THE ESA NLP-SOLVER WORHP – RECENT DEVELOPMENTS AND APPLICATIONS

Dennis Wassel, Florian Wolff, Jan Vogelsang, and Christof Büskens

Center for Industrial Mathematics, University of Bremen, PO Box 33 04 40, 28334 Bremen, Germany,
Email: {dwassel, fwolff, janv, bueskens}@math.uni-bremen.de

ABSTRACT

The European NLP solver WORHP (***We Optimize Really Huge Problems***, also referred to as *eNLP* by ESA) is a tool for solving large-scale, sparse, nonlinear optimization problems with millions of variables and constraints. It is being developed at the University of Bremen, supported by the TEC-EC division through GSTP-4 and 5, with the intention to replace SNOPT in existing and future ESA projects that involve continuous optimization.

This paper will briefly skip over the mathematical and algorithmic foundations of WORHP and focus on notable features, and recent or ongoing improvements, such as the genuinely sparse structure-preserving BFGS update, the weak-Active Set mode, the parametric sensitivity analysis module and the transcriptor TRANSWORHP, as well as two exemplary applications in model-predictive control of a swarm of independent agents and in satellite constellation and schedule optimization.

Key words: eNLP; Nonlinear Programming; Large-Scale Optimization; Mathematical Optimization; Numerical Optimization.

1. INTRODUCTION

WORHP solves problems of the form

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} && f(x) \\ &\text{subject to} && \begin{pmatrix} l \\ L \end{pmatrix} \leqslant \begin{pmatrix} x \\ g(x) \end{pmatrix} \leqslant \begin{pmatrix} u \\ U \end{pmatrix} \end{aligned} \tag{OP}$$

with bounds

$$l, u \in [-\infty, +\infty]^n, \quad L, U \in [-\infty, +\infty]^m,$$

and functions $f\colon \mathbb{R}^n \to \mathbb{R}$ and $g\colon \mathbb{R}^n \to \mathbb{R}^m$. If either x or g are unbounded, the corresponding bound is $\pm\infty$ (in practice, ∞ is replaced by a finite, large constant, such as 10^{20}).

Both the **objective function** f and the **constraints** g may be linear, quadratic or nonlinear. WORHP makes no assumptions on the problem structure, like convexity or conicity – only certain smoothness and regularity assumptions are needed by the theoretical foundations, however, these are next to impossible to ascertain for most problems of practical relevance. In practice, WORHP often finds solutions even if the differentiability requirement is not satisfied.

1.1. Fundamentals of WORHP

The general framework of WORHP is *Sequential Quadratic Programming* (SQP), a method devised in the 1960's by Wilson [10] and later revived by Han [8]. Since then they belong to the most frequently used algorithms for the solution of practical optimization problems due to their robustness and their good convergence properties. A more extensive account can be found in [6].

SQP methods can be proven to achieve global convergence and locally superlinear convergence rate; even locally quadratic convergence can be achieved, if second derivatives are available. KNITRO and the NLP-solver SPRNLP included in SOCS are all SQP methods.

The fundamental principle of SQP methods is to find KKT points, i.e. points that satisfy the *necessary* optimality conditions[1]. To find such points, the the **Lagrange function** L is introduced in terms of functions F and G, which are simple transformations of f and g to the mathematical standard formulation

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} && F(x) \\ &\text{subject to} && G_i(x) = 0, \quad i \in I \\ & && G_j(x) \leqslant 0, \quad j \in J \end{aligned} \tag{NLP}$$

The Lagrange function is $L(x) = F(x,\mu) + \mu^{\mathsf{T}} G(x)$. Any KKT point (x^*, μ^*) has to satisfy $\nabla_x L(x^*, \mu^*) = 0$, which is why the principle of SQP methods is to find zeros of $\nabla_x L(x, \mu)$ by applying Newton's method. This results in

[1] *Sufficient* optimality conditions can be formulated and tested, but this is not usually done, since the computational effort for medium- to large-scale problems is prohibitive.

quadratic subproblems of the form

$$\begin{aligned} \min_{d\in\mathbb{R}^N} \quad & \tfrac{1}{2}d^\intercal \nabla^2_{xx}L(x,\mu)d + \nabla_x F(x)^\intercal d, \\ \text{subject to} \quad & G_i(x) + \nabla_x G_i(x)d = 0, \quad i \in I \\ & G_j(x) + \nabla_x G_j(x)d \leqslant 0, \quad j \in J, \end{aligned} \tag{QP}$$

whose solution d, the **search direction**, is used to define the next iterate via

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

The **step size** α is chosen by using either a **merit function**, i.e. a scalar measure of "goodness" of the current trial point, such as the L_1 merit function

$$\begin{aligned} L_1(x;\eta) := F(x) + \sum_{i\in I} \eta_i |G_i(x)| \\ + \sum_{j\in J} \eta_i \max\{0, G_j(x)\}, \end{aligned}$$

or by a **Filter** i.e. a two-dimensional set of points that considers the objective function value and constraint violation separately, and accepts trial points, if they yield an improvement in either direction. The basic principle is illustrated by Fig. 1.

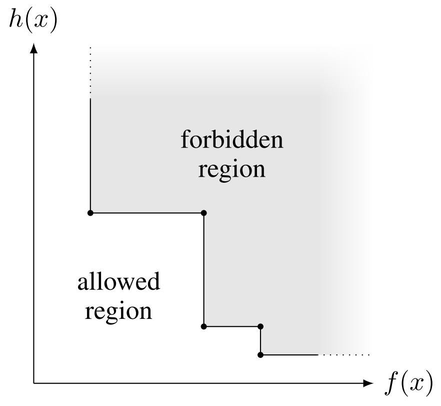


Figure 1. Schematic drawing of a Filter with 5 points that define the allowed and forbidden region – $f(x)$ is the objective and $h(x)$ the constraint violation. New points are accepted, if they lie in the allowed region.

1.2. Derivatives and Sparsity

In addition to f and g, NLP solvers in general and WORHP in particular need the derivatives ∇f, the **gradient** of the objective function, and ∇g, the **Jacobian** of the constraints. Both are assumed to be **sparse**, i.e. they have many structural zeros and only few entries that may attain numerical values other than zero. The ratio between structural zero and all entries of a matrix is called **sparsity**; matrices with sparsity of 100% are **dense**, i.e. they have *no* structural nonzeros, while matrices with sparsity of 0% contain *only* structural zeros.

Sparsity is of little concern for small problems with hundreds of variables and constraints (these are usually dense, or can be treated as such without noticeable performance impact), but essential for handling large-scale problems with dimensions of the order of 10^6 or above. Most classes of large-scale optimization problems have sparsity of 0.1% or even lower. An important class of optimization problems with these properties are discretized optimal control problems using *full discretization*, the method used by WORHP's companion transcriptor TRANSWORHP[2].

Determining the derivatives of non-academic problem formulations is nontrivial at best; for many it is outright impossible due to technical reasons (e.g. legacy codebase), the inherent problem formulation (e.g. interpolated table data), the sheer model complexity, or all of the above (e.g. interplanetary flight involving ephemeris calculations of celestial bodies). Since most of these cases are not tractable by automatic differentiation tools, either, users have to fall back to finite difference approximations.

Standard finite difference calculations are infeasible: The simplest case is the forward difference approximation

$$\frac{\partial \phi}{\partial x_i}(x) \approx \frac{\phi(x + \epsilon e_i) - \phi(x)}{\epsilon}, \quad i = 1, ..., n$$

requiring $n+1$ function evaluations, which can be prohibitively costly, since n may be large, or ϕ (either F or G) may be expensive to evaluate.

The problem is alleviated by the fact that F in many cases depends on few optimization variables only, hence the number of nonzeros n_{nz} is small compared to n, and only $n_{\text{nz}} + 1 \ll n + 1$ evaluations are needed. The constraints G, however, as a vector-valued function depend on *all* optimization variables, so a more sophisticated approach is necessary to calculate finite differences with tolerable effort:

Let the Jacobian of G have sparsity structure

$$\nabla G = \begin{pmatrix} * & * & & \\ & * & * & \\ & & * & * \end{pmatrix} = \begin{pmatrix} \bullet & \circ & & \\ & \circ & \bullet & \\ & & \bullet & \circ \end{pmatrix},$$

where $*$ denotes structural nonzeros. We can group variables, if all rows of ∇G depend on at most one variable per group. One possible grouping is denoted by $\circ$ and $\bullet$ in the example above. To calculate a finite-difference approximation of the Jacobian, we can now perform multiple perturbations in a single evaluation, and calculate the whole Jacobian approximation by evaluating

$$\frac{G\big(x + \epsilon(e_1 + e_3)\big) - G(x)}{\epsilon}$$

and

$$\frac{G\big(x + \epsilon(e_2 + e_4)\big) - G(x)}{\epsilon}$$

[2]For a discussion of transcription methods cf. [1, 4] and the presentation of TRANSWORHP by M. Knauer.

i.e. two evaluations of G (omitting the unperturbed one, which can be cached) in contrast to four evaluations for the naïve approach. An additional benefit of the group strategy is the *discretization-invariance* when applied to fully discretized optimal control problems – the number of groups is constant, irrespective of the discretization grid.

Constructing these groupings such that their overall number is minimal is an NP-hard problem. WORHP therefore uses efficient heuristics, which provide groupings close to optimal for most problems. With some extensions, the grouping approach can also be used to calculate finite difference approximations of second derivatives, which are needed to solve (QP). For a detailed account of advanced derivative approximation techniques see [9].

2. RECENT DEVELOPMENTS

2.1. TRANSWORHP

WORHP, from its inception, has been designed as the NLP solver component for solving optimal control problems of the form

$$\begin{aligned} \min_u F(y,u) &= \eta\big(y(t_0), y(t_f)\big) + \int_{t_0}^{t_f} \phi_0\big(y(t),u(t)\big)\,dt \\ \text{subject to} \quad \dot{y}(t) &= \phi\big(y(t),u(t)\big), \\ y(t_0) &= y_0, \\ \psi\big(y(t_f)\big) &= 0, \\ C\big(y(t),u(t)\big) &\le 0, \quad t \in [t_0,t_f] \end{aligned} \tag{OCP}$$

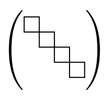
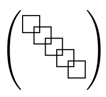
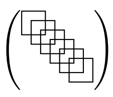
with $\eta\colon \mathbb{R}^{2n} \to \mathbb{R}$, $\phi_0\colon \mathbb{R}^{n+m} \to \mathbb{R}$, $\phi\colon \mathbb{R}^{n+m} \to \mathbb{R}^n$, $\psi\colon \mathbb{R}^n \to \mathbb{R}^r$, $0 \leqslant r \leqslant n$, and $C\colon \mathbb{R}^{n+m} \to \mathbb{R}^k$ being sufficiently smooth functions on appropriate open sets. The admissible class of control functions is that of piecewise continuous controls. The final time t_f is either fixed or free.

Optimal control problems can be understood as infinite-dimensional optimization problems, since the states y and the controls u have to be optimal for every point of time $t \in [t_0, t_f]$. When using *direct* methods, two major approaches exist to transform these infinite-dimensional optimization problems into finite-dimensional ones, either resulting in small and dense problems (NUDOCCCS [4]), or large and sparse problems (TRANSWORHP in "From WORHP to TRANSWORHP" by M. Knauer).

2.2. Structure-Preserving Sparse BFGS

Since finite-difference approximations to second derivatives may still be expensive to calculate, and since the finite difference approach is inherently ill-conditioned, it is usually infeasible to apply them to particularly difficult or large problems. The BFGS update technique is named after Broyden, Fletcher, Goldfarb and Shanno, who generalized the secant method to calculate approximations to the Hessian of the Lagrange function $\nabla^2_{xx}L(x,\mu)$. BFGS methods take an initial matrix and perform an *update* on it that is cheap to compute from known quantities. There exist variations of the basic technique, such as rank-1 vs. rank-2 updates (cf. [5, 6]), but all of them share favorable properties that are the reason for their widespread use in NLP solvers. One common property of the BFGS update is the fact that a single update step generally produces a dense matrix, irrespective of the sparsity of the initial matrix (this is usually chosen as the identity matrix, which *is* very sparse). This property precludes the use of standard BFGS update techniques for problems with more than a few hundred variables, since the number of entries of the Hessian $\nabla^2_{xx}L(x,\mu)$ grows as $\mathcal{O}(n^2)$, and would thus destroy any performance advantage over (sparse) finite-difference approximations.

Therefore WORHP offers a number of non-standard BFGS update techniques that are also suitable for large-scale problems: A first modification is to perform non-intersecting, intersecting or even multiply-intersecting block-BFGS updates of the form

$B_{ni} =$  , $B_i =$  , $B_{mi} =$  .

The non-intersecting case is straightforward, because the classic BFGS method can be applied to the individual blocks, while the intersecting cases need a technique called *convexity shifts* to maintain the BFGS properties. By choosing appropriate block sizes and overlapping, the structure of Hessian matrices with small bandwidths can be approximated or reconstructed using BFGS matrices of B_i or B_{mi} structure. If, however, the Hessian has (many) far off-diagonal entries, none of these approaches can cover them without sacrificing a substantial degree of sparsity (which is essential for adequate performance in large-scale optimization). Even though the solver is able to cope with missing elements, and will even converge if the Hessian is replaced by the identity matrix, this adversely affects convergence order. A worse (lower) convergence order in turn causes the NLP solver to require more – possibly *many* more – iterations to converge to an optimal point.

For these cases, WORHP offers the so-called SBFGS method. One central idea behind this method is illustrated by the following example:

Consider the sparse symmetric matrix

$$A = \begin{pmatrix} a & b & 0 & 0 \\ b & c & d & e \\ 0 & d & f & 0 \\ 0 & e & 0 & g \end{pmatrix}.$$

By selecting and rearranging the elements of A, we can form three dense sub-matrices matrices

$$A_1 = \begin{pmatrix} a & b \\ b & c \end{pmatrix},\ A_2 = \begin{pmatrix} c & d \\ d & f \end{pmatrix},\ A_3 = \begin{pmatrix} c & e \\ e & g \end{pmatrix}$$

and perform the standard BFGS update on them, to obtain $\tilde{A}_i$. An updated form $\tilde{A}$ of the original matrix A can then be reassembled from the $\tilde{A}_i$, since all elements are accounted for. However, c is present in all three sub-matrices A_i, resulting in *three* potentially different elements $\tilde{c}_i$ after the dense BFGS updates on $\tilde{A}_i$. The updated element $\tilde{c}$ can be obtained from a convex combination $\sum_i \lambda_i \tilde{c}_i$ with suitably chosen λ_i.

The SBFGS method uses the above idea to decompose the Hessian into smaller dense sub-matrices, perform the classic BFGS update on them, and then reassemble them appropriately. The mathematical challenge is to preserve certain properties of the update to admit a convergence proof and maintain positive definiteness, while the technical challenge is to find the minimum number of dense sub-matrices of maximum size. Mathematical details and a convergence proof can be found in [9].

2.3. Post-Optimality Analysis

WORHP's post-optimality analysis module is currently in development. It is based on the theory of Parametric Sensitivity Analysis [2, 3] and allows a range of advanced applications of nonlinear optimization, including real-time optimization and deterministic risk assessment.

The mathematical background is based on a perturbed version of (NLP)

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & F(x;\mathbf{p}) + \mathbf{r}^\mathsf{T} x \\ \text{subject to} \quad & G_i(x;\mathbf{p}) = 0 + \mathbf{q}_i, \quad i \in I \\ & G_j(x;\mathbf{p}) \leqslant 0 + \mathbf{q}_j, \quad j \in J \end{aligned} \tag{NLP$_\mathrm{p}$}$$

where $\mathbf{p} \in \mathbb{R}^n$, $\mathbf{q} \in \mathbb{R}^m$ and $\mathbf{r} \in \mathbb{R}^k$ are arbitrary perturbations. Parametric sensitivity analysis is a tool that allows to compute sensitivity derivatives of important optimal values, such as

$$\frac{dF}{d\star}(x^*),\ \frac{dG}{d\star}(x^*),\ \frac{dx^*}{d\star}$$

with $\star \in \{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$. The sensitivity derivatives have a wide range of possible applications, in all areas where optimization is used: The sensitivities of optimal trajectories enable on-board *optimal* closed-loop control of any kind of craft; an example is a reentry vehicle that maintains its optimal trajectory even under atmospheric density fluctuations. Applied to a design cycle, for instance applying topology optimization, high sensitivities can be used to improve a design by raising its robustness against the respective perturbation. Sensitivities allow deterministic and quantitative risk assessment, since it is possible to exactly[3] determine the size of (even multiple, concurrent) perturbations that would cause the modeled process to fail.

[3]The actual precision depends on the size of the perturbation, since parametric sensitivity analysis is a *local* theory. Using second-order sensitivity derivatives increase admissible perturbation sizes and precision.

As a more extensive example for an application of post-optimality analysis, we can consider a (possibly MDO) launcher design, where the objective $F(x;\mathbf{p})$ is the mass that can be launched into a specific orbit, and our goal is to vary some design parameters and observe the change of the objective with respect to these changes.

The standard approach to this problem is Monte-Carlo simulation, using a high number of simulations with randomly chosen design parameters, which yields a non-deterministic result, i.e. a probability distribution for the optimum. The computational costs are high, since many complete optimization runs have to be performed.

Using post-optimality analysis, we can instead fix a design parameter $\mathbf{p_0}$ and compute $F(x^*;\mathbf{p_0})$ and $\frac{dF}{d\mathbf{p}}(x^*;\mathbf{p_0})$ once. Changing the design parameter $\mathbf{p_0}$ can be considered as a perturbation $\mathbf{\Delta p} = \mathbf{p} - \mathbf{p_0}$ of the original problem, whose solution is known. Using the previously computed sensitivity derivative, the perturbed optimal solution for a specific value of $\mathbf{p}$ can now be computed through

$$F(x^*;\mathbf{p}) \approx \frac{dF}{d\mathbf{p}}(x^*;\mathbf{p_0}) \cdot \mathbf{\Delta p},$$

which is very cheap to evaluate. The error (due to truncating the Taylor approximation) is of order $\mathcal{O}(\|\mathbf{\Delta p}\|^2)$.

Compared to the Monte-Carlo approach, the post-optimality analysis has initial computational costs that are somewhat higher than a single optimization run, but allow exhaustive analysis of the perturbation parameter space at virtually no cost, with deterministic results. This process can further be improved by combining this with $\mathbf{q}$ and $\mathbf{r}$ perturbations (see (NLP$_\mathrm{p}$)), and possibly second-order sensitivity derivatives for higher precision and greater admissible perturbation size.

2.4. Weak-Active Set

Textbook SQP methods use the so-called *Active Set* method to handle inequality constraints. An inequality constraint $g(x) \leqslant 0$ is called *active*, if the inequality turns into an equality $g(x) = 0$, i.e. the constraint "reaches its bound". The Active Set approach tries to determine, which of the inequality constraints $G_j(x) \leqslant 0, \quad j \in J$ in (NLP) will become *active* in the next iteration. The active inequality constraints are then treated as equalities, while the inactive constraints are dropped. Since the method has to make a prediction about a nonlinear function G, it is not always exact, and therefore has to iterate until the correct active set has been determined. Although the active set stabilizes close to the optimum, the iterative determination has worst-case exponential complexity, and could thus be unsuitable for large-scale problems. For this reason, WORHP uses a different internal approach that is able to handle inequality constraints, but has higher complexity than the standard approaches for inequality constraints only.

Due to this, the Weak-Active Set (WAS) approach tries to reduce the number of inequality constraints that have to be considered by making a similar prediction to the strict (non-"weak") method, and dropping those inequality constraints that have a high probability of not becoming active in the next iteration. Potential candidates for inequality constraints that may become active are left untouched, since WORHP can internally handle them fine.

n	m	standard	WAS
50,000	149,997	8.1 s	1.6 s
100,000	299,997	13.3 s	3.5 s
250,000	749,997	54.4 s	15.2 s
500,000	1,499,997	249.2 s	40.3 s

Table 1. Timings for a modified version of the mccormck *problem from CUTEr, comparing the standard and the Weak-Active Set approach for different problem sizes. Given times are pure NLP times without time spent in AMPL routines.*

This approach is especially powerful for optimal control problems with inequality path constraints (C in (OCP)) that are active only in certain phases of the overall process. Using the Weak-Active Set, WORHP can significantly reduce the problem sizes and thus achieve a noticeable performance boost. Table 1 shows an example that profits greatly from the Weak-Active Set approach.

3. APPLICATIONS

3.1. CUTEr

While CUTEr by itself is not a genuine application, it is a commonly used "yardstick" to measure a solver's robustness and performance.

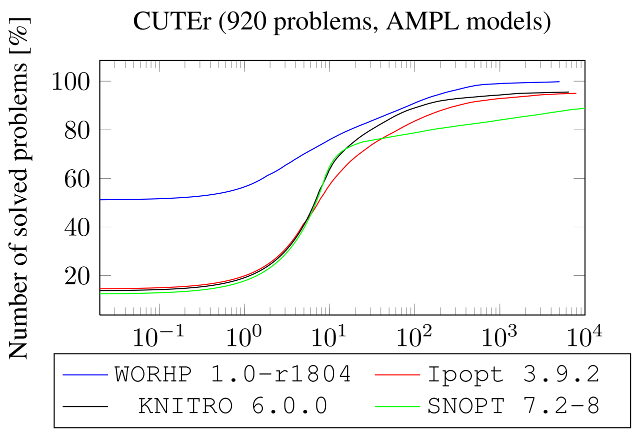


Figure 2. Semi-logarithmic plot showing the percentage of solved problems (of the CUTEr set) against the time needed to solve them.

Figure 2 demonstrates that WORHP is the fastest and most robust among the test candidates on the CUTEr test set.

3.2. Model-Predictive Control of a Swarm

The following example is motivated by [7]. Consider a swarm of P agents moving in the 2-dimensional plane, described by

$$\dot{x}_i(t) = \phi\big(x_i(t), u_i(t)\big), \quad i = 1, \dots, P \qquad (1)$$

with states $x_i = (x_{i1}, \dots, x_{i4})^\intercal \in \mathbb{R}^4$ and controls $u_i = (u_{i1}, u_{i2})^\intercal \in \mathbb{R}^2$ and a system dynamic $\phi(x_i, u_i) = (x_{i2}, u_{i1}, x_{i4}, u_{i2})^\intercal$. The position of each agent i in the plane is described by (x_{i1}, x_{i3}) and its velocity by (x_{i2}, x_{i4}). The acceleration of each agent can be controlled by the control input (u_{i1}, u_{i2}). The overall dimension of the problem is therefore $4P$ states and $2P$ controls.

The control input is constrained by $-12 \leqslant u_{i1}(t) \leqslant 12$, and $-12 \leqslant u_{i2}(t) \leqslant 12$. The state constraints are given by

- collision avoidance constraints

$$\|(x_{i1}(t), x_{i3}(t))^\intercal - (x_{j1}(t), x_{j3}(t))^\intercal\|_2 \geqslant 0.08 \quad \forall t \;\; \forall i, j = 1, \dots, P,\; i \neq j.$$

- velocity constraints

$$\|(x_{i2}(t), x_{i4}(t))^\intercal\|_2 \leqslant 1 \;\forall t \quad \forall i = 1, \dots, P.$$

- static obstacle constraints

$$(x_{i1}(t), x_{i3}(t))^\intercal \notin \overline{B}_{r_p}(y_p), \quad \forall t, \quad p = 1, 2, 3, \quad \forall i = 1 \dots P,$$

  i.e. circular obstacles with midpoints $y_1 = (1.1, -0.3)^\intercal$, $y_2 = (2, 0.3)^\intercal$, $y_3 = (2.4, -0.2)^\intercal$ and $(r_1, r_2, r_3) = (0.4, 0.3, 0.15)$. $\overline{B}_r(y)$ denotes the closed ball with radius r around y.

- moving obstacle constraints

$$(x_{i1}(t), x_{i3}(t))^\intercal \notin \overline{B}_{0.13}(y_4(t)), \;\forall t, \;\forall i = 1 \dots P,$$

  with $y_4(t) = (-0.35 + 0.925t, -1.2 + 1.225t)^\intercal$.

- initial state constraints

$$x_i(0) = \big((i-1)0.12, 0, 1, 0\big)^\intercal \;\; \forall i = 1 \dots P.$$

The state constraints render the overall problem nonlinear, even though the dynamic (1) is linear.

The goal is to find a closed-loop control $u = \mu(x)$, $\mu\colon \mathbb{R}^{4P} \to \mathbb{R}^{2P}$ that moves all agents to the the point of origin $x_1^e = 0$ until $t = 4.5$ s and after $t = 4.5$ s until $t = 8$ s to $x_2^e = (3, 0, 0, 0)^\intercal$.

We solve this problem with a nonlinear model predictive control (NMPC) approach. The idea of NMPC is to solve an optimal control problem at each sampling instant k for

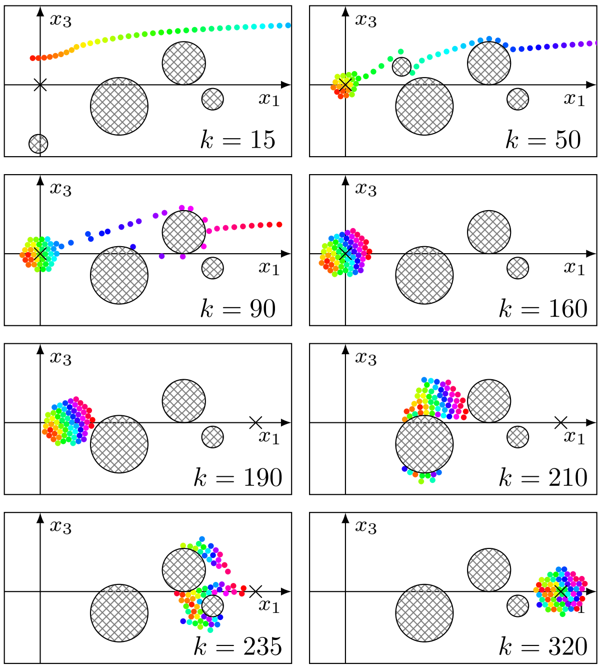


Figure 3. Agents and obstacles at 8 different time points. Note the moving obstacle in the first two plots. The swarm first moves towards the origin x_1^e, trying to minimize their distances, and after the switch to the second target x_2^e at $k = 180$, begin to move through the obstacles.

a given finite optimization horizon. To obtain a closed loop control, only the first step of the optimal control is actually carried out by the system, and after each time step, a new control sequence with updated initial state constraints is calculated. With the NMPC approach, a sequence of optimal control problems (OCP) within the sampling period T have to be solved.

Figure 3 shows how this problem is solved for $P = 64$ agents by an NMPC-Algorithm with WORHP. To solve the swarm problem, the OCP are discretized using an explicit Euler scheme. The sampling period is chosen as $T = 0.025\,\mathrm{s}$ and the horizon as $N = 7$, such that in each step k, the objective function

$$\min_{u \in U^N} J(x,u) := \sum_{j=0}^{N-1} \sum_{i=1}^{P} \Big(\big\| (x_{i1}^j, x_{i3}^j)^\mathsf{T} - x^e \big\|_2^2 + \big\| (x_{i2}^j, x_{i4}^j)^\mathsf{T} \big\|_2^2 / 4000 \Big)$$

is minimized subject to the abovementioned constraints and the equality constraints arising from the discretization of the system dynamic, with $x^e = x_1^e$ for $k = 1 \dots 179$ and $x^e = x_2^e$ for $k = 180 \dots 320$. Note that the control u is not penalized and that the problem is solved without terminal constraints. For this specific case WORHP has to solve 320 optimization problems with each 2,560 variables and 18,912 constraints. The system shows exactly the desired behavior, as seen in Figure 3.

The limited look-ahead horizon of each agent can be observed by noticing that they only start evading obstacles shortly before they are reached, whereas an optimal control approach would start the evasive maneuver earlier.

3.3. Satellite Coverage Optimization

The task of this application is to maximize the covered area of an Earth observation satellite or a constellation of satellites. The available degrees of freedom are (usually a subset of) the Keplerian elements of the satellite, and the observation schedule, where the latter may be subject to various constraints concerning the sensor duty cycle, downlink availability and on-board storage space, energy budget, etc. Additional requirements may be revisits, minimum time to observation, local sunlight conditions of the observed area, etc. All in all, the general problem statement has very many possible specializations, and is highly relevant to Earth observation.

If we denote with x_{2i-1} the starting time, and with x_{2i} the ending time of swath i, we can formulate a basic problem as

$$\begin{aligned} \max_{x \in \mathbb{R}^{2n}} \quad & \mathrm{Area}(x) \\ \text{subject to} \quad & x_{2i} - x_{2i-1} \leqslant l_{\max} \text{ (sensor limitations)}, \\ & x_{2i} - x_{2i-1} \geqslant 0 \text{ (positive swath length)} \\ & x_{2i+1} - x_{2i} \geqslant s \text{ (time between swaths)}. \end{aligned}$$

The established approach is to cover Earth with a grid of discrete points to keep track of observed areas. Due to its discrete nature, this approach cannot make reliable statements about the area *between* points (although this can be remedied by local grid refinement), and is not tractable for smooth optimization, since it is inherently *non-differentiable*.

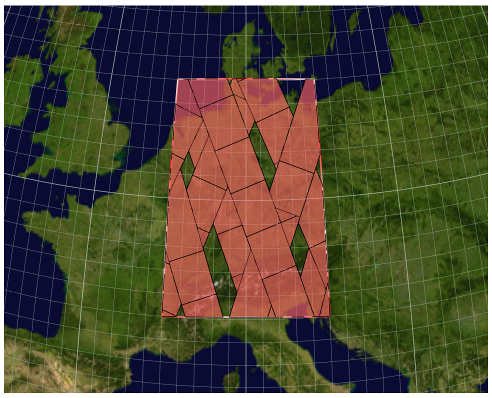


Figure 4. A couple of overlapping swaths over Europe. One can clearly observe that all polygons are convex, and that overlaps have been simplified by merging.

Our new approach models areas on an ellipsoid as a set of polygons. This allows precise calculation of the covered area with respect to WGS 84 or other ellipsoid Earth models in use. One major challenge is the calculation of arbitrary polygons, which can be simplified by requiring them to be convex – a non-convex polygon is split into two or more convex ones. Since it is desirable to represent the covered area with as few polygons as possible, iterative simplification, as the covered area increases, is used to simplify the area by merging adjacent polygons into fewer polygons. Another complication is caused by the fact that swaths often cover an area multiple times, which has to be taken into account when merging to calculate the whole observed area. Figure 4 shows an example of multiply-covered areas with merged polygons.

The formulation is differentiable, so it is tractable for optimization with WORHP, allowing to optimize orbits, constellations and observation schedules with unprecedented accuracy.

ACKNOWLEDGMENTS

Development of WORHP has been supported by BMWi (German Federal Ministry of Economics and Technology), the TEC-EC Control Division of the European Space Agency (ESA) in the projects eNLP (GSTP-4) and eNLPext (GSTP-5) and Steinbeis Research Center (SFZ) Optimization and Optimal Control.

REFERENCES

1. John T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM Press, Philadelphia, Pennsylvania, 2001.

2. Christof Büskens. *Optimierungsmethoden und Sensitivitätsanalyse für optimale Steuerprozesse mit Steuer- und Zustands-Beschränkungen*. Dissertation, Universität Münster, 1998.

3. Christof Büskens. *Echtzeitoptimierung und Echtzeitoptimalsteuerung parametergestörter Probleme*. Habilitation, Universität Bayreuth, 2002.

4. Christof Büskens and Helmut Maurer. SQP-methods for solving optimal control problems with control and state constraints: Adjoint variables, sensitivity analysis and real-time control. *Journal of Computational and Applied Mathematics*, pages 85–108, 2000.

5. Roger Fletcher. An optimal positive definite update for sparse Hessian matrices. *SIAM Journal on Optimization*, 5(1), 1995.

6. Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, 1981.

7. Lars Grüne. NMPC without terminal constraints. Preprint, 2012.

8. Shih-Ping Han. Superlinearly Convergent Variable Metric Algorithms for General Nonlinear Programming Problems. *Math. Programming*, 11(3):263–282, 1976/77.

9. Patrik Kalmbach. *Effiziente Ableitungsbestimmung bei hochdimensionaler nichtlinearer Optimierung*. Dissertation, Universität Bremen, 2011.

10. Robert B. Wilson. *A Simplicial Algorithm for Concave Programming*. PhD thesis, Harvard University, 1963.