Zentrum für Technomathematik

Fachbereich 3 – Mathematik und Informatik

Solver Development Strategy

Christof Büskens **Patrik Kalmbach**
Tim Nikolayzik **Dennis Wassel**

Report 08–02

Berichte aus der Technomathematik

Report 08–02 **März 2008**

Solver Development Strategy

C. Büskens, P. Kalmbach, T. Nikolayzik, D. Wassel

Abstract. Future space missions arising from the demand for branched trajectories for rockets and entry vehicles, weak stability boundary trajectories as well as trajectories for space vehicles with extremely low thrust propulsion are very complex. Discretization of such missions results in the necessity to solve extremely large nonlinear optimization problems with up to more than 300,000 variables and constraints.

The following paper shows the strategy how to develop a sparse nonlinear optimization solver due to the requirement of the European Space Agency and to be able to solve these larger nonlinear problems associated with trajectory optimization problems.

.

CONTENT

FIGURES

1 INTRODUCTION

According to the requirement eNLP-F-001, the solver developed in this activity shall solve problems of the form

$$\begin{aligned} \min_x \quad & f(x) \\ w.r.t. \quad & g(x) \leq 0, \\ & h(x) = 0. \end{aligned} \qquad \text{(NLP)}$$

where

$$f : \mathbb{R}^n \to \mathbb{R}, \quad g = (g_1, \ldots, g_m)^\top : \mathbb{R}^n \to \mathbb{R}^m, \quad h = (h_1, \ldots, h_p)^\top : \mathbb{R}^n \to \mathbb{R}^p$$

are sufficiently smooth functions.

NLP denotes the standard nonlinear optimization problem. The most important other abbreviations we will use are SQP (Sequential Quadratic Programming), QP (Quadratic Programming) and IP (Interior Point). The methods presented in this document combine the experiences of SQP technologies with the newest cognitions about the IP methods.

As the solver is supposed to solve especially space related applications, the problems will be of large scale and sparse. This means that one has up to million optimization variables and constraints or even more. A usual dense NLP solver would fail due to the enormous memory requirement and computational effort. To solve those problems one has to consider their sparse structure.

This document describes the strategies that we propose in order to get the best possible solver at the end of this activity and in the given time frame. There are many criteria how to measure a good solver. As robustness, accuracy, computational performance, user-friendliness, real-life suitability, maintainability and augmentability are all important properties of a solver some of which do not fit perfectly together, a good solver might imply that one has a good balance of these properties.

As again, it is difficult to clarify what a good balance is, the strategy we would like to suggest is provide a wide range of modules at all solver levels such that a user's preferences or the concrete application determines how the solver should work.

As already pointed out (see also "Current Trends in NLP methods", output of WP 1200):

"The best strategy is to have many strategies."

2 PROJECT DEVELOPMENT

2.1 GENERAL STRATEGIES

2.1.1 The Concept of Modularity

In order to build a platform for further developments in an european environment, the NLP solver will have a modular architecture on all possible levels (see requirements eNLP-M-005, eNLP-M-006, eNLP-M-007 and eNLP-M-008). This means that not only different solvers can be chosen, but also different combinations of QP methods, derivative calculations, outputs, user support, calling hierarchy, interfaces and many more. Figure 2-1 shows a graphical overview of the main implemented and foreseen modules.

For example, one can choose the combination consisting of the Full-Feature interface, the SQP solver with exact Hessian, the group strategy, a pre-iteration analysis and the IP method for solving the subproblems. This allows that every new research development leading to promising NLP solver properties can be implemented in the solver very easily just as an additional module.

Furthermore, it allows every user to taylor his version of the solver to the needs of his own single problem and his own preferences. The modularity will probably be one of the most advantageous properties of the solver as providing several methods and modules is obviously much more sensible than to provide only one single method, claiming this to be the best for almost ideological reasons.

Every module can be chosen via the parameters (see requirement eNLP-U-006). Of course, according to requirements eNLP-U-003, eNLP-U-004 and eNLP-U-005, there will be a standard setting that can be chosen without any work for the user. All parameters will be stored in a dedicated data file, according to requirement eNLP-U-007.

It might happen that some parameters do not fit together. For this reason and according to requirement eNLP-U-008 there shall be a plausibility check that tells the user about this problem.

To make conversion between the defined modules easy and clear, there will be a unified interface such that every variable is passed to the different routines in the same way and is available everywhere.

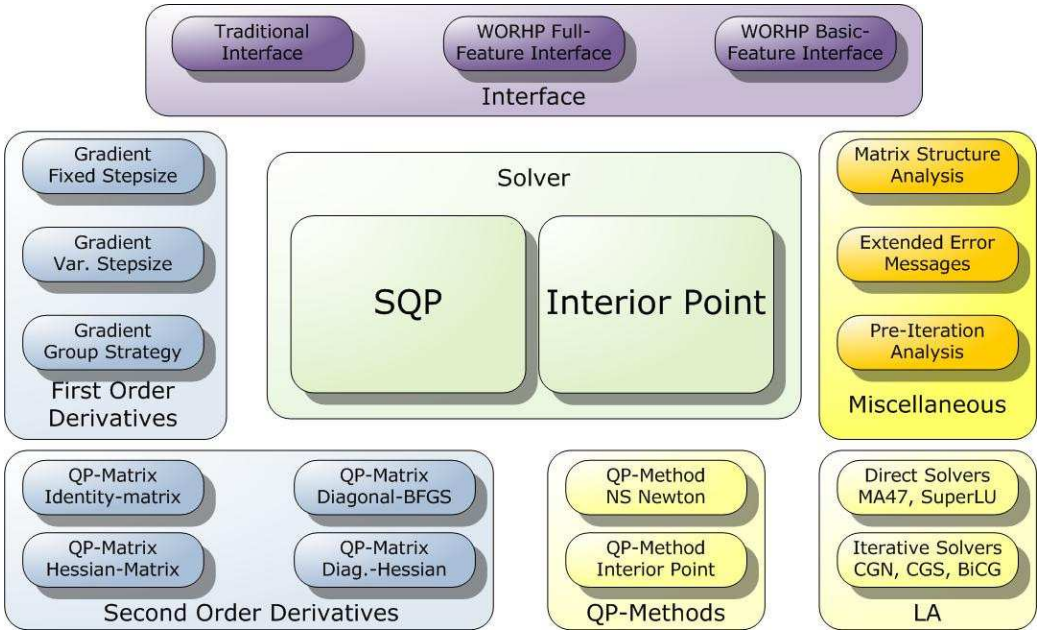

Figure 2-1: Modularity

2.1.2 Functionalities

2.1.2.1 Feasibility Only

Many NLP methods work much better when they are given a feasible or at least close to feasible starting guess. Hence one strategy to guarantee a higher robustness and to enlarge the modularity is to incorporate a method that produces feasible solutions. This has been formulated in requirement eNLP-F-011.

However, many solution strategies do not care if there starting guess is feasible or not. So this option should only be applied when the used method needs it or in cases where one is only interested in a feasible solution.

2.1.2.2 Monitoring and Reverse Communication

By starting the optimization process in the usual way, one has no influence on the result of the solver. Although this might be very convenient to solve an NLP problem by simply formulating it and getting back the optimal solution, this will not always work and one will end up with an error message instead of an optimal solution, even if one has an excellent solver. A human interaction cannot be replaced by a machine in every situation.

This holds especially if the solver provides a wide range of different modules and parameters. Therefore a strategy to improve the solver is to provide the most possible information about the optimization process and to allow an influ-

ence at as many points of the solver as possible.

Reverse Communication offers the experienced user a lot more options to take influence on the optimization process than the classical calling convention. This means not only to analyze the iterations but also to change some parameters or even the whole optimization method during the iteration for one single step or more. This concept has been shown to be very helpful, e.g. in the NLP solver NLPQLP of Schittkowski.

The solver will realize the concept of the so called “Single-Step Solvers” which is called by the user in every iteration step. This fulfils the requirement eNLP-U-017.

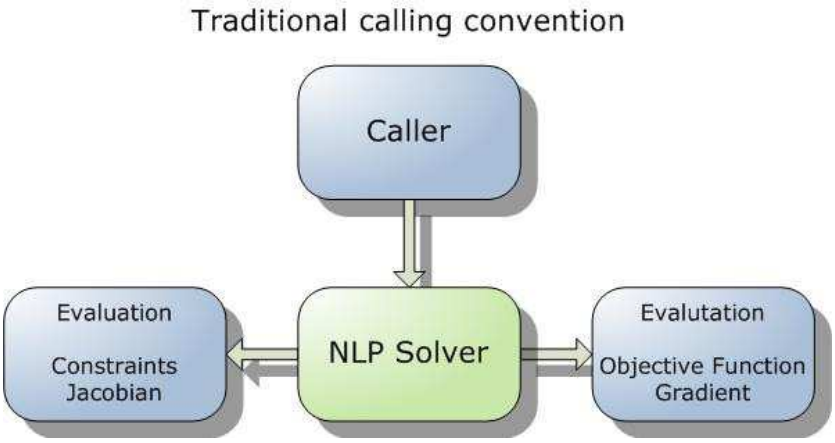

Figure 2-2: Traditional calling convention

Among the total transparency of the optimization process for the informed user, this interface allows any complex formulation of the problem specifying subroutines, which are used by the user to define his optimization problem.

More than this, it allows a deep analysis helping the user to tackle almost every problem by telling him exactly what has gone wrong and what he shall change. This further development of the Reverse Communication is currently subject of a project between the DLR, ASTOS and SRC Bremen.

The Reverse Communication will be implemented to the largest possible extend. However, as this activity is build up with already existing solvers, this cannot be done on every level.

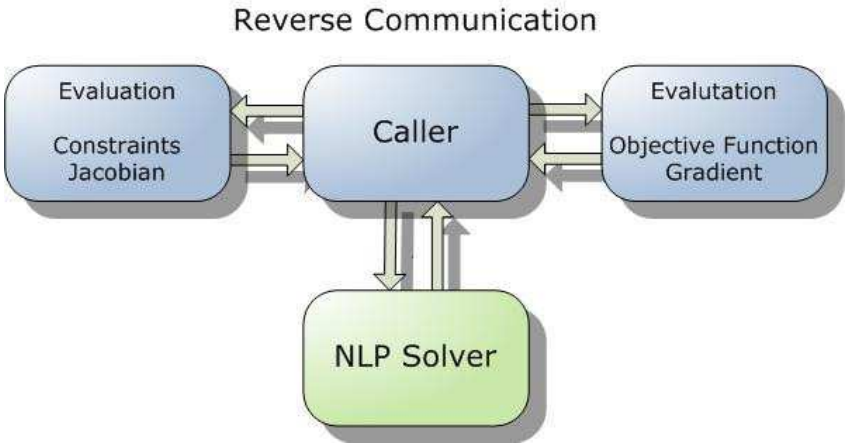

Figure 2-3: Reverse Communication

2.1.2.3 Warm Start Capability

According to requirement eNLP-F-013, the user will be able to abort the optimization process at any possible point and start again later from the current optimization point if he wishes so. As the whole set of variables and parameters will be available anywhere, it is clear that these can serve as an input for a "new" optimization.

2.1.2.4 Regularization

The solver developed in this activity shall fit to real-world, especially space related problems. There exists a wide range of NLP solvers that work excellently for purely mathematical problems but fail in many real-world applications. This is caused by a lack of robustness which has to be tackled. To this end several regularization strategies will be taken into account.

The difficulties one has to cope with are

ill-conditioning,

bad scaling,

rank deficiencies,

non-convexities.

All these difficulties may influence directly or indirectly the numerical solution of linear equations as matrices become (numerically) singular or very ill-conditioned, or they may lead to breakdowns in the line-search of the solvers. We will investigate and develop suitable approaches to deal with ill-conditioning, bad scaling, rank deficiencies, and non-convexity.

An automatic scaling strategy that is supposed to transform the data of the problem into a suitable desired range will be taken into account, both in the

context of QP (and other related subproblems) and at the NLP level. Such automatic scaling strategy will be tested within WORHP and IPFILTER (see requirement eNLP-F-014).

However, it is a generally accepted opinion that there is no good generic scaling method. First of all, it is not clear what "good scaling" actually means. There exist only *hints* how this could be done, no rules how the robustness can be improved by scaling. For example, a good scaling in the objective function might lead to a bad scaling in the Hessian.

Furthermore, even if a *good* scaling is achieved at a certain point (maybe the starting guess), this may be a *bad* scaling at another point. The NLP solver SOCS for example, uses only a scaling at the initial guess but allows the user to provide more different scaling methods.

Moreover, in the ill-conditioned or rank deficient case, we will modify the underlying linear equations by adding a weighted positive definite diagonal matrix to the system matrix. This will also lead to a convexification of the problem, but to a worse Hessian.

On the one hand this will enable us to solve the linear equation and compute a step direction, but on the other hand suitable strategies for the update of the weights have to be derived. This is actually the most crucial part in this strategy, because this regularization approach requires a certain amount of mathematical analysis and justification in order to guarantee the overall performance of the NLP solver.

It has to be taken into account that certain properties of the search direction (e.g. descent property) should be guaranteed also for the regularized problem.

Finally, we will develop safeguard strategies based on alternative NLP methods, if for instance a search direction turns out not to be a direction of descent or if the QP solver is not able to provide a suitable direction at all.

For primal-dual interior-point methods (e.g. IPFilter) that base the step computation on the solution of symmetric indefinite systems, most of the issues are similar, but here the appropriate control of the inertia of the matrix is the dominant task.

The usage of the (exact) Hessian of the Lagrange function has the drawback that it may be indefinite and/or singular. This eventually causes linear equations involving the matrix to be singular.

In the context of large-scale problems it is a highly non-trivial task to identify linearly dependent rows or columns as standard decomposition methods like LU or QR-decomposition (meaning decompositions into lower and upper triangular matrices or orthonormal and upper triangular matrices) cannot be applied

anymore to large-scale systems. Even detecting singular matrices is a difficult task. Hence, one needs to regularize linear equations using, e.g., strategies of Levenberg-Marquardt type.

Another approach would be to solve linear equations in a least-squares sense (in combination with a Levenberg-Marquardt approach). This approach may lead to severe ill-conditioning.

Current strategies to deal with singularities can be summarized as follows:

Try to regularize the KKT matrix by adding a sufficiently large suitable matrix to the matrix.

If solvers fail to compute a solution, try something else (different search direction, different method, feasibility restoration).

2.1.2.5 Control of Inexacteness

Many computations in an NLP solver are performed inexactly, like the ones involved in the solution of

approximated subproblems

linear systems

linear least-squares systems

restoration of feasibility, warmstarts, etc.

Inexacteness can affect both the local and global convergence properties of the NLP algorithms. The impact of inexact calculations on the global convergence behaviour, mainly in the context of SQP is actual research and will be part of this NLP project.

This requires a careful inspection of the convergence theory and a thorough testing campaign. Far from the solution we would like to allow a certain amount of inexacteness but these quantities should be tighten as the iteration progresses.

A practical control of inexactness requires the a priori knowledge of the tolerances to be imposed. On the other hand, the future incorporation of iterative methods for the NLP solver linear algebra is dependent on a rigorous study of the impact of inexactness. A certain amount of work will thus be devoted to the study of inexactness. We will also make some preliminary attempts to incorporate matrix-free Krylov methods within the solver.

There will be mutual synergetic effects of this working package with WP3200, WP3300 and WP3400 and an intensive testing of the algorithms in WP4100, WP4110 and WP4120.

2.1.3 Sparsity Information

Many large scale problems are sparse, meaning that sometimes much less than 1% of the occurring matrix entries are zero by definition. Hence many computations need not be done as the result is known in advance to be zero. This holds on every level of the solver so that a huge amount of computational operations need not be performed. Moreover, the variables need not be stored as they do not change during the optimization process. Regarding the sparsity fulfils requirement eNLP-F-005.

2.1.4 Documentation

The documentation will consist of a technical report, describing the implemented algorithms as well as a manual describing how to use the solver (see requirement eNLP-M-010). Furthermore it is foreseen to write a document continuously to fix every implemented change of the software. Of course, the whole documentation will be written in English (see requirement eNLP-M-012).

2.2 SOLVER STRATEGIES

2.2.1 Function Definition

In order to hold the computation time to a minimum and the mathematical robustness to a maximum, it is crucial to use as much available information as possible about the functions arising in an NLP problem. The strategy is to subdivide functions into different parts so that their derivatives can be computed very easily.

Formulating an optimization problem means of course to formulate various functions. An inexperienced user will surely not put too much effort in the input of the solver. Hence it is not necessary to provide more information than the values of the functions at each desired point. However, the more information the user provides, the better the solver will work with respect to computation time, robustness, accuracy, etc.

2.2.1.1 Objective Function

The objective function is the function that the user wants to be minimized or maximized. The solver will be able to handle an arbitrary form of this function. It can be given as a routine that calculates each value depending on an input value.

Usually, not all optimization variables have an influence on the objective function. If the objective function is independent of a certain variable, this should be clear from the sparsity information, provided by the user, see section 2.1.3.

Assuming a variable has an influence on the objective function, this influence can have various shapes. Therefore, the objective function will be subdivided into three parts, the linear part, the quadratic part and the general nonlinear part.

A general objective function looks like this

$$f(x) = a^T x + \frac{1}{2} x^T Q x + \tilde{f}(x),$$

Where a denotes a real valued vector of dimension n (being the dimension of x) and Q a real valued n-times-n-matrix.

If the user does not provide such information, the objective function will be assumed to be nonlinear. This distinction allows a quicker evaluation of the objective and its derivatives, it enlarges stability and it helps detecting an NLP to be a QP problem.

2.2.1.1.1 Linear Part

The first part of f, $a^T x$, is called the linear part. If this is known in advance, it facilitates the calculation of the derivatives a lot.

Independently of the value of x, the first derivative of this part is a. Hence, once a is determined, nothing has to be calculated any more. Furthermore, the second derivative of this part is always known to be zero. Thus, no calculation at all has to be done.

2.2.1.1.2 Quadratic Part

The second part of f, $x^T Q x$, is called the quadratic part. Again, this information is very helpful for the determination of the derivatives. It is clear that the first derivative of this part is Qx. Once Q is known, one can easily determine every first derivative and the second order derivative is always Q itself. Furthermore if f is a purely quadratic function, it can directly be minimized by the QP solver. Their exist many applications for quadratic optimization problems, e.g. in control theory. This is why they are well understood.

2.2.1.1.3 Nonlinear Part

The rest of the function, denoted here by $\tilde{f}(x)$, is the general nonlinear part and therefore assumed to be arbitrarily complicated. However, if the information about these three parts is provided, this complicated part is reduced to a minimum and therefore it can be evaluated much quicker.

2.2.1.2 Constraints

One can think about subdividing all constraints into the three parts mentioned above. On the other hand, this might lead for the quadratic part to rang three tensors and hence to an unacceptable memory requirement and an enormous user effort. Hence another distinction might be more sensible.

2.2.1.2.1 Box Constraints

A constraint that borders a variable only, such as

$$l_B \leq x \leq u_B$$

is called box constraint. Of course, it can be treated as a common constraint. However, knowing the fact that it is a box constraint, one can set the first derivative with respect to x to 1 and any other and all second derivatives to zero. Note that l_B, u_B and x can be vectors, of course.

2.2.1.2.2 Linear Constraints

A constraint of the form

$$l_L \leq Ax \leq u_L,$$

where A is a matrix, is called linear. Analogously to the argumentation above, the knowledge about linearity saves a lot computational effort.

2.2.1.2.3 Nonlinear Constraints

All constraints that have at least one nonlinear part, such as

$$l_N \leq g(x) \leq u_N,$$

where g is a vector valued nonlinear function, is called nonlinear and its derivative has to be calculated without any further knowledge about this function.

2.2.1.2.4 Handling Equalities and Inequalities

When a problem consists of equality and inequality constraints, then there are many ways how to formulate it. One way is to divide them into two vectors g and h (see equation (NLP)), where g is the vector of inequality constraint functions and h the vector of equality constraints function.

Furthermore the values of the equalities and the border(s) of the inequalities have to be defined in another vector. Additionally, we might have to divide them into inequalities with lower and upper bounds and inequalities with only one bound.

This leads to lots of different types of constraints to be handled separately while this distinction is useless.

Therefore the more sensible definition of constraints is to keep them all in one vector function g and to give all these constraints a lower and upper bound l and u. In case of an equality constraint, l will equal u. In case of a one-side inequality one border will be set to plus or minus infinity. This is the most user-friendly approach and shall therefore be the strategy.

2.2.2 NLP Solution Strategies

There are several strategies how to solve an NLP problem. Two of the most powerful ones (see market Study, output of WP 1100), SQP and IP, will be considered within this activity. These shall be described in the following.

2.2.2.1 General Remarks and Background

The Lagrange function defined by

$$L(x,\lambda,\mu) \quad = \quad f(x) + \lambda^{\top} g(x) + \mu^{\top} h(x)$$

with multipliers $\lambda = (\lambda_1, \ldots, \lambda_m)^T \in R^m$ and $\mu = (\mu_1, \ldots, \mu_p)^T \in R^p$ plays an

important role for all optimization methods in order to solve (NLP).

Essentially all algorithms work iteratively and construct sequences $\{(x^k, \lambda^k, \mu^k)\}$ which converge to a point $(\hat{x}, \hat{\lambda}, \hat{\mu})$ which satisfies the first order necessary KKT conditions:

$$\begin{aligned} \nabla_x L(\hat{x}, \hat{\lambda}, \hat{\mu}) &= 0, \\ \hat{\lambda} &\geq 0, \\ \hat{\lambda}^\top g(\hat{x}) &= 0, \\ g(\hat{x}) &\leq 0, \\ h(\hat{x}) &= 0. \end{aligned}$$

Practical implementations are often designed for problems with lower and upper bounds on the constraints and treat nonlinear constraints, linear constraints and box constraints separately (see section 2.2.1).

The following figure shows how a NLP-method basically works:

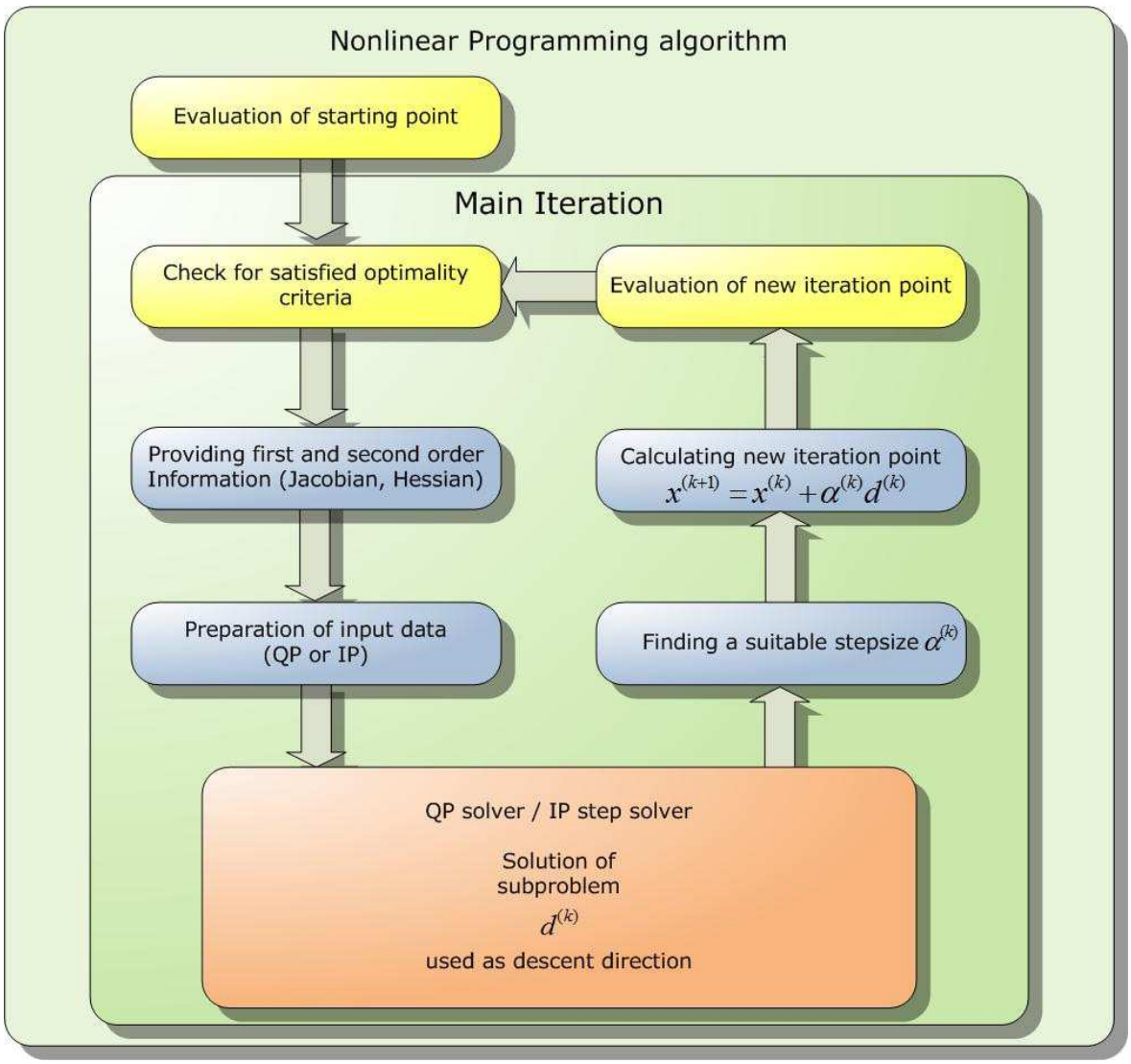

Figure 2-4: Nonlinear Programming algorithm

2.2.2.2 Sequential Quadratic Programming

One of the two basic NLP methods that will be implemented is the Sequential Quadratic Programming (SQP). SQP methods are still frequently used and belong to the most powerful algorithms in particular for highly nonlinear problems.

The basic idea of a SQP method is to formulate and to solve QP subproblems. The QP subproblems are local approximations of the Problem (NLP). SQP-methods consist of primary and secondary iterations. The secondary iterations are used to solve the QP subproblems. In the primary iterations the SQP

method uses a (quasi) Newton method to calculate a sequence of iterates $\{(x^k, \lambda^k, \mu^k)\}$ which converges to the optimal solution $(\hat{x}, \hat{\lambda}, \hat{\mu})$. The new iterates are calculated by the following formula

$$x^{(k+1)} = x^{(k)} + d^{(k)}.$$

Herein, $d^{(k)}$ denotes a search direction. This search direction is the solution of the QP subproblem. To achieve convergence from any starting point one has to use globalization strategies (see requirement eNLP-F-003). For example,

line-search methods,

filter methods or

trust region methods.

If line-search is used the calculation for the next iterate changes to

$$x^{k+1} = x^k + \alpha_k d^k, \qquad k = 0, 1, 2, \ldots$$

Herein, $\alpha_k > 0$ denotes a suitable step-size. For more details concerning line search see section 2.2.5.1.

The different ways how to solve the QP subproblems efficiently, how to tread inconsistent QP subproblems, how to globalize the algorithm (line-search, trust-region, filter methods) and how to regularize the problem are an ongoing research area. The most promising new latest results of research will be incorporated in the SQP module of the solver. For more details see section 2.2.4.

Most of the existing SQP solvers work only with small and dense solvers. Therefore the theory about SQP method is mainly related to this kind of problems. As the solver developed in this activity shall be able to solve large scale problems and to exploit their sparsity, the applied methods can differ to classical theory (see Current Trends, output of WP 1200).

2.2.2.3 Interior-Point

The other method that will be considered is an interior-point method.

The idea of interior-point methods is to eliminate inequality constraints from (NLP) by adding them to the objective function using a barrier function. For illustration, we will use the most common barrier function, the negative logarithm. The typical interior-point approach to NLP looks as follows. Firstly, non-negative slack variables are introduced in (NLP) for the inequality constraints transforming them to equality constraints:

$$g_i(x) + s_i = 0, \qquad s_i \geq 0, \qquad i = 1, \ldots, m.$$

Then, the restrictions $s_i \geq 0$ are eliminated and the standard NLP problem is replaced by the barrier problem

$$(NLP_{IP}) \qquad \begin{aligned} \text{Minimize} \quad & f(x) - \eta \sum_{i=1}^{m} \log s_i \\ \text{subject to} \quad & g_i(x) + s_i = 0, \quad i = 1, \ldots, m, \\ & h_j(x) = 0, \quad j = 1, \ldots, p. \end{aligned}$$

Now, roughly speaking, this problem is solved for a sequence $\eta_k \downarrow 0$ by a Lagrange-Newtontype algorithm which aims to solve the KKT conditions of (NLP_{IP}) by Newton's method. For this problem the KKT conditions have to be satisfied. As already discussed in "Future Development" (output of WP 1210), these conditions can be formulated in three different forms.

All these approaches are equivalent from a mathematical point of view, but the three linear systems differ substantially in their size. Current interior-point methods differ in the way which of these systems is actually being solved and by which method (iteratively or by direct factorization).

A current field of research is the development of preconditioners for iterative solution methods which exploit the structure of the different KKT systems. For a more detailed discussion about current research in this field see again, "Future Develoment", output of WP 1210.

2.2.3 Recommendation of Solvers to Develop

One of the main strategic concepts of the developed solver is to provide many different modules on every level of the solver. This point has been discussed in more detail in section 2.1.1. As a result of the idea of modularity the requirement eNLP-F-002 has been formulated saying that two different solution methods shall be developed.

The Market Study of NLP solvers (WP 1100) has shown that there currently exist two European solvers being developed that fit in the requirements. These two solvers will be introduced in the following.

2.2.3.1 WORHP

WORHP ("**W**e **O**ptimize **R**eally **H**uge **P**roblems") is a very young solver. Its developer team consists of young researches whose aim is to develop a solver that does not play with purely mathematical problems but is oriented at the application.

It has been build very modulary from the beginning and allows new methods and modules to be inserted very easily. But this is not the only reason for its modularity.

A *good* solver cannot be measured clearly. It should solve many problems which means it should be robust. This should be done as quickly as possible, i.e. it should be fast. Its results should be accurate and it should be easy to use. Maybe one is interested in a good balance of all this.

Hence, the philosophy of WORHP is to provide as many different functionalities as possible, leading to different advantageous properties of the solvers. Already now WORHP incorporates many different methods that can be chosen by the user so that they fit into his preferences and to his single problem.

Furthermore it incorporates the so called Reverse Communication (see section 2.1.2.2) which allows the best possible interaction of the user who wishes to do so. It uses the compressed column matrix storage format which is most commonly used and as described in section 2.4.2.1.1, is superior to other storing formats.

WORHP works with a SQP method. While many solvers expect the user to provide first and second order derivatives, a user of WORHP need not care about it. WORHP attempts to provide robust, accurate derivative routines with as little computational effort as possible. Of course this cannot always be achieved at once, so that again there are and will be provided several modules satisfying the preferences of the user. WORHP uses FD-methods (see section 2.3.2.1.2), so that arbitrary functions can be handled. Of course, the user is free to provide derivative information.

Although the exact Hessian matrix leads to the best convergence properties, it is not always wished to use it for the approximation and solution of the problem. WORHP provides various functionalities how to deal with the Hessian matrix. This can be the exact Hessian, but also the identity matrix -resulting in a Sequential Linear Programming (SLP) method-, a diagonal BFGS or only the diagonal of the Hessian.

WORHP will be continuously further developed, improving robustness, computation efficiency, accuracy and of course modularity.

2.2.3.2 IPFilter

The code IPFilter is an implementation of an interior-point filter method for Nonlinear Programming. This method belongs to the class of the Newton primal-dual interior-point algorithms.

It incorporates a filter technique (see requirement eNLP-F-004) and a line

search for the purposes of globalization. It relies on a new decomposition of the primal-dual step obtained from the perturbed first-order necessary conditions into a normal component and a tangential component.

The normal component can be seen as a step towards the quasi-central path (the set of strictly feasible points which are central) whereas the tangential component aims at reducing duality (norm of the gradient of the Lagrangian) and complementarity.

The line search acts on both components. The new iterate must also lie on a neighbourhood of the quasi-central path (used frequently in infeasible primal-dual methods for linear and quadratic programming). Each entry in the filter is a pair of coordinates: one resulting from feasibility and centrality (associated with the normal step); the other resulting from optimality (complementarity and duality) and related to the tangential step.

It has been proved that the method possesses global convergence to first-order critical points. The method incorporates a restoration phase and all new iterates must necessarily be acceptable to the filter.

This approach and the corresponding implementation of IPFilter had been shown to yield promising numerical results. The robustness rate on a set of over 450 problems of the CUTEr collection is now above 93% (meaning that only less than 7% of the problems caused early termination or exceed the maximum number of iterations). For the problems solved successfully the average number of iterations is on the order of 25.

So, we have a solid ground basis for the development of an efficient and robust NLP solver, and to offer a complementary alternative to the main solver developed under this ESA project. However, the current IPFilter approach has a number of limitations and there is still an enormous amount of work ahead to improve the current status.

2.2.4 QP Solution Strategies

2.2.4.1 Problem Formulation

The main effort in the line-search SQP method is to solve the occurring quadratic programs. Without loss of generality we consider

$$(QP) \qquad \min_{x\in\mathbb{R}^n} \frac{1}{2}x^\top Hx + c^\top x \qquad \text{s.t.} \qquad \tilde{g}(x) := Ax - u \leq 0, \quad \tilde{h}(x) := Bx - v = 0.$$

Solving non-convex quadratic programs efficiently is a difficult task and in the

context of trust-region methods, the problems are often solved approximately only.

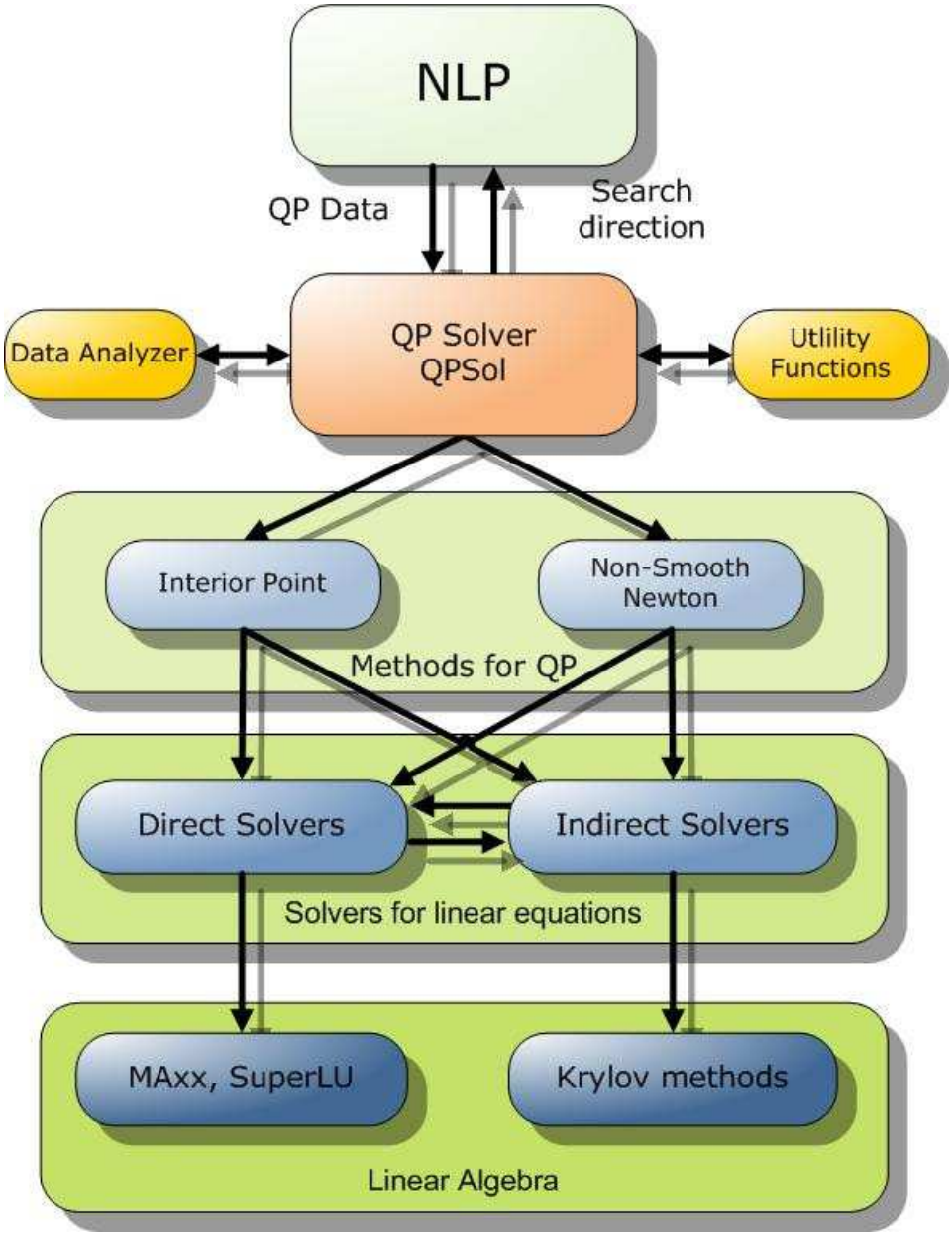

Figure 2-5: QP Solution Strategy

2.2.4.2 QP Solution Methods

According to the concept of modularity, the developed QP solver will consist of various methods. We provide an overview about different strategies that will be executed in order to solve the quadratic problems.

2.2.4.2.1 Primal-Dual Interior-Point Method

Similar as for (NLP) we can apply the interior-point idea to (QP) and obtain the so called barrier problem. Writing down the KKT-conditions for the barrier problem leads to a nonlinear equation system. From one of these equations, one obtains the complementarity measure which denotes the error in the complementarity conditions for (QP). So, the KKT conditions to the barrier problem can be viewed as perturbations of the KKT conditions to (QP).

Primal-dual interior-point methods generate primally and dually feasible multipliers such that the complementarity measure tends to zero.

In case of convergence the KKT conditions are satisfied at the limit point and (QP) is solved provided (QP) was convex.

Notice that this procedure does not require an initially feasible point. Gertz and Wright developed an algorithm that uses a heuristic of Mehrotra which was originally designed for linear programs and turned out to be successful. Firstly, an affine scaling step is computed which corresponds to a Newton step for $\eta = 0$. Secondly, a corrector step is applied which aims at improving the complementarity measure.

2.2.4.2.2 A Nonsmooth Newton Method

The idea of the nonsmooth Newton method is to solve the KKT system of (QP) directly. The KKT system for (QP) is reformulated into an equivalent nonlinear equation system

$$F(z) = 0.$$

Unfortunately, F incorporates a nondifferentiable function. A generalization of Newton's method for nonsmooth F has been formulated. The non-existing Jacobian $F'(x)$ is replaced by a set-valued generalized Jacobian $\partial F(x)$ which has a certain well-investigated structure.

Replacing the non-existing Jacobian of F in the classic Newton method by an arbitrary element of the generalized Jacobian leads to the nonsmooth Newton method.

It is possible to globalize the nonsmooth Newton method and a quadratic convergence rate for strictly convex quadratic problems can be obtained, see future development (output of WP 1200).

The structures of the occurring linear equations are very similar to the interior-point method. Unfortunately, the generalized Jacobian matrix is not symmetric which makes the numerical solution of linear equations involving this matrix more expensive. Nonsmooth Newton methods currently are not very well investigated in view of their practical performance for convex quadratic programs.

However, preliminary results suggest that they show a similar performance as interior-point methods. It seems as if nonsmooth Newton methods do not show the ill-conditioning with respect to parameter settings which sometimes can be observed for interior-point methods, but this conjecture has to be investigated in more detail.

2.2.4.2.3 QP Equality Module

The QP equality module solves equality constrained QP problems only. In this special case only the linear equation system has to be solved.

2.2.4.2.4 Projected CG Method

The projected CG method (with a slight modification) is used in KNITRO to compute the tangent step within the QP subproblem of the trust-region SQP method. It is particularly well-suited for large-scale and sparse quadratic problems with equality constraints.

This method shall be implemented in the future. See “Future Development” for more details.

2.2.4.3 Treatment of Non-Convex Quadratic Programs

Treatment of non-convex quadratic problems is a challenging task as these problems may have local and global minima. For large-scale problems with many degrees of freedom range-space and null-space methods cannot be applied anymore without further ado.

There exist algorithms for non-convex quadratic programs as they appear in trust-region SQP methods. Their approach uses different methods. A preconditioned projected conjugate gradient method is used to solve equality constrained problems. An interior-point method is used for problems with inequalities. The inner iteration uses a trust-region strategy.

Recently, Friedlander and Leyffer proposed a method for general non-convex quadratic programs. The method combines gradient projection, augmented Lagrangian techniques, and filter methods. Global convergence and finite termination was established. It is suitable for large-scale problems with many degrees of freedoms.

2.2.4.4 Development Goals

Clearly, the overall goal of the QP solver development is a further improvement of the reliability and performance of the existing QP solver. This will be achieved by the following actions:

The improvement of the preconditioning techniques is one aim of the project. Iterative conjugate gradient based solvers for sparse linear equations and efficient preconditioning strategies for KKT-type saddle point systems (this is a current field of research) will be investigated.

The strategies of WP3510 for treatment of singular or numerically singular linear equations or badly scaled problems will be used. Using these regularization strategies we will extend the QP solver towards problems with rank deficiencies in the constraints or the Hessian which enables us to test the solver for the complete test-set of Maros and Meszaros.

Another goal is to extend the solver towards non-convex problems. Recently two suitable methods for indefinite QP's will be investigated. The first method is a gradient projection method which uses an augmented Lagrangian and a filter technique to achieve global convergence.

The second approach uses an interior-point approach and a projecting conjugate gradient method in combination with a trust-region technique for promoting global convergence towards second order critical points.

A thorough testing of these codes in view of their eligibility for the NLP solver has to be performed within this project. The advantage of these codes is the ability to solve non-convex QP's.

Interior-point methods tend to suffer from ill-conditioning in the vicinity of first-order critical points. Controlling this ill-conditioning by tuning the parameters of the algorithm is one of the major strategies.

There will be close connections to the NLP development as IPFILTER might be suitable for the solution of QP's as well. Owing to the desired modularity of the NLP solver, a quick combination of different solvers won't be problem.

Similarly, the components on linear algebra level can be exchanged.

2.2.5 Measuring Progress

According the different NLP methods, there are different strategies how to define or when to accept a new iterate.

In the following we describe these different strategies. A line-search is a classical step in a SQP method although there also exist filter SQP methods and hence the different measuring strategies can modularly be combined with dif-

ferent NLP methods.

2.2.5.1 Line-search

Line-search oriented solvers compute the next iterate by the formula

$$x^{k+1} = x^k + \alpha_k d^k, \qquad k = 0, 1, 2, \ldots$$

Herein, d^k denotes a search direction and $\alpha_k > 0$ denotes a suitable step-size.

The search direction d^k in step k of the NLP method is given by the solution of the

following quadratic program (provided it is solvable).

Once the search direction d^k is determined, the step-length α_k is computed by an approximate one-dimensional minimization of a merit function along the $x^k + \alpha^k d^k$ with respect to $\alpha_k > 0$.

Common merit functions are the L_1-penalty function or the augmented Lagrangian function

Herein, the penalty parameter has to be adjusted appropriately in order to guarantee d^k to be a direction of descent of the respective merit function. Notice that the L_1-penalty function is only directionally differentiable but not continuously differentiable. The augmented Lagrangian function is continuously differentiable. Both merit functions are exact (if a suitable constraint qualification holds), which is an important property in view of the boundedness of the penalty parameter.

The introduction of a line-search procedure is necessary to achieve convergence from remote starting points (global convergence). From a purely theoretical point of view the exact Hessian of the Lagrange function is a very good choice for H_k because a locally quadratic convergence rate can be obtained under suitable assumptions. If H_k is determined by a modified BFGS-update rule, a superlinear convergence rate can be obtained, and the QPs are strictly convex as H_k is positive definite, provided H_0 was positive definite.

2.2.5.2 The Filter Idea

Recently, filter methods became increasingly popular. The filter idea avoids merit functions. The idea of filter methods is to measure the progress in the objective function and the constraint violation separately and not by a single merit function which combines both measures.

The method uses ideas from multi-objective optimization and Pareto optimality. A filter is a set of pairs (f^k, c^k) such that no pair dominates another pair in the filter, that is no pair is better than another pair with respect to both, the objective function value and constraint violation. Herein, f^k and Hc^k denote the objective function value and the constraint violation (measured in a suitable norm), respectively, at point x^k.

A new point will be accepted by the filter if it is not dominated by any element in the filter. New candidates for the filter can be generated for instance by solving suitable QP subproblems. There are some crucial points in implementing filter methods which have to be obeyed in order to guarantee global convergence.

Numerical investigations suggest that the filter method is less restrictive than using merit functions in view of the acceptance of new iterates and this sometimes turns out to be advantageous.

A main issue in filter methods is the definition of the filter components. Within this activity, alternative definitions for the optimality filter measure that better reflect the purpose of minimization and become closer to the optimality filter measure used in SQP methods will be investigated.

2.3 NECESSITIES STRATEGIES

2.3.1 Sparsity Definition

On the one hand the solver should be very easy to use, also for inexperienced users. On the other hand, the solver should be as efficient as possible. In some point these points contradict each other. However, thanks to the modular concept, different strategies regarding the sparsity information can be implemented.

If the user is not able to provide any sparsity information, the solver will still work. However, the price to pay will be a much higher computational cost and sometimes also a lower accuracy.

If the user does provide a sparsity information, it need not be the minimal pattern. Obviously the better the pattern is the better the performance will be. The sparsity pattern the user should be encouraged to provide is the pattern of the gradient of the objective function and the Jacobian matrix of the constraints. According to requirement eNLP-U-010, this shall be done in compressed column format.

Only a user with knowledge in the area of optimization will be able to provide the sparsity pattern of the Hessian of the Lagrangian. Therefore there will be a routine that determines this pattern from the information given about the gradient of the objective function and the Jacobian of the constraints. Note that the resulting pattern is only a worst case pattern and the true sparsity can be much higher than the one resulting from this routine.

According to requirement eNLP-U-012, the sparsity information provided by the user can be checked so that an entry that the user might have erroneously declared as zero entry turns out not to be zero. Unfortunately such a mistake cannot be detected in every case but if such a check claims that there is one there will be one. Such a check is foreseen to be part of the solver.

2.3.2 Derivatives

As described below, there exist and will be implemented several strategies to determine the derivative information. The most sensible and therefore the standard way is to use finite differences (see section 2.3.2.1.2) and exploit the sparsity. If one expects the structures to be full, i.e. that there are no zero-entries at all, the solver will work but the computation time will be unnecessarily high and the coloring methods cannot be applied.

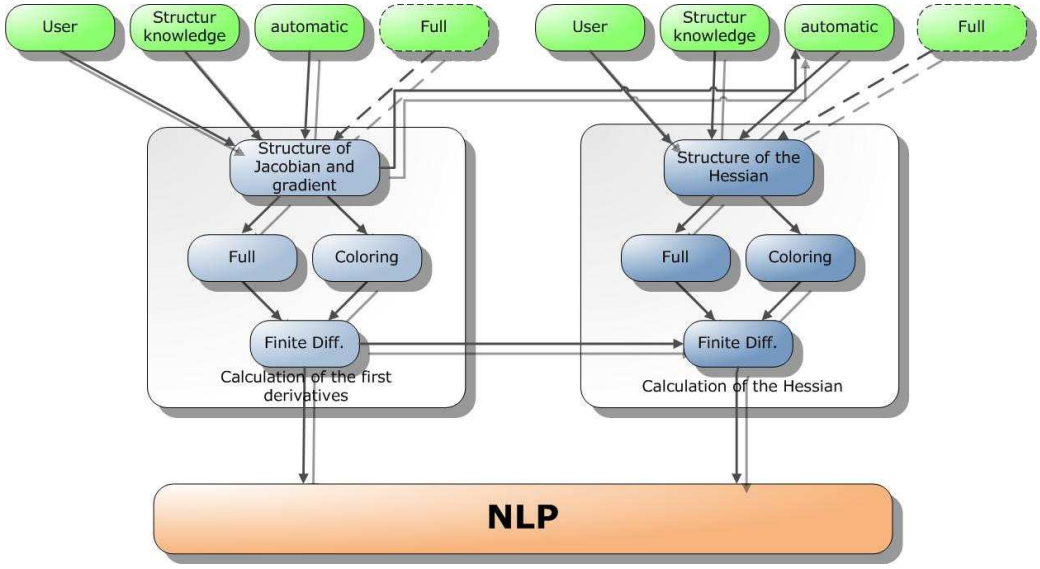

Figure 2-6: Derivative calculation

As discussed in section 2.1.3, the user shall be encouraged to provide the sparsity information of the gradient of the objective function and the Jacobian matrix. Within this activity there will only be an automatic check if this sparsity has been given correctly. In some cases, such as in transcription methods of optimal control problems, there exists some structure knowledge that can be provided for an arbitrary problem without user input.

Using finite differences, the best way how to exploit the sparsity is to apply the graph coloring groups. However providing the sparsity information is not only important for the derivative calculation and should therefore also be done if no graph coloring method is applied.

As also mentioned in section 2.1.3, the structure of the gradient of the objective function and the Jacobian matrix can be used to automatically determine the Hessian matrix structure. However, a detailed user provided sparsity cannot be achieved by this automatism. A full Hessian is not very advisable as in large scale problems this will lead to an enormous memory and computation effort.

The general strategies how to determine the derivative information will be described in the following.

2.3.2.1 Strategies

2.3.2.1.1 Provided Derivatives

Of course, it is the simplest way just to expect the user to provide all derivatives analytically. If the problems have large dimensions, providing analytical deriva-

tives implies lots of work for the user which is often unacceptable. The situation is even worse since unfortunately in most cases the functions (objective, constraints) themselves are not given analytically. This is the case e.g. if a function is given by numerical computations/routines.

However, if the user is able to provide derivatives, there will be a module that supports that (see requirement eNLP-F-009). This will lead to a higher performance with respect to both accuracy and computation time.

2.3.2.1.2 Variable Order Finite Differences

The Finite-Differences (FD) methods work with every problem (which has to be differentiable, of course). Therefore they have to be implemented for generic problems.

However, the calculation is known to be numerically instable and their numerical accuracy is worse than other methods. Because of their general applicability, these methods will be chosen as a central point for the calculation of the derivatives in the NLP solver (see requirement eNLP-F-007).

The precision of the computed first and second order finite differences is crucial to the performance and stability of the optimization process. Currently WORHP uses a forward difference algorithm to approximate derivatives. Modules with using higher order finite difference methods will be implemented which are more expensive but also notably more precise.

There are several strategies to calculate FD-derivatives whose accuracy depends strongly on how many function evaluations are taken and this determines the computation time of each method. Possible methods are

Forward Difference

Backward Difference

Central Difference

Weighted Difference

Implementing a number of different order finite difference methods not only enables the solver to solve increasingly difficult problems, but also enables the user to choose a suitable method for his problem. Dynamically using methods of different orders can also speed up the optimization process, or make it possible to recover from errors caused by insufficient precision of the problem approximation.

2.3.2.2 Sparsity Exploitation

In order to cope with large dimensions, one has to exploit the sparsity when-

ever possible. This holds especially for the determination of derivatives with FD. A value of a derivative is known in advance to be zero when a function does not depend on a certain variable. If the sparsity information is provided this has to be respected whenever possible.

Unfortunately, considering the functions whose derivatives have to be determined, this arises not always to be as simple as it might seem at the first place. These functions will be presented in the following sections.

2.3.2.3 First Order Derivatives

2.3.2.3.1 Objective Function Gradient

The gradient of the objective function is not as hard to determine. One only has to check whether a variable has an influence on the objective function or not. If this is not the case, nothing has to be done and if it is, one applies the chosen FD method.

2.3.2.3.2 Constraint Jacobian

The Jacobian matrix of the constraint vector is generally more difficult to determine. Often a single constraint cannot be evaluated. However, if this is possible, one can proceed analogously to the objective function.

In general one has to apply a more sophisticated method based on the graph theory. This is described in the following section.

2.3.2.3.2.1 Full Derivative Determination

Ignoring the sparsity, the derivative calculation will lead to huge computation time. Every optimization variable has to be perturbed and each function has to be evaluated. It is obvious that this might not be the most sensible way.

2.3.2.3.2.2 Classical Sparse Derivative Calculation

If a variable has no influence on a certain function the corresponding function evaluation can be omitted. Unfortunately, in many cases, it is not possible to evaluate one single constraint. Therefore only with respect to the gradient of the objective function, evaluations can be omitted. Regarding constraints, one can only save computation time in the case where there exist an optimization variable having no influence to one single constraint. This case is very unlikely and therefore the sparsity exploitation is quite poor.

2.3.2.3.2.3 Graph Coloring Groups

By solving NP-hard graph-colouring problems, variables can be grouped in such way that only a (sometimes tiny) fraction of the number of function evalua-

tions of the standard method is necessary to compute the finite differences.

In optimal control problems using certain discretization techniques, this number is even constant with respect to the discretization, while the standard method has a complexity of O(n).

The so called “group strategy”, a more sophisticated method which allows to generate several columns of a Jacobian or Hessian while only one evaluation of the considered vector will be applied. Each column of such a matrix corresponds to an optimization variable. Many variables are allowed to be in the same group if no constraint is dependent on more than one of them at the same time, allowing the variables of each group to be evaluated together, instead of consecutively.

The group strategy can save considerable amounts of computational time. In a sparse problem with 100,000 optimization variables, every evaluation of the Jacobian needs 100,000*k function evaluations. k denotes the number of evaluations needed for one FD-computed value and depends on the particular FD method used to approximate the Jacobian. Usually, k will range between 1 and 4.

The determination of the best group collections is extremely costly. However, even with a suboptimal group determination, in such a case, the number of groups can easily be less than 20. This means that one only needs (less than) 20*k evaluations in each iteration which indicates an enormous saving potential.

The crucial task in the group strategy is to build the groups of columns that can be evaluated in one step. However, the problem to find the minimum number of groups is NP-hard which means that the computational time for solving the problem grows exponentially with the number of variables. Hence one has to find a good heuristic that finds a low number of groups in a reasonable amount of time.

As the structures of the matrices are usually not going to change within the optimization process, one idea is to build these groups only once, at the beginning of the optimization.

Therefore, to this end, it will be necessary to establish a preanalysis. With respect to the computational time, it might be sensible to include this preanalysis in the Hessian structure determination which also has to be done only once at the beginning of an optimization process.

It will probably depend on the application, what “reasonable amount of time” actually means.

If one likes to solve many similar problems with the same sparsity pattern, it

might be sensible to use a method with a quite high computational time but generating excellent results. For this whole problem class the group detection has to be done only once and can be exploited every time one has to solve such a problem. Such a method could be the so called “Smallest-Last-Ordering-Algorithm with Exchange Method”.

On the other hand, for a standard problem, one might be interested in an algorithm like the “CPR (Curtis Powell and Reid)-Algorithm” which guarantees a lower computational time and still reduces the computational time considerably.

There are many different heuristics, how to determine the number of groups. Without claim of completeness, the following shall be mentioned:

CPR (Curtis Powell and Reid) –Algorithm

LFO (Largest First Ordering) –Algorithm

SLO (Smallest Last Ordering) –Algorithm

IDO (Incidence Degree Ordering) –Algorithm

The Exchange Method is an add-on that generally improves each of these methods. Of course, the price to pay is a higher computational time.

All these methods will form a module in the solvers developed within this activity.

One task will be to detect which methods are suitable for WORHP, IPFilter and the ASTOS software, and which method shall be suggested for certain applications. It is obvious that the possibility to choose a group algorithm will enlarge the modularity.

A central point will be to implement the several group strategies and to speed up and stabilise the numerical calculation of the derivatives. As this task reaches deeply into mathematical graph theory, which is normally far away from NLP development, this is expected to claim a lot of work.

2.3.2.4 Second Order Derivatives

According to the requirements eNLP-F-006 and eNLP-F-008, the solver will have various robust strategies how to determine or approximate the Hessian.

2.3.2.4.1 Identity

The simplest method to provide an approximation of the Hessian will be just to use the identity matrix instead of the Hessian. Of course, this is far away from a good approximation. On the other hand no computation time has to be expended. This leads only to linear convergence but sometimes it has been shown that the identity matrix is better than an approximated Hessian with a

tiny error due to numerical instabilities.

A module will be provided that allows choosing the identity matrix for single iterations or even for the whole optimization.

2.3.2.4.2 Diagonal BFGS

Dense solvers use often BFGS updates instead of the Hessian matrix. These lead not to as good convergence properties as methods using exact Hessian information, but as the computational time is much lower, it is often worth doing so. This seems not to be the case with sparse solvers.

Classical BFGS methods generally lead to dense matrices, even if the Hessian matrix is sparse. Therefore, with these BFGS updates, one cannot exploit the sparsity structure during the optimization process and, additionally, one will end up with a huge memory requirement.

Limited memory-BFGS updates are able to reduce the memory requirement but work with dense matrices internally. There are current investigations about sparse BFGS updates. If future shows that these perform well they might become a part of our Hessian in the future.

For problems that have a Hessian matrix whose structure is close to diagonal, there will be a module with a so called diagonal BFGS. This is a matrix basically computed with the BFGS update formula but ignoring every nondiagonal entry.

2.3.2.4.3 Exact Hessian Matrix

The use of the exact Hessian leads to the best possible convergence property. The only reason why this is not always applied is that this matrix is expensive to determine. In this respect *exact* means that theoretically the values should be the real values. In fact due to rounding and computational reasons, there will always some inaccuracy.

The general method to compute the Hessian matrix is generally the same as for the first derivatives, namely FD.

2.3.2.4.3.1 Full Hessian Determination

Again, it is not very sensible to calculate each entry of the Hessian even if it is known to be zero.

2.3.2.4.3.2 Classical Sparse Hessian Determination

For the Hessian determination, the classical sparse derivative calculation is more sensible than for first order derivatives. If a variable has only a linear influence to all functions, a whole column of the Hessian is known to be zero.

However, many zero entries will be computed unusefully.

2.3.2.4.3.3 Graph Coloring Groups

Although a Hessian is basically a special case of a Jacobian, the saving concerning the Hessian is probably even higher, as it seems that one can exploit the advantage of the group strategy twice. By exploiting the sparsity with applying the group strategy and various points, the determination of the derivatives will be done with as little computation time as possible and will lead to excellent convergence properties of the solvers.

2.3.3 Linear Algebra

2.3.3.1 Solving Sparse Equations

The main computational effort in the above optimization algorithms is consumed for the solution of large-scale and sparse linear equation systems which have a certain structure. There are two general approaches to solve such systems.

2.3.3.1.1 Direct methods

The direct factorization approach essentially computes an LU-factorization of the matrix taking into account the sparse structure of the system. Column and row permutations are used to minimize the fill-in in the factors of the matrix. Popular methods are the multifrontal codes MA-XX (e.g. MA47,MA27 etc) or SuperLU.

2.3.3.1.2 Iterative Methods

The iterative solution methods are based on either fixed-point iteration (e.g. Gauss-Seidel, Jacobi, successive overrelaxation) or Krylov subspace methods like conjugate gradient methods.

In the context of optimization problems one usually prefers conjugate gradient methods (compared to fixed-point iteration) in combination with a suitable preconditioning technique. Preconditioning is the key issue in applying conjugate gradient methods successfully. In the meanwhile there exist several versions of conjugate gradient methods (CGS, CGS2, BI-CGSTAB, GMRES, QMR) which are tailored to the properties (symmetric or non-symmetric, positive definite or not) of the linear equation under consideration.

2.3.3.2 External Libraries

The aim for the long term perspective is to have a standalone NLP solver. During the development, it might be inevitable to use some commercial libraries. Until the end of this activity, there should at least be the possibility to avoid

such libraries, being aware of the fact that the performance might not be always as good as with those libraries(see requirements eNLP-U-019 and eNLP-U-020).

2.3.3.3 Preconditioning

The memory requirements for direct factorization methods are high while iterative methods have minimal memory requirements (which may not be true for the preconditioner). The performance of iterative methods highly depends on the preconditioner.

Preconditioning is used to improve the condition of the matrix which results in a faster convergence. Constructing a preconditioner is highly problem dependent. Several general strategies are known such as incomplete LU or incomplete Cholesky decomposition. However, these standard preconditioners seem to be not very successful for general optimization problems.

The current trend is clearly to construct preconditioners for the special structure of the matrices which occur in interior-point methods, nonsmooth Newton methods, or active-set methods. As outlined above, all these optimization approaches essentially lead to matrices with a very similar structure (KKT-matrices or KKT-like matrices).

Unfortunately, these KKT matrices have a saddle-point structure and are indefinite. To tackle this problem, another matrix is determined. The product of these two matrices have much more preferable properties and a conjugate gradient method will perform much better. The computation of this second matrix requires a direct factorization method.

Summarizing, currently and in the future both approaches (direct factorization and iterative methods) will play an important role. Their combination promises the most success. Each approach for itself is limited: direct factorization needs too much memory, iterative methods need a preconditioner, which in turn may require direct factorization.

If an approximate solution of a linear equation is computed by direct factorization, then iterative refinement can be used to improve the solution and to reduce the residual error.

2.3.4 Outputs

As already mentioned in section 2.1.2.2 one of the main strategies of the solver is to allow interaction between the user and the solver. For this strategy it is essential that the experienced user gets all the information from the solver that he requires for his desired interaction.

On the other hand an inexperienced user should not be bothered with informa-

tion which is not useful for him.

At any time of an optimization process, the user will be able to get any information he likes. He will be free to write a routine that gives him (only) the information he wants in the form he likes.

Of course, there will be several standard outputs provided by the solvers (see requirements eNLP-U-013, eNLP-U-014 and eNLP-U-015). The inexperienced user will be able to choose the standard output including only the most necessary information.

2.3.4.1 Process Monitoring

According to eNLP-U-016, in each iteration, the user is able to watch the progress of the optimization. This means he can see the value of the objective function, the norm of the violated constraints and the KKT-value that measures optimality.

Depending on the chosen solver, more values then these shall be given, such as the values like the step size, the norm of the search direction or the value of the penalty parameters.

2.3.4.2 Abortion Outputs

When the solver terminates, the most important output is the reason why it did. There will be a variable that stands for the reason of the abortion. This reason can be read on the output so that no manual is necessary to understand it. Possible reasons are

Optimal solution found

Optimal solution found, but not with the requested accuracy

Minimal step size reached

Could not find search direction

…

Furthermore, independently of the abortion reason, the output will certainly contain the objective function value and the constraint violation (if any) and the KKT value which is crucial for the optimality check. Optionally, the output can also contain all the optimization variables. This might not always be wished, especially if the number of variables is huge.

Furthermore, there will be an option returning all the chosen parameters that have been used.

2.3.4.3 Error Outputs

In case something has gone wrong, it is important to provide the user with as many information as possible to enable him to make it better in the second run. Hence the output shall tell the user as detailed as possible at which point something must have gone wrong (see requirement eNLP-U-018). This is not always possible as the point of the algorithm where it stopped must not necessarily be the point where something has gone wrong.

The consortium is aware of this problem. Hence there is running another activity between *SRC Optimization and Optimal Control, Astos* and the german aerospace agency *DLR* and that analyzes every failed run of the algorithm and is supposed to return a detailed proposal which parameters to change in order to obtain a satisfying solution.

2.3.5 Test-sets

Test-sets help to identify strength and weakness of an NLP solver. This is not only important for the selection process in order to define the solvers. Test-sets are a major part of the development strategy as they form a good measure for the continuous improvement and functionality of the solvers.

2.3.5.1 Hock-Schittkowski

The Hock-Schittkowski library (see requirement eNLP-P-005 and eNLP-P-006) is a collection of 306 optimization problems. Its intention is to present an extensive set of nonlinear programming problems that were used by other authors in the past to develop, test or compare optimization algorithms.

The new collection of this library which will be used in this activity has a special emphasis on “real world” problems. The Hock-Schittkowski library incorporates only dense problems with up to 100 variables. Many of these problems are artificial and bring any NLP solver to its limits.

However, it is an excellent choice for testing methods and functionality of a sparse solver, as well. Furthermore, as it is written in Fortran, it has a very easy interface and can comfortably be used.

Hence this testset will be the standard test-set to evaluate the solver.

2.3.5.2 CUTEr

CUTEr is a versatile testing environment for optimization and linear algebra solvers. The package contains a collection of test problems, along with Fortran 77, Fortran 90/95 and Matlab tools intended to help developers design, compare and improve new and existing solvers.

The test problems provided are written in so-called Standard Input Format

(SIF). A decoder to convert from this format into well-defined Fortran 77 and data files is available as a separate package. Once translated, these files may be manipulated to provide tools suitable for testing optimization packages. Ready-to-use interfaces to existing packages, such as MINOS, SNOPT, filterSQP, KNITRO, and more, are provided.

CUTEr is available on a variety of UNIX platforms, including Linux and is designed to be accessible and easily manageable on heterogeneous networks.

The default size of the CUTEr problems has been raised in an attempt to reflect the increasing size of modern problems.

2.3.5.3 General Test Examples

Every upcoming problem will be welcome to test the functionality of the developing solvers. The industrial partners will continuously work on testing the solvers on their own applications and will therefore ensure the practicability of the solvers (see requirement eNLP-P-007).

2.3.5.4 QP Test-sets

The QP solver is in the core of the SQP solver. The main part of the overall computation time will be accumulated during the numerical solution of the quadratic programming subproblems.

Depending on the origin of the NLP, e.g. trajectory optimization, these quadratic subproblems will be large-scale with a sparse structure. Even for dense problems, the resulting QP has a certain structure depending on the algorithm. For instance, an interior-point method leads to linear equations with a KKT-type matrix, a symmetric block matrix with zero blocks and a diagonal block. Similar structures occur for nonsmooth Newton methods.

Therefore, the largest gain in view of computational performance beside the derivative computation is achievable in the QP solver if efficient sparse numerical linear algebra is employed. Thus, a thorough individual test campaign for the QP solver is strongly advisable. This test campaign will be independent of the superordinate SQP solver.

The QP solver will be tested and validated for a collection of convex quadratic programming problems from the test-set of Istvan Maros and Csaba Meszaros.

The test-set contains a total of 138 problems and is available online.

The test set of Maros and Meszaros is coded in a special format, the QPS format, and requires a decoder. The decoder transforms the data given in QPS format into the form required by the QP solver.

The test set of Maros and Meszaros will be used throughout the whole project

to validate the QP solvers. This is not only beneficial in view of monitoring the improvement of the performance but it is especially useful in the debugging process.

The results will be compared and evaluated in view of accuracy, robustness, and runtime to those obtained from a reference solver like the Matlab QP solver. The Matlab QP solver offers a medium-scale version based on an active-set method and a large-scale version based on a Newton-type interior-point method.

2.4 SOFTWARE STRATEGIES

2.4.1 Programming Language Definition

The computational core components of the solver will be implemented in Fortran conforming to the ISO/IEC 1539:1997 standard ("Fortran 95"). New language features from the new ISO/IEC 1539:2004 standard ("Fortran 2003") may be used, where this is considered beneficial, if the stable release of the GNU Compiler Collection available at the time of development already supports the features under consideration.

Non-computational components, such as user interfaces or file processing, may be created using different languages (like C/C++), if this does not inhibit the platform-independence of the solver. The next chapters will justify this decision. According to eNLP-M-011, the coding language will be English.

2.4.1.1 Performance

Fortran was created to serve as a puristic, high-performance, platform-independent computational programming language. Because of this, Fortran has clear performance advantages over general-purpose languages such as C/C++.

A performance study at the SRC has revealed (in parts drastic) performance differences between Fortran and C++ code for a standard numerical task. The main tasks of an NLP solver are multiplying and adding. Therefore a matrix-matrix multiplication serves as a good measure because several multiplications and additions have to be performed.

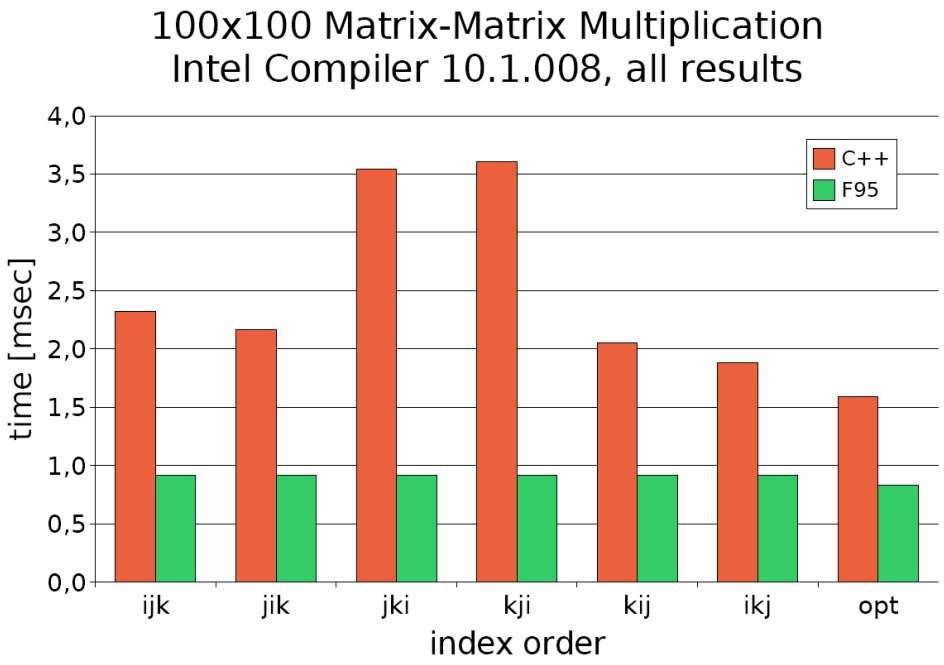

Figure 2-7: Fortran 95 vs. C++

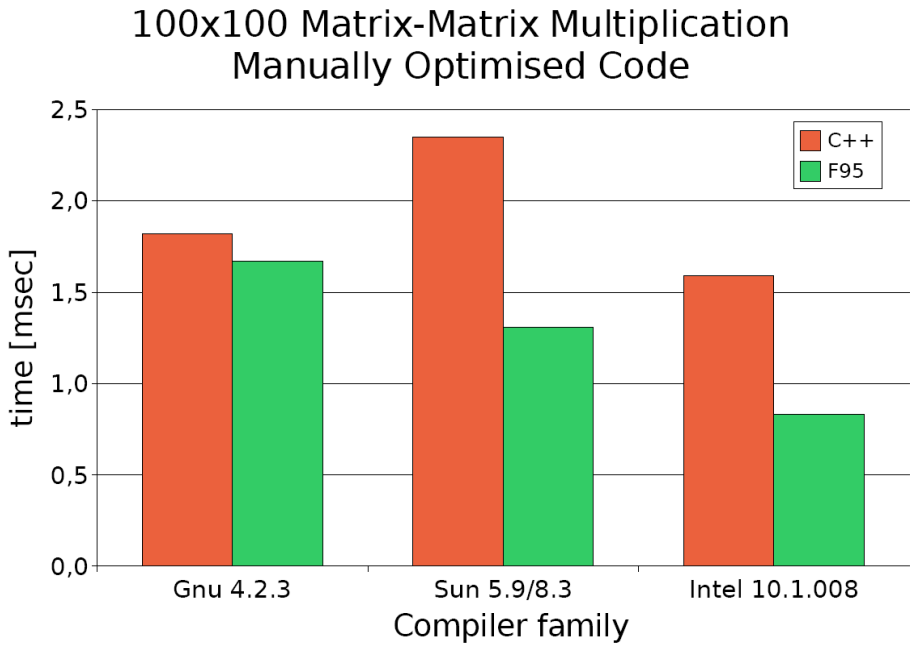

Figure 2-8: Fortran 95 vs. C++

The plots above show the average 5-best timings of a 1000-run sample of C++ and Fortran 95 matrix-matrix-multiplication code of two 100x100 matrices populated with random values; the time for random value generation is not measured.

The upper figure shows the timings for the six possible index orders and the timings for manually optimised code, compiled with the Intel Linux C++ and Fortran compiler 10.1.008.

The striking speed advantage of Fortran is due to at least two differences: Thanks to the stricter standard of Fortran 95, the compiler has been able to reorder the Fortran loop into the optimal index order for fastest memory access, and make better use of the processor cache hierarchy.

By manual optimization (such as loop unrolling, use of intrinsics, direct pointer manipulation etc.), the timings could be improved by similar factors for both languages. It should be noted that the manually optimized Fortran code is by far more expressive and maintainable than its C++ counterpart.

The lower figure shows only the timings of the manually optimised code, compiled with current versions (as of Q1 2008) of three important compiler families. The two rightmost bars coincide with the two rightmost bars of the upper plot.

It is noticeable that, although the Intel and the GNU compiler seem to share significant parts between the Fortran and C++ frontends, they still yield the demonstrated timing differences.

A further note on the GNU compiler: While the C compiler reaches back to as far as 1987, the Fortran 95 frontend has only been released in 2005; we therefore expect that the GNU compiler still has lots of potential for improved performance.

2.4.1.2 Fortran Compilers

Fortran compilers, also free ones, are widely available for all relevant platforms. Long-term support and availability of compilers is guaranteed by the ongoing demand of the scientific community, among them the ever-growing High-Performance-Computing (HPC) community, for high-quality compilers for legacy, mature and new Fortran applications. Because of the language design as a purely computational language, Fortran compilers offer

more expressive diagnostics,

more aggressive code optimization,

higher stability and

better automatic parallelisation than compilers for most other languages.

2.4.1.2.1 Free Compilers

There are three major compilers freely available

gfortran – the Fortran frontend of the GNU compiler collection gcc,

f95 – the Fortran 95 compiler of the freely available Sun Studio,

g95 – a gcc branch, maintained as single freestanding compiler.

Of the free compilers, gfortran will be used heavily in the NLP development, for two main reasons: Its availability for all major and even exotic platforms, and its long-term support.

The gcc has been ported to over 60 different platforms, making it the most portable compiler world-wide. Every Linux user will have the compiler available as a standard compiler, while Windows and Mac users can easily find installation instructions and download sources for packages at the gcc website.

The gcc has an active development history reaching back over 20 years, a stable community of developers, and aims to provide service releases every two to three months. It is backed by the Free Software Foundation (FSF), which provides different kinds of support to the development community. The gcc is furthermore an integral part of all Linux distributions, which further ensures its long-term availability.

The Sun f95 compiler comprises a very mature codebase and offers good code optimization. It has been freely available only since 2007, as it comprises a central component of the formerly commercial Sun Studio. The Sun f95 compiler is only available for a few major Linux distributions and Solaris. Its long-term availability (not its free availability, however) is virtually guaranteed, since compilers play a key role for global players in the IT business like Sun Microsystems.

The g95 compiler is a 2003 branch of the gcc and is essentially maintained by a single enthusiast. It shares large amounts of its codebase and therefore virtually all of its functionality with the gcc, and is available for all major platforms. Its long-term support and availability is questionable, given the small developer "community" who maintain g95.

2.4.1.2.2 Commercial Compilers

There are a number of commercially available compilers, most of which are compiler "families" of Fortran and C/C++ compilers: (this list is not meant to be complete)

IBM – offering the xl* compilers, specially suited for their HPC architectures,

PGI – the Portland Group, also offering compilers specially suited for HPC ap-

plications,

nag – offering highest-quality C and Fortran libraries and compilers,

intel – producing state-of-the-art x86 CPUs and tailored compilers.

All commercial compilers have in common that they offer professional support by experts, high degrees of error checking and code optimization, thus producing efficient and stable applications and aiding the developers in producing high quality code.

The perspectives of long-term support and compiler availability are excellent, since there is an ongoing need for high quality Fortran compilers, not only in the HPC community, but in the whole scientific community running Fortran for all kinds of computational applications.

2.4.1.2.3 Compilers to be used in NLP Development

The development of the NLP solver will be driven by milestones. This implies the selection of a compiler family to drive the development and debugging efforts. Because of its extreme portability, its guaranteed free availability, high cross-language integration and good stability properties, the gcc compiler family will serve as a tool for judging and ensuring code portability and standard conformity. The rationale for this is that any platform-independent code that compiles and runs without errors when compiled with the gcc is virtually guaranteed to work on any given platform supported by the gcc.

A further benefit is that the code is tested to work with a compiler available to every potential user free of charge. With the gcc's main emphasis being on portability, many compilers offer better runtime performance on a small number of platforms. Performance checks involving timing will therefore be performed using the best-performing compiler available to the developers for a specified platform. Similar considerations will be made for the creation and delivery of the compiled solver library.

Other compilers, subject to their availability to the developers, will be used intermittently to check for standard conformity and portability, and to ensure that the NLP solver is supported by wide range of compilers. This way, many users may use their favourite compiler (if this is not gcc) to link the NLP solver to their application.

2.4.1.3 Portability and 64-bit Compatibility

The developers of the NLP solver will take advantage of the platform- and compiler-independent facilities in Fortran to ensure the portability of the solver between various platforms, like Microsoft Windows, Linux, MAC or Solaris, and between 32-bit and 64-bit systems.

The requirements eNLP-M-003 and eNLP-M-004 demand the solver to be compatible with major platforms (Windows, Linux, MAC, Solaris) and with 64 bit platforms. Fortran 95 is highly suitable to this end, since the standard defines it as a numerical language to work reliably, independent of the underlying computer architecture, operating system or compiler; this sets Fortran apart from other languages often used in computational applications.

This feature of Fortran is reflected by the fact that the ISO/IEC 1539:1997 standard defines mandatory language facilities for explicitly platform- and compiler-independent choices of datatypes and required numerical range or precision, which are supported by all modern Fortran compilers.

Any Fortran code that uses these facilities can thus be compiled on either 32- and 64-bit platforms, running arbitrary operating systems and hardware architectures, without any changes to the code.

When regarding C/C++ in this respect, it is clear that due to their origin as all-purpose low-level system programming languages no such level of portability can be achieved without rather painful efforts, involving platform-dependent conditional preprocessor datatype macro definitions and similar approaches used to overcome the lack of rigorous datatype standardisation in C/C++.

2.4.1.4 Maintainability

For every long-term software development project, maintainability must be considered a major issue, due to the high complexity of such software and the potential fluctuations in the development staff. It is redundant to mention that thorough documentation is a (language-independent) key element in achieving long-term maintainability. When concerning language-dependent features, Fortran is an excellent choice for creating maintainable computational code.

An important language-dependent element of maintainability is the expressiveness of the code. Expressiveness is higher, when standard operations can be executed using expressive high-level (preferably intrinsic) language facilities, instead of elemental bit/byte/element-wise operations.

A good example to illustrate this point is the intrinsic MATMUL function in Fortran, which multiplies any conforming combination of two given matrices or vectors. Most other languages need two or three nested for-loops to achieve this, and probably need to specialize between vector-vector, matrix-vector, … operational mode, while this is done internally and most efficiently by Fortran compilers.

Since Fortran is inherently vectorized, with arrays enjoying the status of being a datatype, a great number of expressive, flexible and high-performance intrinsic routines are defined by the standard. For C/C++, similar functionality can be

achieved by linking against third-party libraries (e.g. Blitz++). This is unnecessary in Fortran, since it is an integral part of the language standard.

Maintainability is also influenced by the ability to adapt existing code to new trends in software and hardware engineering. One major foreseeable trend for (at least) the next decade is increasing hard- and software parallelisation.

Fortran is probably one of the most suitable languages to meet and exploit this trend: having been used on mainframes and the first parallel supercomputers in its younger days, Fortran has been influenced by the needs and challenges of parallel computation before any other language.

With Fortran being an inherently vectorized language providing efficient high-level intrinsic array manipulation routines, modern compilers can auto-parallelise well-written code to a great extent, without the need to make any changes to the code. By introducing few and painless adaptions, parallelisation can be increased even further by methods that fit seamlessly into existing Fortran applications.

2.4.2 Memory Reducing Strategy

Handling large scale problems means always facing a huge memory requirement. This section describes the strategies how to tackle this problem.

2.4.2.1 Matrix Storage Format

Sparse matrices have the property that they can be stored efficiently by just saving value and position of each nonzero-entry. The position of the nonzero entries is also called *sparsity pattern.*

The evaluation of a sparse matrix-vector-product and the decomposition into LU factors can also be done in a very efficient way by ignoring all zero entries. Note that in this respect a “nonzero entry” is an entry which is not zero by definition because of a deficient dependency. Of course a “nonzero entry” can occasionally become zero during the optimization process.

2.4.2.1.1 Compressed Column Storage

In the compressed column format the matrix is passed column-wise and stored in this order into the double precision array VAL of the length DIM, where DIM denotes the number of nonzero entries. In the same way as in the coordinate storage, the DIM-dimensional array ROW contains the row indices of each value, here in the column-wise order. The array COL contains N+1 variables where N indicates the number of columns in the matrix. The i-th value of COL denotes the index with which the i-th column starts. The N+1st entry is always DIM+1 and facilitates many operations.

Transposing with this format is quite complicated but single columns can be selected very easily. The memory requirement is just 2*DIM+N+1.

The compressed Column storage format is the state of the art and the most commonly used format. Hence while developing a new solver it is very sensible to use this format. Most of the test sets use this format as well so that interfaces to such test sets can easily be established.

2.4.2.1.2 Coordinate Storage

The coordinate storage format might be more intuitive than the compressed column format. However, this is only an advantage for the programmer and should not be decisive. The crucial point is that it needs more memory requirement (3*DIM) and is less common.

2.4.2.2 Workspace Reutilization

Besides exploiting the sparsity with the compressed storage, another strategy to reduce the amount of used memory will be the workspace reutilization.

A significant amount of integer and double precision workspace for matrix operations and internal functionality will be used. The main task of this strategy is to identify the workspace partitions that may be reused in different stages of an iteration or by different internal routines. While making it possible to lower the memory footprint of the solver, great care has to be taken when performing this improvement. The difficulty is not to introduce any errors which are hard to identify and could lead to a non-deterministic program behaviour.

2.4.3 Interface Definition

The large scale sparse NLP-solver WORHP and its data structures, provided by the SRC, will serve as interfacing hub between the user, WORHP SQP, IP-Filter IP and the solver back-end functions. This places great significance on the interface which needs to be clean, concise, easy to use and computationally efficient on the one hand, but also flexible, modular, debugging-capable, mixed-language-capable and extendable on the other. These requirements are formulated in detail in eNLP-M-005, eNLP-U-001 and eNLP-U-002.

2.4.3.1 Unified Interfaces

The WORHP interfaces will make heavy use of four dedicated solver data structures for the following purposes:

problem variables,

solver workspace and working variables,

solver parameters,

solver control flow.

Consistent use of these data structures will enable the creation of a Unified Interface for all high-level internal and external functions, which greatly promotes modularity and simplicity, since arbitrary high-level routines may be used interchangeably at every point of the code, where these routines are visible.

It also simplifies debugging, since all significant solver activities need to be communicated and performed through these data structures. This modularity is a strong requirement, as formulated in eNLP-M-005.

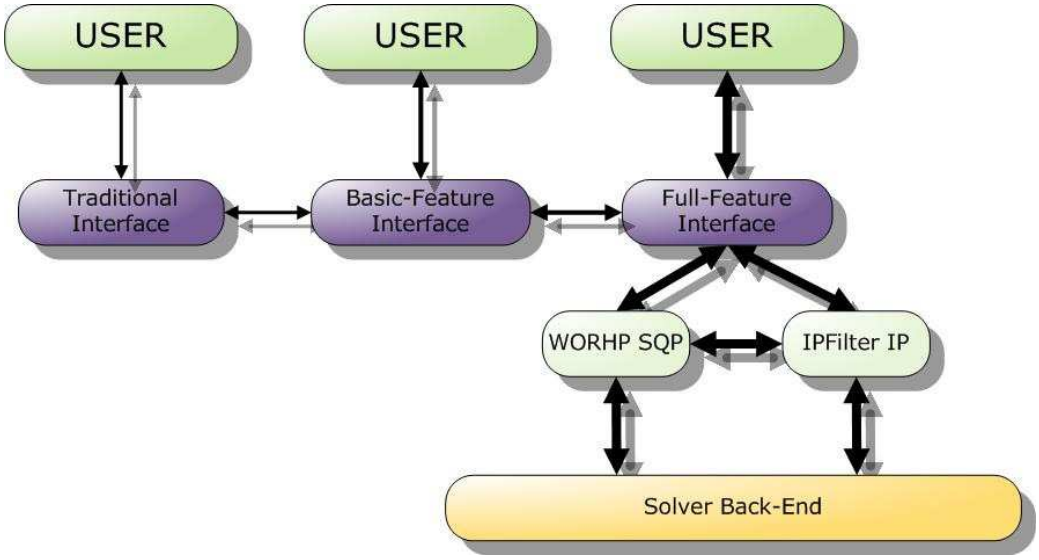

Figure 2-9: Interfaces

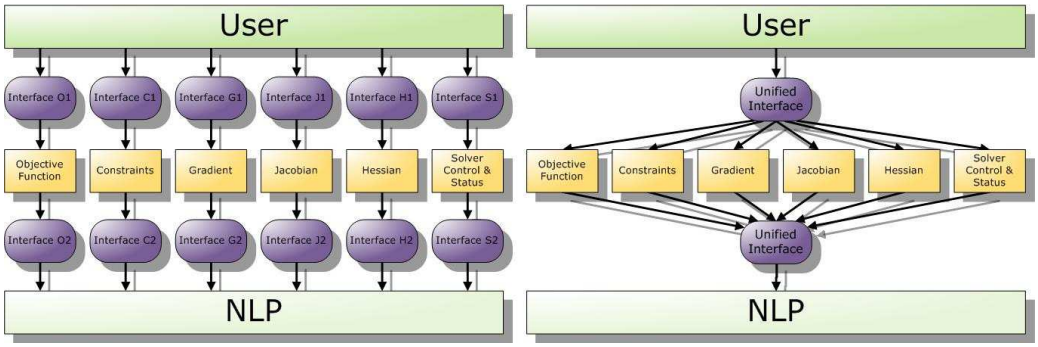

Figure 2-10: Interfaces Comparison

2.4.3.2 Full-Feature Interface

The solver will provide an interface that, by means of reverse communication and consistent use of the dedicated data structures, exposes all relevant internal communication, data and control flow steps to the user.

This interface will meet the requirements eNLP-U-001 and eNLP-U-002 in that it will be very simple and clearly structured, despite offering the flexibility to extract all kinds information useful in debugging.

2.4.3.3 Basic-Feature Interface

The solver will provide an interface that encapsulates the reverse communication calling process, while still exposing all relevant internal data to the user on entry and exit. In this aspect, it is oriented towards more traditional NLP interfaces and calling conventions.

It constitutes an even further simplification of the Full-Feature interface, and requires the user to use solver-prescribed interfaces for the user's objective and constraint functions.

2.4.3.4 Traditional Interface

The solver will offer a traditional interface, which will conform with the interface of an existing NLP solver that is widely used, as required by eNLP-I-001. The traditional interface will simplify linking existing software to the solver, while not offering any of the benefits of the Full-Feature or Basic-Feature interfaces.

2.4.4 Software Development Techniques

The solver will be developed, according to eNLP-M-001. In the following sections some more details are considered.

2.4.4.1 Version control

The distributed development of highly complex software, like the NLP solver, by multiple partners in different locations poses great challenges to software development techniques. Besides the necessary measures on a management level, technical measures will be taken to support the developers in managing and archiving different code versions, minimising merge errors and resolving conflicting changes to the code.

Subversion, being one of the currently most widely used softwares packages for version control, will be used to establish source code version control and to support the collaboration of the developers. Most importantly, it allows defining and archiving the milestone versions used to measure development progress. The related technical details will be arranged between the project partners.

2.4.4.2 Debugging

As soon as a bug is detected or reported, this bug will be tried to fix as soon as possible. It is clear that bugs will occur during the development. Every bug that occurs will be considered as chance to improve the solver. Every occurring bug will be documented in order to facilate fixing similar bugs.

2.4.4.3 Progress Tests (2-3 months) /Milestone /Version Number

Testing will be done continuously. Nevertheless from time to time there will be a frozen version that shall be tested extensively. These versions will be given version numbers and serve as milestones in the development.

2.4.4.4 Communications

It is clear that throughout the whole activity there will be a close cooperation between all consortium partners that will enforce a lot of communication.

3 MIDDLE AND LONG TERM DEVELOPMENT

As this activity is restricted to 18 months, not all methods, concepts and ideas can be implemented within this time period. The developers have to restrict themselves to those modules which seem to guarantee the best result for this time period. The justification for this distinction has been given in the proposal. However, this chapter shall give a short outlook which strategies could be pursued in future development phases.

3.1 SOLVER STRATEGIES

3.1.1 Constraints Oriented Strategies

3.1.1.1 Active Set Methods

The idea of the primal active-set strategy for (QP) is to estimate the (unknown) set of active inequality constraints and adapt this estimate iteratively. In each iteration an auxiliary QP with equality constraints has to be solved. If the KKT matrix is positive definite on the null space of the constraint matrix, solving this QP is equivalent to solving a sparse linear equation system

A drawback of the primal active-set method is that it requires a feasible starting point. Alternatively, one can use a dual active-set method for strictly convex problems which does not require a feasible starting point.

As an additional module, the solvers could be enriched by such a strategy.

3.1.1.2 Exterior Active Set

Typically, more than half of a problem's dimension is due to inequality constraints. An exterior active set strategy determines and keeps track of active and inactive constraints, eliminating the inactive ones before passing them to the solver. This is supposed to significantly lower the computational costs of solving large-scale inequality constrained problems.

The crucial task is to determine the inactive constraints that will not be violated when they are neglected for one or more iterations. In order to reduce computational costs, this would be a helpful tool.

3.1.1.3 Strong IP

In an interior-point method, the Lagrange multipliers tend not to be zero, even if they belong to inactive constraints. Hence the strategy of a strong IP method is to set those multipliers to zero. Again, it is a difficult task how to determine inactive (and active) constraints. However, if this is done successfully one can save computation time considerably.

3.1.2 Method Oriented Strategies

3.1.2.1 Trust-Region

The Trust-region SQP a search direction is given by the solution of a standard QP problem, where the length of the resulting search direction is restricted to a given trust-region radius, which is adjusted in each iteration of the SQP method by comparing the actual reduction in a merit function and the predicted reduction in a model function. In contrast to the line-search version of SQP, the KKT matrix is allowed to be indefinite in the trust-region setting.

Unfortunately, the additional constraint may cause the QP problem to be infeasible. To avoid this difficulty, one can replace it by an augmented QP problem-which is always feasible.

Another approach to approximately solve the trust region QP problem is to use a two stage procedure. Firstly, a normal step is computed by minimizing the residual norm of the linear constraints subject to the trust-region constraint. This can be done efficiently by the dogleg method, which combines a Cauchy step and a Newton step appropriately.

Secondly, a tangential step is computed by solving a second optimization problem using a projected preconditioned conjugate gradient method. This approach is implemented in the code KNITRO.

Again, the Maratos effect may occur and can be avoided by second-order corrections. The trust-region version (compared to the line-search version) is less restrictive with respect to the choice of the KKT matrix as indefinite matrices are allowed. Similarly, at least for the second version of the trust-region method, linearly dependent constraints are not a severe problem.

Trust region SQP methods have attracted a lot of attention lately, and have proven to yield robust and reliable algorithms, eliminating a number of problems that the standard methods have to cope with. To broaden the functionality of the solvers, a Trust Region module could be implemented.

3.1.2.2 Piecewise linear l_2 penalty function

A cutting-edge result from optimization theory is the piecewise linear l_2 penalty function devised by Chen and Goldfarb, which builds a bridge between the traditional and well-understood penalty function and the upcoming Filter methods. First results suggest that in many cases it may be superior to Filter methods.

Since the developers are geared to use the latest developments in large-scale nonlinear optimization, this method should be investigated and integrated as an additional measuring progress module at a later stage.

3.1.2.3 Augmented Lagrangian Method

Augmented Lagrangian methods (or multiplier methods) replace the standard NLP problem) by successive unconstrained minimization with respect to x of the so called augmented Lagrangian function which is continuously differentiable and exact.

The minimization can be done by methods for unconstrained minimization, e.g., by Quasi-Newton methods with suitable adaptations for sparse problems.

This is an entire different approach for solving an NLP problem. It might be an option to implement such a method in order to solve some problems that cannot be solved by WORHP or IPFilter.

3.2 NECESSITIES STRATEGIES

3.2.1 Optimal Transcription Determination

The developed software is expected (although being a general-purpose NLP solver) to be used with two major discretization techniques of optimal control problems: Full discretization generates very large and very sparse problems, while multiple shooting generates much smaller but notably denser problems.

A combined technique of fully discretized blocks linked by multiple shooting nodes results in moderate problem size and sparseness. It is expected that for every problem there is an optimal block size, i.e. a ratio between both methods, such that the solver yields the best possible performance.

A future goal is to investigate the combined discretization method and to find optimal block sizes.

3.2.2 Extended Sparsity Strategy

3.2.2.1 Online Sparsity

The user is not always able to give minimal gradient/Hessian sparsity structure. Even when they are given, they may (and usually do) contain a fraction of additional zeros. If this fraction is non-negligible, scanning the matrices online to further reduce the sparseness can lower the computational times and memory footprint of the solver.

However, this is quite a complicated task as it can clearly occur that a value is untruly handled as zero. This has to be checked a reasonable amount of times. Every check will demand computation time and every restructuring will cause problems with the memory. Hence this concept does not suite in the coming development phase.

3.2.2.2 Automatic Differentiation Sparsity Framework

Automatic Differentiation (AD) provides an analytical form of a derivative of a given numerical routine with respect to given variable. However this technique is not always applicable.

Many applications do not provide appropriate numerical routines due to the chosen programming language or the use of tabular values (e.g. ASTOS). Thus the software only returns the demanded value and cannot be investigated algorithmically by the solver.

Hence AD cannot be the (only) method for providing derivatives in an NLP solver that is supposed to solve generic problems. However it might be a good additional module for the future.

Furthermore AD has some more advantages. Extracting and formulating the structure of the objective function and the constraints is cumbersome and may be very difficult, especially so for large-scale problems. Using (upper) approximations of structures is unsatisfactory since it affects performance. OOP methods derived from existing AD techniques can be used to automatize this process and find exact dependency structures. On various levels of sophistication linear versus nonlinear dependency or even analytic derivatives can be supplied, resulting in higher performance and precision.

3.2.3 First Order Derivatives

In trajectory optimization, other methods to approximate the derivatives are to formulate sensitivity differential equations. However these have to be formulated individually for every problem and again, analytical expression are needed.

Sensitivity differential equations depend strongly on the application and do not satisfy the demand to be applicable to generic NLP problems. Therefore, again, this can only be an alternative, not the only way but it might be worth as an additional module.

3.2.4 Second Order Information

3.2.4.1 Sufficient Optimality Conditions Check

Generally, NLP solvers seek points that satisfy necessary optimality conditions. Actually these are also fulfilled when a feasible point maximises the objective function although a minimum is wished.

Hence it might be sensible to check also the sufficient optimality conditions. The problem about those is that they do not have to be fulfilled in an optimal point and actually do quite rarely.

However, it would be a nice tool as an additional check that, if positive, guarantees an optimal solution.

3.2.4.2 Post-Optimality Analysis

After the solver as aborted, from the given variables, derivatives etc., one can still get additional information. For example how the optimal solution would react to some perturbation in some problem input data. This leads to the theory of sensitivity analysis and would give the solver an entirely new functionality which will cause a lot of work and is therefore not foreseen in the coming development phase.

3.2.4.3 Sparse BFGS Update

As already mentioned in section 2.3.2.4.2, BFGS Updates are commonly used in nonlinear optimization to approximate the Hessian matrix. They lead to a worse convergence than an exact Hessian but their computational costs are much lower so that it is worth accepting this. As they generally lead to dense matrices this seems not to hold for sparse problems. A normal BFGS would lead to unacceptable memory costs.

There are current investigations about sparse BFGS updates. If there arise promising results, such updates might become part of the solvers in the future.

3.3 SOFTWARE STRATEGIES

3.3.1 Stand Alone Linear Algebra

In order to minimize the source of potential errors we will use commercial linear equation solvers like the multi-frontal solvers MA47, MA48, and MA57 from the HSL library for direct solution of linear equations. We expect a better performance by using these commercial tools. SuperLU was mainly chosen because it is freely available. It will be replaced in the future.

In a further project these solvers shall be replaced by our own developments. This is a very challenging task and hence not sensible to be fulfilled within this activity.

3.3.2 Parallelisation

Many of the computational routines that will be contained in the solver may be parallelised, especially (but not limited to) the linear algebra and the finite difference routines.

With today's hardware standard tending towards multi-processor machines, a complex and involved computational routine like an NLP solver should exploit the benefits of widely available parallel hardware in the future.

3.4 COMMERCIALIZING

A future goal of this activity is of course to commercialize the developed software in as many areas as possible, first of all in space applications, of course. This will be kept in mind in every stage of development and therefore at some points the purely academic position has to be left.

On the other side, academia plays an important role at it makes possible that brand new developments can be implemented and improve the solvers.