WORHP MULTI-CORE INTERFACE, PARALLELISATION APPROACHES FOR AN NLP SOLVER

Sören Geffken, Christof Büskens

Universität Bremen, Bremen, Germany

ABSTRACT

The goal of this paper is to present current research activities aiming at improved efficiency and stability within the ESA NLP-Solver WORHP. It is designed to solve high dimensional sparse non-linear optimisation problems.
The underlying SQP method is inherently sequential, therefore parallelism cannot be exploited straightforwardly.
In order to obtain the best solver performance the parameter configuration should be adapted accordingly for every problem itself. An approach running several solver instances using different parameter sets in parallel has been developed and proven highly beneficial on a given set of problems. The First-Across-The-Line approach stops all instances when the first local optimum has been found, thus improving the solver's speed and stability as well. Furthermore, the approach allows the user to experiment with specialised algorithms within the optimisation as the threads using basic parameter settings serve as safeguards guaranteeing the solver to converge as usual.
In order to improve the solver's efficiency for one special problem, the new operational mode can be used to automatically attune the solver's parameters accordingly. Again the solver is started with several instances at once, but this time the Best-Of-All mode is used in order to obtain the best local optimum and the corresponding parameter settings. Additionally, the mode enables users to perform parameter sweeps to further improve the solver's configuration.
The results presented show the improvement of the solver's performance on the state-of-the-art CUTEst test set that has been solved faster and with more optimal solutions found compared to the traditional single-core approach. The parameter attunement is applied to specific single problems from the test set as well.

Index Terms— Non-linear Programming, Sequential-Quadratic-Programming, NLP solver

1. INTRODUCTION

The numerical solution of non-linear optimisation problems is a crucial technology for many engineering tasks like parameter identification or optimal control problems. Therefore, the efficiency of the used methods to solve such problems is substantial for those research fields. The mathematical theory of these problems is being researched for several years and different methods like sequential quadratic programming and interior-point methods have been developed to solve them. As those methods are based on Newton's method, they are inherently sequential and offer only limited possibilities to exploit parallelism. In this paper we present an approach to benefit from multi-core processors as those are available in nearly every computer today. We use the NLP-solver WORHP, that has been developed for some years and offers a multitude of specialisations of the basic SQP algorithm. The parallelism is used to run several algorithm specialisations in parallel and take benefit from the multi-core processors. Different application scenarios for the multi-core interface aiming at speed improvements, increased robustness or user comfort are presented.
This paper is organised as follows. In Section 2 the fundamental mathematical theory for non-linear programming is briefly presented. Section 3 describes the technical details such as thread management and class hierarchy of the implementation. The different application scenarios and the corresponding operational modes are presented in Section 4. Numerical results using the new interface to the state-of-the-art CUTEst test set for non-linear optimisation are shown in Section 5.

2. FUNDAMENTALS

Let $x \in \mathbb{R}^{N_x}$ be the vector of optimisation variables and $f : \mathbb{R}^{N_x} \to \mathbb{R}$ denote the objective function, $g : \mathbb{R}^{N_x} \to \mathbb{R}^{N_g}$ denote general non-linear inequality constraints and $h : \mathbb{R}^{N_x} \to \mathbb{R}^{N_h}$ general non-linear equality constraints. Furthermore, let f and g be twice continuously differentiable. Then

$$\begin{aligned} \min_{x \in \mathbb{R}^{N_x}} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0 \\ & h(x) = 0 \end{aligned} \tag{1}$$

is called a *non-linear program*. The goal of non-linear optimisation is to find a local minimum and corresponding Lagrangian multipliers (x^*, λ^*) such that first and second order conditions are fulfilled.

The NLP solver WORHP [1] is designed to solve such problems even with possibly huge problem dimensions. In order to handle problems with up to 10^7 optimisation variables and up to 10^9 different general constraints the sparsity structure of the derivatives must be taken into account.
Within the solver we apply a sequential quadratic programming algorithm (SQP). The basic principle is to approximate the non-linear problem by a series of quadratic subproblems of the form

$$\begin{array}{ll} \min\limits_{d \in \mathbb{R}^{N_x}} & \frac{1}{2} d^\top \nabla^2_{xx} L(x^{[k]}, \lambda^{[k]}, \mu^{[k]}) d + \nabla_x f(x^{[k]})^\top d \\ \text{unter} & g(x^{[k]}) + \nabla_x g(x^{[k]})^\top d \leq 0 \\ & h(x^{[k]}) + \nabla_x h(x^{[k]})^\top d = 0 \end{array}$$

that are solved using an interior point method based on Mehrotra's predictor corrector method for linear programming [2]. The Hessian of the Lagrangian $L(x^{[k]}, \lambda^{[k]}, \mu[k])$ can be given analytically, computed by finite differences or composed either using classic or sparse BFGS algorithms.
Specialised sparsity exploiting algorithms are applied to solve these subproblems and especially the resulting linear equation systems are handled efficiently.
The optimal solutions $d^{[k]}$ of those subproblems are used as search directions in order to generate a converging sequence of points $x^{[k]}, k = 0, 1, 2, \ldots$ towards a local minimum of problem (1). The search direction $d^{[k]}$ in the k-th iteration is used by the solver within its linesearch module to obtain the next iterate by application of the formula

$$x^{[k+1]} = x^{[k]} + \alpha^{[k]} d^{[k]}.$$

The linesearch module offers different evaluation strategies for the computed search directions. Merit function as well as filter based approaches are implemented within the solver and can be chosen by the user.
In Figure 2 a schematic overview of an NLP solver using sequential quadratic programming is given. The main steps within the algorithm are shown. Within the loop visible in the figure users are able to control multiple aspects of the algorithms used within WORHP. These parameters control different derivative computation methods, variations of the used linesearch algorithm, special features like feasibility refinement, the configuration of the integrated solver for quadratic programs or a great number of floating point parameters. Thus, the user is able to highly customise the solver to his special needs.
Unfortunately, most of these parameters require a detailed knowledge of the implementation and are mostly used with their default values, thus denying the users to take benefit. The WORHP Multi-Core Interface presented in this paper allows the user to test and even heuristically identify those solver parameters for his special problem.

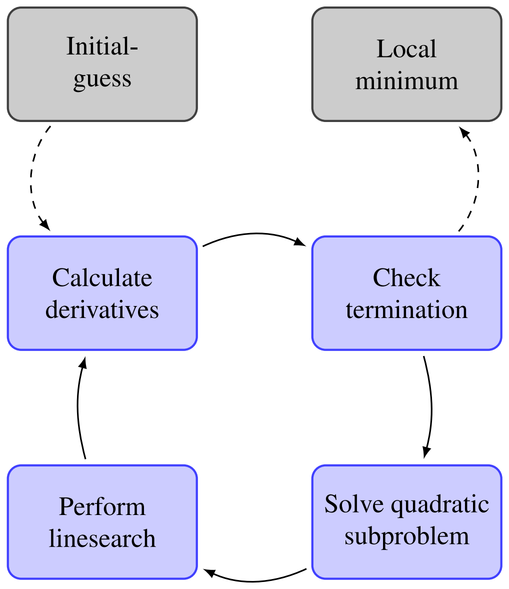

Fig. 1. Schematic view of NLP solver using SQP method

3. TECHNICAL IMPLEMENTATION

The default WORHP interface is based on the reverse-communication paradigm [3]. The basic idea of this interface is, that the user implements his model within the reverse-communication loop of WORHP, that can be implemented when the solver library is linked against the user code. This circumstance forbids parallelisation of objective and constraint function evaluations in the first place, because the loop is inherently sequential and duplication of these evaluations would mean rigorous parallelisation of user code. As that is not known to the programmers of the WORHP interface and possibly not thread safe this might lead to possible data races and thus lead to undefined behaviour. In order to overcome these technical issues the WORHP multi-core interface is introduced.
The WORHP multi-core interface consists of two implementation layers. The upper layer is the *NLPSolver* class. Within the solver class the automatic parameter variation module is included. The *NLPSolver* class is responsible to handle all thread related tasks. The lower layer of the interface contains the implementations of the *NLPProblem* given by the user.
To keep the implementation effort for the user as small as possible we chose an object oriented approach for the new interface. Users implement the abstract class *NLPProblem* that requires the user to override the problem related functions and derivatives as depicted in Listing 1.
At least objective and constraint functions always have to be implemented. If the derivatives are not available the corresponding functions must be overridden by empty functions to implement the NLPProblem interface and the correspond-

ing parameters must be set accordingly to allow WORHP to compute the necessary derivatives by finite differences or use Hessian approximations like BFGS matrices as second order derivatives.

Listing 1. Interface for problem functions

```
class NLPProblem {
public :
/* Prototype for evaluation of
 * objective function */
virtual void evalF(double *f)       = 0;
/* Prototype for evaluation of
 * constraints function */
virtual void evalG(double *G)       = 0;
/* Prototype for evaluation of
 * objective gradient */
virtual void evalDF(double *dfVal) = 0;
/* Prototype for evaluation of
 * jacobian of the constraints */
virtual void evalDG(double *dgVal) = 0;
/* Prototype for evaluation of
 * hessian of the lagrangian */
virtual void evalHM(double *hmVal) = 0;
...
};
```

One instance of the concrete user problem is created for each thread that will be used. All problems to be solved are then added to the queue within the solver class. The user specifies a certain number of slots available for the multi-core interface. The number of slots should not exceed the number of available cores on the system. Additionally, one slot should be reserved for the solver class.

The *NLPSolver* class launches the specified number of instances of WORHP in order to solve the user problem with different parameter settings and afterwards oversees the progress of all running threads, processes output and controls the workflow according to the chosen operational mode. The described workflow is sketched in Figure 2. While the worker threads solve their corresponding problems, on each iteration a notice containing information about the iteration progress, current objective value and constraint values is send to the solver class. As those notices are send by multiple threads in parallel a lockfree queue is used as multi producer single consumer container. Similar to an event queue of a graphical user interface the solver class waits for new notices and processes them in order to perform several tasks that are not thread safe like providing output to the user, launching new threads if slots have become available or handle the final results.

4. OPERATIONAL MODES

The multi-core interface can be used to achieve multiple results. As mentioned above, the solver can be greatly customised to the users need, but the default parameter settings have been configured in order to perform best on the CUTEst

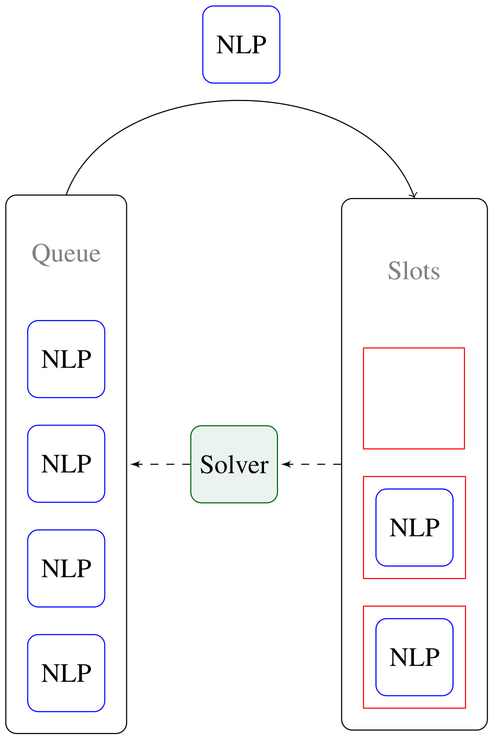

Fig. 2. Thread organisation through NLPSolver class

and COPS test set in total. Thus, the settings are not specialised for any special kind of problem, but to improve the performance of the solver on the majority of the problems. In their paper about the free lunch theorems Wolpert and Macready [4] point out, that adapting solver parameters for one class of problems always leads to a loss of efficiency for another class. At this point the multi-core interface comes into play and allows the user to apply different parameter settings in parallel to his special problem formulation, thus aiming to increase speed as well as robustness of the solver.

Currently WORHP offers 175 parameters, thus users struggle to identify relevant ones to adapt to their problem. Therefore, in order to make the usage of the Multi-Core interface most practical for the user, the interface offers a preconfigured set of parameters that can be altered by the different threads. These different settings have been proven highly beneficial in various situations in order to improve the solvers performance with respect to speed, robustness and quality of the solution.

Using automatic code generation realised by a perl script, parameter variations can be specified and are automatically incorporated into the parameter variation module of the solver class. In Listing 2 a short snippet of the default possible settings script is shown.

Listing 2. Possible settings in script language for automatic code generation of parameter variations

```
//###################################
// This file is organised as follows:
// (Setting) (optional: Requirement)
//###################################
## General Method Parameters
(par->NLPmethod = 3) (par->qp.ipLsMethod == 1)
(par->NLPmethod = 1) ()
## Number of Relaxation Variables
(par->MoreRelax = false) (opt->m > 0)
(par->MoreRelax = true) (opt->m > 0)
## ScaledKKT for Thoroughness
(par->ScaledKKT = true) ()
(par->ScaledKKT = false) ()
```

The script language is organised as follows. Parameter settings that can be exchanged against each other are grouped together and are separated by lines starting with ##. Using a line starting with ## a new sequence of exchangeable parameter settings is commenced. Each line containing a setting consists of two bracketed parts. The first part contains the parameter setting to be set and the second part the requirements that must be met by the problem and the solver in order to apply the setting in the first part. Multiple requirements or settings can be paired within the brackets using the boolean `AND` operator &&, thus BFGS matrices as second derivatives can be activated and a specific method chosen at the same time.

The first block of exchangeable settings, commented with *General Method Parameters* changes the linesearch method from filter (`NLPmethod = 3`) to merit function (`NLPMethod` = 1). But the usage of the filter is only possible, if the linear solver is able to compute the inertia of the KKT matrix, thus the linear solver method must be chosen correctly beforehand (`ipLsMethod` = 1).

The second block alters the number of used relaxation variables based on the ideas described by Powell [5]. In order to relax constraints, obviously the problem must be constrained (`opt->m` > 0).

If no requirement is necessary the second bracket will simply be left empty as shown in the third block.

In the following sections we present different approaches to use the capabilities of this new multi-core interface.

4.1. First-Across-The-Line

The principle idea of the First-Across-The-Line (FATL) approach is to run several instances of WORHP in parallel in order to obtain an optimal solution as fast as possible. These instances will be configured with differing parameter sets, with the assumption that this will enable one instance to outperform the others. Once an optimal solution is found, all other instances are terminated, and the solution is returned to the user. Neglecting the slight technical overhead for thread management, startup etc., this approach is never slower than a usual sequential run, and has the potential to yield remarkable speed-up.

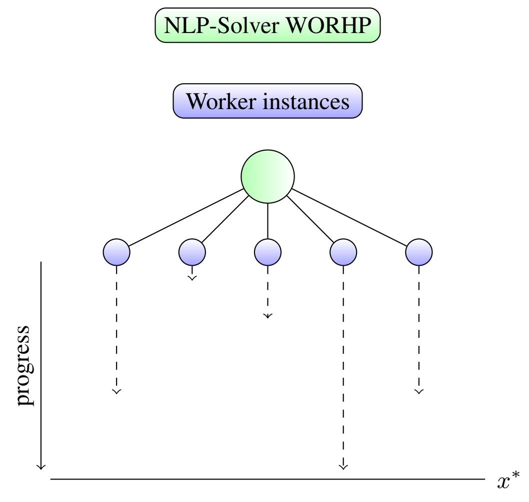

Fig. 3. First-across-the-line: The solver launches multiple threads towards optimal solution x^*

The FATL-approach basically wants the solver to work on all threads until one thread finds an optimal solution. WORHP distinguishes between optimal solutions, acceptable solutions and unsuccessful terminations. Only in the first case the multi-core interface should stop working, whereas in the other cases the remaining threads should continue their work. Figure 3 shows the workflow of the First-Across-The-Line approach. In the figure one worker instance reached at a local optimum and thus the remaining worker threads are terminated.

4.2. Best-Of-All

WORHP is designed to find local minima of the given problem. Depending on the problem structure the solver is able to find different local minima with different parameter settings. The Best-Of-All mode is designed to overcome this caveat. Again multiple instances of the solver are launched in parallel, but in this case the thread handling solver class waits until all threads have finished their job and compares the achieved results. Thus, the best obtained local minimum is returned to the user. This approach obviously does not focus on speed but on thoroughness.

4.3. Parameter Identification

The usage of an NLP solver is often an iterative process for the user. An optimisation run is performed and the result analysed. Afterwards the underlying mathematical model, constraints or the objective function are adapted according to the results achieved. Having adapted everything the solver is run again in order to improve the previous result and evaluate the performed changes. This loop is often processed multiple

times.
Therefore, the application of an optimisation tool is performed often. Still the problem to be solved varies only sligthly from run to run. This motivates the application of the *Parameter Identification* operational mode. The user can supply multiple parameters to be tested within this mode using the automatic code generation explained in the beginning of this section.
Similar to the Best-Of-All mode all threads will be finished first and then the results will be compared. Taking different criteria into account, parameter suggestions for the solver will be generated for the user.
A different approach on parameter tuning specifically focussed on WORHP can be found in the work of Wassel [3]. The predecessor of the parameter identification operational mode, a parallelisation approach on process level rather than on thread level is discussed there. The method described there was used to identify the currently used default parameters and focusses on parameter tuning over a given set of test examples. The application of the new parameter identification mode is realised by switching simply the operational mode of the multi-core interface, in contrast to the old tuning script that required some coding effort and was only used for internal parameter configurations of the solver.

5. NUMERICAL RESULTS

The proposed parallelisation approaches for the NLP solver WORHP were tested using the state-of-the-art CUTEst test set from Gould, Orban and Toint [6] that is specifically designed to be thread safe in comparison to its predecessor the CUTEr test set, that was part of the default testing environment of WORHP already. In this section the First-Across-The-Line mode will be applied to the CUTEst test set and the results are compared to the current version of WORHPs basic single thread interface. Furthermore, the parameter identification mode will be used to show the potential of parameter variations on the specific examples.
The CUTEst test set consists of 1149 examples. During our tests we faced some technical issues with 17 examples, therefore the following tests will be performed on the subset of the remaining 1132 examples. The baseline for the multi-core interface using the First-Across-The-Line approach is given by the results achieved by the single threaded interface of WORHP. The solver is run with its default configuration. Termination criteria are an optimality and feasibility tolerance of 10^{-6} each and the complementarity tolerance is set to 10^{-3}. If one of the first two measures is only fulfilled with a tolerance of 10^{-3} the minimum is regarded is acceptable. Analytic derivatives are given and used for all problems. The results are presented in Table 1.
We considered four available cores as a default configuration in most desktop computers and therefore have chosen four parameter settings for the First-Across-The-Line approach in order to improve the results on the CUTEst test set. The first thread runs with the same configuration as the single threaded WORHP. The second thread activates the additional feasibility refinement feature, the third thread uses a merit function approach instead of the filter during linesearch and the fourth thread will use BFGS matrices instead of the analytic Hessian matrix. In order to identify four beneficial parameter settings some prior knowledge of the solver is required. To overcome this issue for unexperienced users the parameter identification mode will be explained later. Table 2 summarises the used settings in the FATL approach.
The multi-core interface was able to improve the solvers performance significantly. 1084 examples were solved to optimal tolerances, thus 19 results were improved. The number of unsuccessful runs was reduced by 8 and the number of acceptable solutions by 11 examples. Consequently, the quality of the found local minima was increased from acceptable to optimal tolerance in these cases. The results are shown and compared to the single thread results in Table 3.
In order to yield the best results the chosen settings should enable different solver instances to perform best. Table 4 shows the distribution of the first terminating threads with the given settings. The First-Across-The-Line approach has

	Quantity	Percent
Optimal	1065	94.08
Acceptable	19	1.68
Unsuccessful	48	4.24
Total	1132	100

Table 1. Results of single threaded WORHP on subset of CUTEst test set

Setting	**Thr. 1**	**Thr. 2**	**Thr. 3**	**Thr. 4**
Feas. Ref.	Off	On	Off	Off
Hessian	Analytic	Analytic	Analytic	BFGS
Method	Filter	Filter	Merit fun.	Filter

Table 2. Used settings in four thread configuration of First-Across-The-Line mode

	Quantity	Change	Percent	Change
Optimal	1084	$+19$	95.76	$+1.68$
Acceptable	8	-11	0.71	-0.97
Unsuccessful	40	-8	3.53	-0.71
Total	1132		100	

Table 3. Results of multi threaded WORHP on subset of CUTEst test set using First-Across-The-Line approach in comparison to single threaded results

Thread 1	Thread 2	Thread 3	Thread 4
290	236	244	362

Table 4. Number of problems solved fastest by each thread

two different goals. On the one hand the parallel usage of multiple parameter settings is aimed at increasing the quality of the achieved local minima or finding one at all, as shown above. On the other hand the interface is applied in order to yield a speed up of the optimisation run as all threads are terminated upon successful termination of the first thread. The local minima of the single- and multi-threaded WORHP were compared for all examples within the CUTEst test set and for the following timing analysis only optimisation runs finding the same local minimum in both cases will be considered. Deviations of 2 percent of the final objective value were chosen as threshold to determine two results as the same local minimum. From the initial 1132 examples, the single- and multi-thread interface find the same local minimum in 784 cases. The elapsed time of the parallel runs is calculated as user time, not taking cumulative clocks of parallel threads into account. Thus, the FATL approach solves the complete test set in 207 minutes, while the single thread WORHP requires 279 minutes. Therefore, a speed-up factor of 1.3478 is achieved. The results are summarised in Table 5.

The parameter identification mode is applied to the *DTOC6*

Multi Thread (FATL)	Single Thread	Speed-up factor
207 min.	279 min.	1.3478

Table 5. Timing results and speed-up factor for First-Across-The-Line approach in comparison to single threaded WORHP

example. This test problem consists of 10001 variables and 5000 constraints. The multi-core interface is configured to vary the linesearch method, use analytic Hessian or a BFGS like alternative and activate the feasibility refinement. Thus, 8 different settings can be combined. Table 6 gives a short overview on this configuration. The user can determine a

Setting	**Value 1**	**Value 2**
Linesearch method	Filter	Merit function
Feasibility Refinement	On	Off
Hessian	Analytic	BFGS

Table 6. Possible values for different settings and any combination thereof for parameter identification mode

number of simultaneously runnable threads according to his system specification. All problems assigned to the solver are solved. Upon termination of all tasks, the solver evaluates the results with respect to different criteria. Currently the parameter identification mode is configured to provide the configuration with the least number of required major iterations which correlates directly to the number of derivative evaluations necessary and quadratic subproblems to be solved. Furthermore, as WORHP is a local optimisation software the settings leading to the best objective function value are determined by the parameter identification mode.

Listing 3. Snipppet of results from parameter identification mode for DTOC6 example

```
##        WMCI parameter identification mode          ##
### Least major iterations required by these settings ###

Thread 2:
---------
par->NLPmethod = 1;
par->UserHM = true;
par->RefineFeasibility = 2;

Final values after iteration 7:
Final objective value ............   1.4290199970E+05
Final constraint violation .......   1.1559653998E-07
Final KKT conditions .............   9.2700494022E-09
Successful termination: Optimal Solution Found.
------------------------------------------------------------
 ### Best objective value achieved by these settings ###

Thread 1:
---------
par->NLPmethod = 3;
par->UserHM = true;
par->RefineFeasibility = 2;

Final values after iteration 18:
Final objective value ............   1.3485126076E+05
Final constraint violation .......   7.0624611073E-07
Final KKT conditions .............   4.5137218774E-10
Successful termination: Optimal Solution Found.
------------------------------------------------------------
```

In Listing 3 the results for the parameter identification mode applied to the DTOC6 example are shown. In order to reduce the number of major iterations as efficiently as possible, the user should use a merit function approach with analytic Hessian and use the additional feasibility refinement. In this case only seven iterations are required. This speed-up comes at the cost of a slightly higher objective function value of $1.429 \cdot 10^5$, whereas the multi-core interface was able to identify five settings leading to an objective function value of $1.349 \cdot 10^5$ with at best 18 necessary major iterations.

6. CONCLUSION AND OUTLOOK

We presented a straight forward parallelisation approach for the NLP solver WORHP. Multiple problem instances are run in parallel and depending on the users choice the parallelism can be used to achieve different goals. In order to decrease the time necessary to obtain a local minimum all remaining threads can be terminated, when the first local minimum is found. Alternatively, the user can apply the multi-thread interface to increase the thoroughness of the optimisation run, as the interface waits for all running threads to terminate and pick the best obtained local minimum afterwards. This Best-Of-All approach can be extended to do some parameter identification for the solver for an user given particular problem. Again all threads continue until they finish. Afterwards all results are analysed and different parameter settings are

suggested to the user regarding different criteria, like speed, quality and low iteration counts.
In addition to the variation of parameters the multi-core interface could be used to vary initial guesses and thus offer an easily accessible way to globalisation methods. Similar approaches are used for mixed integer problem formulation and could be integrated into the interface.
Due to its flexible structure the interface can easily be extended to solve different problems in parallel and could thus be used to identify pareto fronts or perform reachable set analysis for optimal control problems. Furthermore, a direct integration into WORHP's companion transcriptor method TransWORHP [7] could be performed. In the optimal control context beside the parameters presented in this paper further settings like e.g. discretisation schemes could be customised for each thread.

7. REFERENCES

[1] C. Büskens and D. Wassel, "The ESA NLP Solver WORHP," in *Modeling and Optimization in Space Engineering*, Giorgio Fasano and Jnos D. Pintr, Eds., vol. 73 of *Springer Optimization and Its Applications*, pp. 85–110. Springer New York, 2013.

[2] S. Mehrotra, "On the implementation of a primal-dual interior point method," *Siam Journal Optimization*, vol. 2, pp. 575–601, 1992.

[3] D. Wassel, *Exploring Novel Designs of NLP Solvers*, Ph.D. thesis, Universitt Bremen, 2013.

[4] David H. Wolpert and William G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on evolutionary computation*, vol. 1, 1997.

[5] M. J. D. Powell, *Numerical Analysis: Proceedings of the Biennial Conference Held at Dundee, June 28–July 1, 1977*, chapter A fast algorithm for nonlinearly constrained optimization calculations, pp. 144–157, Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.

[6] N. I. M. Gould, D. Orban, and P. L. Toint, "Cutest: a constrained and unconstrained testing environment with safe threads for mathematical optimization," *Computational Optimization and Applications*, pp. 1–13, 2014.

[7] C. Büskens M. Knauer, "From worhp to transworhp," *Proceedings of the 5th International Conference on Astrodynmaics Tools and Techniques, 29.05.-01.06.2012, Noordwijk, Netherlands*.