S-TaLiRo with Stochastic Optimization with Adaptive Restart (SOAR) Optimization Framework

Logan Mathesen, Giulia Pedrielli

April 2019

1 Overview of SOAR in S-TaLiRo

The Stochastic Optimization with Adaptive Restart (SOAR) optimization framework Mathesen and Pedrielli [2019] has been implemented as an option for stochastic optimizer selection in the S-TaLiRo tool. The S-TaliRo system falsification tool is a publicly available Matlab toolbox, and can be downloaded at `https://sites.google.com/a/asu.edu/s-taliro/s-taliro`.

In general, the SOAR framework consists of four components: 1) a global model, 2) a sampling score that, when assessed over the global model, determines local search starting location, 3) a local search routine, and 4) an indicator for local search rate of progress, to facilitate local search termination/restart. Currently, there are four alternative implementations of the SOAR framework in this repository. All four make use of an Ordinary Kriging Gaussian process for the global modeling component. Moreover, all four implementations have the capability of using either the standard *expected improvement* sampling score Jones et al. [1998], or the SOAR tailored *crowded expected improvement* sampling score Mathesen and Pedrielli [2019]. The primary differences in the implementations lies in the local search component and the mechanism for facilitating dynamic restarts of that local search.

The four implementations included in this repository are:

1. SOAR with Finite Differencing
2. SOAR with SPSA
3. SOAR with 2SPSA
4. SOAR with Local Gaussian Processes

Note, initial experimentation has shown SOAR with local Gaussian processes (Section 4) to have the most robust performance capabilities with respect to the falsification problem in arbitrary dimension Mathesen et al. [2019]. The other three implementations include the explicit formulation and solving of a trust region subproblem that makes use of second order Taylor expansions around the current local centroid; one of these uses finite differencing to approximate the second order model, while the other two make use of a simultaneous perturbation stochastic approximation to estimate the gradient and Hessian at the current centroid.

The remainder of this document briefly explains each of the four algorithms and the minimum syntax to select the SOAR based algorithm of choice as the stochastic optimizer within S-TaLiRo. For more information on how to install the SOAR optimizers in the S-TaLiRO tool, and an example file where each of the four optimizers are called over the Navigation model tested in Mathesen et al. [2019], please refer to the README.md file in this repository.

2 SOAR with Finite Differencing (‘SOAR_Taliro_FiniteDiff’)

This optimizer can be called within the S-TaLiRo tool by setting the following option value:

```
opt = staliro_options();
opt.optimization_solver = 'SOAR_Taliro_FiniteDiff';
```

This algorithm mirrors the implementation presented in Mathesen and Pedrielli [2019]. Making use of an explicit trust region subproblem formulation where the gradient and Hessian matrix are estimated through a finite differencing approach. See Nocedal and Wright [2006] (Chapter 4) for a review on trust region subproblem formulation and solving, and Spall [2005] (Chapter 6) for review of finite differencing methods. As this is a gradient based approach it is generally effective over differentiable surfaces, and due to the need to take $\frac{d^2+3d}{2}$ samples to estimate the Hessian, it works best in moderate to low dimensional problems. Note, that the default setting is to use the crowded expected improvement sampling for trust region centroid restart location, though this can be changed with the command:

```
opt.optim_params.crowded_EI_flag = 0;
```

Moreover, all optimization parameters that are available to be edited (as done in the above command), and their default values, can be viewed in ‘SOAR_Taliro_FiniteDiff_parameters,’ in the optimization folder.

3 SOAR with (2)SPSA (’‘SOAR_Taliro_SPSA’ and ‘SOAR_Taliro_2SPSA’)

These optimizers can be called within the S-TaLiRo tool by setting the respective option value:

```
opt = staliro_options();
opt.optimization_solver = 'SOAR_Taliro_SPSA';
```

or for 2SPSA,

```
opt = staliro_options();
opt.optimization_solver = 'SOAR_Taliro_2SPSA';
```

These algorithms are nearly identical to the above ’Finite Differencing’ approach, but make a critical choice to reduce the number of observations necessary to estimate a gradient and Hessian via a simultaneous perturbation stochastic approximation (SPSA) approach. For SPSA, two samples symettic samples about the current centroid are taken, regardless of problem dimension, to approximate the current gradient. 2SPSA takes this a step further and takes two additional samples and uses this information to estimate a Hessian as well as a gradient at the centroid. These approximations have been proven to be asymptotically correct and are explained at length in Spall [2005] in Chapter 7. In practice this method is able to construct a gradient much quicker, although like the finite differencing approach, the basic premise of gradient based optimizations fail with non-differentiable responses.

4 SOAR with Local Gaussian Processes ('SOAR_Taliro_LocalGPs')

This optimizer can be called within the S-TaLiRo tool by setting the option value:

```
opt = staliro_options();
opt.optimization_solver = 'SOAR_Taliro_LocalGPs';
```

This implementation makes use of a trust region bounding mechanism, but does not explicitly formulate and solve a trust region subproblem. The aim here is to reduce the size of the decision space, such that the Gaussian process can more easily recover and model the true underlying function. The local Gaussian processes can then be much more effective in their exploitative behavior, while not being entirely lost when a function loses differentiability- as in the case of the gradient based local searches.

Once a centroid and trust region size have been established, the trust region itself can be well defined. Within this bounded trust region area a random sampling is taken and a local Gaussian process model is fit. From this local Gaussian process, the location (within the current trust region) with maximum expected improvement is proposed as the next candidate centroid. Identical to the previous three methods, a ratio comparison test is then executed and the trust region is translated and scaled appropriately. An additional random sampling is then taken within the new trust region area such that there are sufficient samples to estimate an effective Gaussian process. Further reading on the exact algorithm implemented can be found in Mathesen et al. [2019].

References

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Logan Mathesen and Giulia Pedrielli. Stochastic optimization with adaptive restart: A framework for integrated local and global learning. *submitted for publication to the Journal of Global Optimization*, 2019.

Logan Mathesen, Shakiba Yaghoubi, Giulia Pedrielli, and Georgios Fainekos. Falsification of cyber-physical systems with robustness uncertainty quantification through stochastic optimization with adaptive restart. *submitted for publication to IEEE International Conference on Automation Science and Engineering*, 2019.

Jorge Nocedal and Stephen J Wright. Trust-region methods. *Numerical Optimization*, pages 66–100, 2006.

James C Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*, volume 65. John Wiley & Sons, 2005.