m}$.

System Description: MLB

Looking at base station B:

- if(utilization > T_{CIO--}) ➔ CIO_B--
- if(utilization < T_{CIO++}) ➔ CIO_B++ (for going back to the default configuration)
- else ➔ Relax (void)





CIO range (-12dB:0dB/ step-size 3dB)